Applications for Symbol Elimination in Combination with Hierarchical Reasoning

by

Dennis Peuter

Approved Dissertation thesis for the partial fulfillment of the requirements for a Doctor of Natural Sciences (Dr. rer. nat.)

Fachbereich 4: Informatik
Universität Koblenz

Chair of PhD Board:	Prof. Dr. Ralf Lämmel
Chair of PhD Commission:	Prof. Dr. Maria A. Wimmer
Examiner and Supervisor:	Prof. Dr. Viorica Sofronie-Stokkermans
Further Examiners:	Prof. Dr. Silvio Ghilardi
	Prof. Dr. Pascal Fontaine

Date of the doctoral viva: 3. April 2024

Summary

The goal of this PhD thesis is to investigate possibilities of using symbol elimination for solving problems over complex theories and analyze the applicability of such uniform approaches in different areas of application, such as verification, knowledge representation and graph theory. In the thesis we propose an approach to symbol elimination in complex theories that follows the general idea of combining hierarchical reasoning with symbol elimination in standard theories. We analyze how this general approach can be specialized and used in different areas of application.

In the verification of parametric systems it is important to prove that certain safety properties hold. This can be done by showing that a property is an inductive invariant of the system, i.e. it holds in the initial state of the system and is invariant under updates of the system. Sometimes this is not the case for the condition itself, but for a stronger condition it is. In this thesis we propose a method for goal-directed invariant strengthening.

In knowledge representation we often have to deal with huge ontologies. Combining two ontologies usually leads to new consequences, some of which may be false or undesired. We are interested in finding explanations for such unwanted consequences. For this we propose a method for computing interpolants in the description logics $\mathcal{EL}$ and $\mathcal{EL}^+$, based on a translation to the theory of semilattices with monotone operators and a certain form of interpolation in this theory.

In wireless network theory one often deals with classes of geometric graphs in which the existence or non-existence of an edge between two vertices in a graph relies on properties on their distances to other nodes. One possibility to prove properties of those graphs or to analyze relations between the graph classes is to prove or disprove that one graph class is contained in the other. In this thesis we propose a method for checking inclusions between geometric graph classes.

Zusammenfassung

Das Ziel der vorliegenden Doktorarbeit ist die Untersuchung von Möglichkeiten zur Anwendung von Symbolelimination, um Probleme über komplexen Theorien zu lösen, sowie die Analyse der Anwendbarkeit solcher einheitlichen Ansätze in verschiedenen Anwendungsbereichen, beispielsweise in der Verifikation, Wissensrepräsentation oder Graphentheorie. In der Arbeit stellen wir ein Verfahren für die Symbolelimination in komplexen Theorien vor, welches der generellen Idee folgt, hierarchisches Schließen mit Symbolelimination in Standardtheorien zu verbinden. Wir untersuchen, wie dieser allgemeine Ansatz spezialisiert werden kann, um ihn in verschiedenen Anwendungsgebieten zu verwenden.

In der Verifikation parametrischer Systeme ist es wichtig zu beweisen, dass bestimmte Sicherheitsbedingungen erfüllt sind. Dies kann erreicht werden, indem man zeigt, dass eine Bedingung eine induktive Invariante des Systems ist, das heißt, dass sie im Anfangszustand des Systems erfüllt ist und unter Veränderungen des Systems erhalten bleibt. Manchmal ist dies für die Bedingung selbst nicht der Fall, aber für eine stärkere Bedingung schon. In der Arbeit stellen wir eine Methode zum zielgerichteten Verstärken von Invarianten vor.

In der Wissensrepräsentation wird häufig mit großen Ontologien gearbeitet. Das Zusammenfügen zweier Ontologien bringt für gewöhnlich neue Konsequenzen mit sich, von denen einige womöglich fehlerhaft oder unerwünscht sind. Wir sind interessiert daran, Erklärungen für das Auftreten solcher unerwünschten Konsequenzen zu finden. Zu diesem Zweck stellen wir eine Methode vor, mit der Interpolanten in den Beschreibungslogiken $\mathcal{EL}$ und $\mathcal{EL}^+$ generiert werden können, basierend auf einer Übersetzung zur Theorie der Halbverbände mit monotonen Operatoren und einer bestimmten Form von Interpolation in dieser Theorie.

In der Forschung zu kabellosen Übertragungsverfahren beschäftigt man sich mit Klassen von geometrischen Graphen, bei denen die Existenz oder Nicht-Existenz einer Kante zwischen zwei Knoten im Graph von Eigenschaften auf deren Distanzen zu anderen Knoten abhängt. Eine Möglichkeit, um Eigenschaften solcher Graphen zu beweisen oder Beziehungen zwischen zwei Klassen zu analysieren, ist zu zeigen, dass eine Klasse in der anderen enthalten ist. In dieser Arbeit stellen wir eine Methode vor, um die Inklusion einer Klasse von Graphen in einer anderen zu überprüfen.

First and foremost I would like to thank my advisor, Prof. Dr. Viorica Sofronie-Stokkermans, for her supervision and guidance over the past few years. With her support of my research she made this thesis possible in the first place. In addition to giving me helpful scientific advice, she always had understanding for any personal matters in my life, for which I am very grateful.

I would also like to express my gratitude to my other examiners, Silvio Ghilardi and Pascal Fontaine, for their appreciation of my work and their helpful feedback.

Furthermore I would like to thank Hannes Frey and Lucas Böltz for their collaboration and discussions about geometric graph theory.

Many thanks also go to Philipp Marohn and Sebastian Thunert for providing implementations of the methods used in this thesis.

Last but not least, I want to thank all family members and friends who have supported me throughout my life.

Contents

1 Introduction

Symbol elimination is used in mathematics and symbolic computation, in verification (interpolation, projection, computing reachable states), in knowledge representation (communication, forgetting), and in logic (theorem proving). The goal is to eliminate certain symbols (variables, function symbols or predicate symbols) of a logical formula and obtain formulae over such simpler languages which are "optimal" in some sense. Depending on the properties of these formulae, we obtain various forms of symbol elimination. One possibility is to eliminate certain symbols from a formula such that the newly generated formula is in some sense equivalent to the original one. However, a weaker form of symbol elimination is also possible. Sometimes we may not need a formula that is equivalent to the original formula, but we would only like to ensure that it is entailed by it and has additional "good" properties with respect to a given other formula or to a class of formulae over a given signature. We can thus distinguish two forms of symbol elimination:

(a) Property-directed symbol elimination

(b) General symbol elimination

In property-directed symbol elimination we are given a formula F from which we want to eliminate certain symbols and an additional formula G which is meant to guide the process of elimination. In this thesis we distinguish two different ways in which property-directed symbol elimination may be used:

(1) Find a formula F' over a simpler language (i.e. not containing certain symbols) which is entailed by F such that some desired property, usually related to the formula G, holds.

(2) If the conjunction of F and G is satisfiable, find a formula Γ only containing a certain subset of the symbols in F and G such that the conjunction of F, G and Γ is unsatisfiable, i.e. a constraint over a given language which guarantees unsatisfiability of F in conjunction with G.

Property-directed symbol elimination as in (1) is for instance important when computing an interpolant. Given two formulae A and B such that A entails B and they have some symbols in common, we say that a formula I which only contains shared (or common) non-logical symbols is an interpolant of A and B if A entails I and I entails B. We can compute such interpolants by eliminating non-shared symbols from A such that the property "I entails B" holds. Alternatively, the problem of interpolation can also be stated in the following way: Given two formulae A and B where $A \wedge B \models \bot$, find a formula I containing only symbols shared by A and B such that $A \models I$ and $I \wedge B \models \bot$.

As an example for (2), assume that we have a description of a system using variables $x_1, ..., x_n$, where some safety property of the system is described by a formula $F(x_1, ..., x_n)$, and the updates of the system are described by a formula $\mathsf{Tr}(x_1, ..., x_n, x'_1, ..., x'_n)$, where the primed variables describe the values of the corresponding non-primed variables after the

update. Using property-directed symbol elimination we can then compute additional conditions on a subset of the function symbols (considered to be underspecified to start with) under which the system is safe. For this we compute a formula Γ over a simpler language (for instance not containing $x'_1, ..., x'_n$) such that $\Gamma \wedge F(x_1, ..., x_n) \wedge \mathsf{Tr}(x_1, ..., x_n, x'_1, ..., x'_n) \wedge G$ is unsatisfiable, where $G = \neg F(x'_1, ..., x'_n)$. In this setting we usually are interested in computing the weakest such formula Γ.

The problem of general symbol elimination is different. Here we only have a formula F given, without an additional formula G to guide the process. The goal of general symbol elimination is to find a formula over a simpler language that is in some sense equivalent to the original formula.

General symbol elimination can for instance be used to compute a stronger kind of interpolant than the one described above, namely a uniform interpolant. Given a formula A, we say that a formula I is a uniform interpolant of A if I is an interpolant of A and B for any formula B that is entailed by A and has only the non-logical symbols occurring in I in common with A. In contrast to an interpolant, a uniform interpolant is not dependant on the formula B and therefore general symbol elimination on A can be used for its computation rather than property-directed symbol elimination.

As another example, consider a class of graphs axiomatized by geometric conditions between points $(x_1, ..., x_n)$, the vertices (V) and the edges (E). These geometric conditions are described by a formula $F(x_1, ..., x_n, V, E)$. In addition we could have a formula $\mathsf{Tr}(E, E')$ describing a transformation on the edges of the graphs to form another graph class. If we are interested in computing an axiomatization for this class of transformed graphs which is based solely on geometric conditions and the edges E' of the transformed graph, then we have to eliminate the symbol E to obtain a formula equivalent to $\exists E\ F(x_1, ..., x_n, V, E) \wedge \mathsf{Tr}(E, E')$.

Methods for symbol elimination are usually applied to formulae of propositional logic or first-order logic or in logical theories (arithmetic). In the thesis we will also use similar ideas for symbol elimination in description logics, by first translating the given problems to propositional or first-order logic, where quantifier elimination or methods for interpolation can then be used.

1.1 Illustration

We first illustrate our ideas on short examples from three different application areas: inductive verification and synthesis, interpolation in description logic and reasoning about geometric graph classes. A description of the used methods and a detailed discussion of the examples will be given later in the thesis.

Inductive Verification and Synthesis

In program verification it is important to check that programs work correctly. One way of doing this is to show that some conditions are invariant, i.e. they hold at every point during the execution of the program. A formula is an inductive invariant if it holds in the initial state of a system and is preserved during updates of the system. If a formula cannot be proved to be an inductive invariant, one possibility is to strengthen it until it becomes an inductive invariant.

In the following we show an example of invariant strengthening.

Consider the program in Figure 1.1. It uses two subprograms $\mathsf{copy}(a, b)$ and $\mathsf{add1}(a)$ with the following meaning:

- $\mathsf{copy}(a, b)$ copies the array b into array a.
- $\mathsf{add1}(a)$ adds 1 to every element of array a.

Assume that b is an array with its elements sorted in increasing order. The task is to prove that then the formula $\Psi \coloneqq d_2 \geq d_1$ is an inductive invariant of the program.

```
d1 = 1;
d2 = 1;
copy(a, b);
i = 0;
while (nondet())
{
   a = add1(a);
   d1 = a[i];
   d2 = a[i+1];
   i = i + 1
}
```

Figure 1.1: A simple program.

For checking that Ψ is an inductive invariant we have to show two things:

- Ψ holds in the initial state, i.e. before the while loop.
- Ψ is an inductive invariant of the while loop, i.e. if it holds before an iteration of the while loop, then it also holds afterwards.

Ψ clearly holds in the initial state, since before the while loop we have $d_2 \approx 1 \geq 1 \approx d_1$.

Ψ is an inductive invariant of the while loop if and only if the formula

$$\forall j(a'[j] \approx a[j] + 1) \wedge d_1' \approx a'[i] \wedge d_2' \approx a'[i+1] \wedge i' \approx i + 1 \wedge d_1 \leq d_2 \wedge d_1' > d_2'$$

is unsatisfiable. As it can be checked that this formula is satisfiable, Ψ is not an inductive invariant. However, we can strengthen $\Psi \coloneqq d_2 \geq d_1$ with the condition $\forall i(a[i] \leq a[i+1])$ to make it become an inductive invariant. Indeed, the formula

$$(d_2 \geq d_1) \wedge \forall i(a[i] \leq a[i+1])$$

is an inductive invariant of the program.

In Chapter 4 we present a method for computing inductive invariants by iteratively strengthening them like shown above. In Section 4.2 we discuss the example above in more detail.

Interpolation in Description Logic

Description logics are used for modeling and reasoning in knowledge bases. Classes of objects are described by concepts and relationships between objects by roles. Relationships between concepts or roles are modeled using ontologies, which can contain concept definitions, concept inclusions and role inclusions. A special class of description logics, which were proved to be tractable [4, 5, 8], are $\mathcal{EL}$ and its extension $\mathcal{EL}^+$. They both allow existential role restrictions and intersection of concepts; the latter also allows role inclusions.

When combining two ontologies or extending an ontology with another one it can happen that some unwanted concept subsumptions can be derived afterwards. For fixing such unwanted consequences it is important to locate their cause. An important goal is therefore to find explanations for subsumption relations in combined or extended ontologies. We here present an example.

$$
\begin{array}{rrcl}
A_1: & \mathsf{AmpOfFinger} & \sqsubseteq & \mathsf{Amputation} \\
A_2: & \mathsf{AmpOfFinger} & \sqsubseteq & \exists \mathsf{site.FingerStructure} \\
A_3: & \mathsf{Amputation} \sqcap \exists \mathsf{site.FingerStructure} & \sqsubseteq & \mathsf{AmpOfFinger} \\
A_4: & \mathsf{InjToFinger} & \sqsubseteq & \mathsf{Injury} \\
A_5: & \mathsf{InjToFinger} & \sqsubseteq & \exists \mathsf{site.FingerStructure} \\
A_6: & \mathsf{Injury} \sqcap \exists \mathsf{site.FingerStructure} & \sqsubseteq & \mathsf{InjToFinger} \\
A_7: & \mathsf{FingerEntity} & \sqsubseteq & \mathsf{FingerStructure} \\
A_8: & \mathsf{FingerPart} & \sqsubseteq & \mathsf{FingerStructure} \\
A_9: & \mathsf{FingerPart} & \sqsubseteq & \exists \mathsf{part.FingerEntity} \\
A_{10}: & \mathsf{FingerStructure} & \sqsubseteq & \mathsf{HandPart} \\
A_{11}: & \mathsf{HandEntity} & \sqsubseteq & \mathsf{HandStructure} \\
A_{12}: & \mathsf{HandPart} & \sqsubseteq & \mathsf{HandStructure} \\
A_{13}: & \mathsf{HandPart} & \sqsubseteq & \exists \mathsf{part.HandEntity} \\
 & & & \\
B_1: & \mathsf{AmpOfHand} & \sqsubseteq & \mathsf{Amputation} \\
B_2: & \mathsf{AmpOfHand} & \sqsubseteq & \exists \mathsf{site.HandStructure} \\
B_3: & \mathsf{Amputation} \sqcap \exists \mathsf{site.HandStructure} & \sqsubseteq & \mathsf{AmpOfHand} \\
B_4: & \mathsf{InjToHand} & \sqsubseteq & \mathsf{Injury} \\
B_5: & \mathsf{InjToHand} & \sqsubseteq & \exists \mathsf{site.HandStructure} \\
B_6: & \mathsf{Injury} \sqcap \exists \mathsf{site.HandStructure} & \sqsubseteq & \mathsf{InjToHand}
\end{array}
$$

Figure 1.2: Example of an $\mathcal{EL}$ ontology.

Consider the extended ontology in Figure1.2, which is based on an example from [12]. We assume that axioms A_1 to A_{13} form our main ontology. It describes what an amputation of a finger (A_1 to A_3) and an injury to a finger (A_4 to A_6) is and models relations between finger and hand using concepts for structure, entity and part of a hand or a finger (A_7 to A_{13}). We now want to include descriptions for amputation and injury of a hand as well and therefore extend the ontology with axioms B_1 to B_6.

Note that we can derive the following two consequences from the extended ontology:

$$
\begin{array}{rcl}
\mathsf{InjToFinger} & \sqsubseteq & \mathsf{InjToHand} \\
\mathsf{AmpOfFinger} & \sqsubseteq & \mathsf{AmpOfHand}
\end{array}
$$

Of those two subsumptions the first one makes sense, since an injury to a finger is also an injury to the hand. The second subsumption, however, is undesirable, as an amputation of the finger is not also an amputation of the hand. We are therefore interested in finding a simple explanation for the unwanted consequence $\mathsf{AmpOfFinger} \sqsubseteq \mathsf{AmpOfHand}$ w.r.t. this extended ontology.

Note that the consequence that an amputation of a finger is an amputation of a hand follows neither from the main ontology ($A_1 - A_{13}$) nor from the extension ($B_1 - B_6$) alone, but it follows only from the combination of both ontologies ($A_1 - A_{13}$ together with $B_1 - B_6$).

Our goal now is to find an explanation for why the consequence holds in the extended ontology. Formally this means that we try to find a concept description C such that

- $\mathsf{AmpOfFinger} \sqsubseteq C$ holds in the extended ontology,
- $C \sqsubseteq \mathsf{AmpOfHand}$ holds in the extended ontology, and
- C only contains symbols which are common to the A-part and B-part of the ontology.

The common concepts (i.e. the concepts occurring in both parts of the ontology) are Amputation, Injury and HandStructure. We can show that the concept description

$$C \coloneqq \mathsf{Amputation} \sqcap \exists \mathsf{site}.\mathsf{HandStructure},$$

has this property, i.e. can be regarded as a high-level explanation for the subsumption $\mathsf{AmpOfFinger} \sqsubseteq \mathsf{AmpOfHand}$. This explanation can give us a hint on how to repair the ontology such that the unwanted subsumption is not a consequence of the extended ontology anymore. We can for example try to change one or more axioms related to either Amputation or $\exists \mathsf{site}.\mathsf{HandStructure}$. Here, replacing $\exists \mathsf{site}.\mathsf{HandStructure}$ by $\exists \mathsf{site}.\mathsf{HandEntity}$ in axioms B_2 and B_3 will fix the ontology.

In Section 5.3 we propose a method for finding explanations for subsumption relations in $\mathcal{EL}$ and $\mathcal{EL}^+$ ontologies and show how to apply it on this example in detail.

Reasoning About Geometric Graph Classes

Classes of geometric graphs occurring in algorithmic wireless network research can be described using axioms over suitable theories. Such axioms typically refer to points in $\mathbb{R}^2$ (and possibly also their coordinates) and often describe geometrical conditions for the existence or non-existence of edges. We assume an edge between two vertices u and v to be the line segment between u and v, i.e. it is always a straight line.

The *class of Gabriel graphs* G contains all undirected graphs without self loops satisfying the following conditions:

- An edge between two vertices u and v **exists** if there is no other vertex w which lies inside the smallest circle passing through u and v (see Figure 1.4).
- An edge between two vertices u and v **does not exist** if there is a vertex w which lies inside the smallest circle passing through u and v (see Figure 1.5).

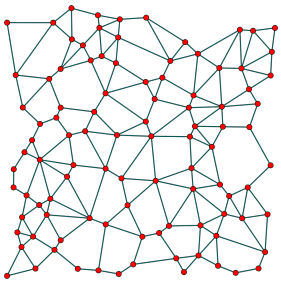

Figure 1.3: Example of a Gabriel graph[1].

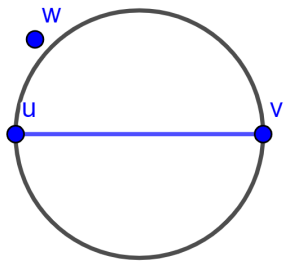

Figure 1.4: Existence of an edge in a Gabriel graph.

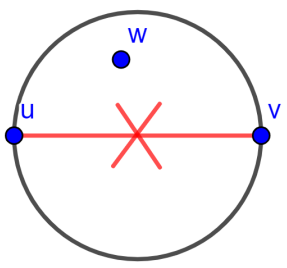

Figure 1.5: Non-existence of an edge in a Gabriel graph.

[1] https://en.wikipedia.org/wiki/Gabriel_graph#/media/File:Gabriel_graph.svg (24.04.23)

The *class of relative neighborhood graphs* R contains all undirected graphs without self loops satisfying the following conditions:

- An edge between two vertices u and v **exists** if there is no other vertex w which is closer to u than v and closer to v than u. Geometrically this means that no vertex w lies inside the intersecting area of the open disk with center u and radius $|uv|$ and the open disk with center v and radius $|uv|$ (see Figure 1.7).
- An edge between two vertices u and v **does not exist** if there is a vertex w which is closer to u than v or closer to v than u, i.e. if there is a vertex located inside the intersecting area of the open disk with center u and radius $|uv|$ and the open disk with center v and radius $|uv|$ (see Figure 1.8).

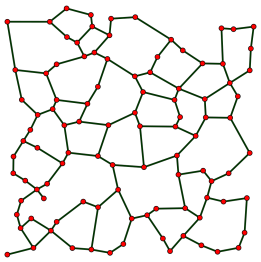

Figure 1.6: Example of a relative neighborhood graph[2].

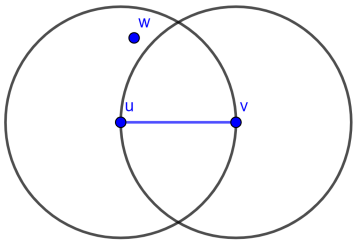

Figure 1.7: Existence of an edge in a relative neighborhood graph.

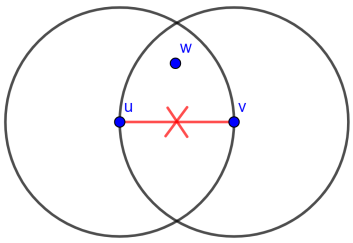

Figure 1.8: Non-existence of an edge in a relative neighborhood graph.

The *class of plane drawings* P contains all undirected geometric graphs without self loops which are drawn in the plane without any intersections between edges. For example, the Gabriel graph in Figure 1.3 and the relative neighborhood graph in Figure 1.6 are both plane drawings. In fact, every Gabriel graph and every relative neighborhood graph is a plane drawing.

We can prove that all Gabriel and relative neighborhood graphs are plane drawings by showing that the class inclusions $\mathsf{G} \subseteq \mathsf{P}$ and $\mathsf{R} \subseteq \mathsf{P}$ hold. To prove $\mathsf{G} \subseteq \mathsf{P}$, for instance, we have to show that $\mathsf{Ax_G} \wedge \neg \mathsf{Ax_P}$ is unsatisfiable, where $\mathsf{Ax_G}$ and $\mathsf{Ax_P}$ are axiomatizations of the classes G and P, respectively. The axioms contain information not only about the position of a vertex or the distance between vertices, but also about the existence of vertices and edges, so it is not possible to use methods for reasoning in arithmetic directly. The idea is therefore to first eliminate the symbols V and E, which correspond to the existence of a vertex or an edge, respectively. Thus, we can use a two-layered approach, in which we first use symbol elimination to derive geometric conditions under which class inclusion does not hold, and then use hierarchical reasoning to show that these geometric conditions cannot be satisfied. In case that the geometric conditions are satisfiable (i.e. class inclusion does not hold), we have methods to derive additional conditions under which class inclusion is guaranteed to hold.

In Chapter 6 we propose a method for checking class inclusions (of simple geometric graph classes like the ones described here, but also for graph classes obtained by transformations) and we present in detail the planarity proofs for the classes of Gabriel graphs and relative neighborhood graphs.

[2] `https://en.wikipedia.org/wiki/Relative_neighborhood_graph#/media/File:Relative_neighborhood_graph.svg` (24.04.23)

1.2 Related Work

In the following we will discuss the related work on general and property-directed symbol elimination as well as the related work on the application areas described in the thesis, i.e. verification of parametric systems, interpolation in the description logics $\mathcal{EL}$ and $\mathcal{EL}^+$ and reasoning about classes of geometric graphs. In this discussion we will state several research questions which will serve as a basis for the contributions stated in Section 1.3.

General Symbol Elimination

The most basic form of general symbol elimination is quantifier elimination. We say that a theory allows quantifier elimination, if for every formula of the theory there exists a quantifier-free formula that is equivalent to it with respect to the theory. Quantifier elimination is important for instance in verification, where it is used for computing reachable states, or in knowledge representation, where it is used for forgetting (restricting the signature to a subsignature without loss of consequences which can be described in the subsignature). The most prominent examples of theories which admit quantifier elimination are Presburger arithmetic and the theory of real closed fields (basically the real numbers).

If we only consider simple theories like linear arithmetic for integers and real numbers we know that quantifier elimination methods exist [22, 112]. But in some cases the investigated theories are more complex and do not allow quantifier elimination. Things get for example more complicated in the following two situations:

(1) We eliminate function or predicate symbols (second-order quantifier elimination).

(2) We consider combinations of theories.

General symbol elimination in complex theories can be achieved using refinements of superposition. In [14], Bachmair et al. mention the applicability of a form of hierarchical superposition to second-order quantifier elimination. This idea and possible links to interpolation are also mentioned by Ganzinger et al. in [49, 50]. In [73], Kovacs and Voronkov study inference systems and local derivations in the context of interpolant generation, and symbol elimination in proofs in such systems. The ideas are concretized using the superposition calculus and its extension LASCA (ground linear rational arithmetic and uninterpreted functions). An algorithm for second-order quantifier elimination, called SCAN, is presented by Gabbay and Ohlbach in [47]. The idea is, given a set of formulae, to use ordered resolution and factorization to compute sufficiently many consequences of the formulae, and to keep from the obtained set only the formulae which do not contain the predicate symbols to be eliminated.

General symbol elimination is closely related to uniform interpolation, which has been studied by Ghilardi et al. [27, 52, 28] (they often use the term "cover" instead of "uniform interpolant"). In [27] relations between uniform interpolation and model completeness are analyzed and a way of computing covers with a constrained version of the superposition calculus is proposed. In [52] the authors propose two algorithms for the computation of uniform interpolants in the theory of equality with uninterpreted function symbols (UIF). Results on computing combined covers, for instance in the combination of UIF and linear real arithmetic, are presented in [28].

Research question 1: Can we use established quantifier elimination methods also in combinations of theories?

Property-directed Symbol Elimination

The main difference to general symbol elimination is that in property-directed symbol elimination we have in addition to a formula F also a formula (or property) G given, which is meant to guide the process. In interpolation, for instance, we have two formulae A and B given such that $A \wedge B \models \bot$ holds and we are looking for a formula I with $A \models I$ such that the property $I \wedge B \models \bot$ holds. Methods for property-directed symbol elimination for complex theories (i.e. combinations or extensions of theories) have been proposed in many cases in relationship with interpolant computation. In [113] Yorsh et al. studied interpolation in combinations of theories. They propose a modular approach for computing interpolants in combined theories, which is based on the Nelson-Oppen framework and uses interpolation methods for the theories in a black box manner. They show that this approach can be used for a class of (convex) first-order theories which they call equality-interpolating. In [26], Brutomesso et al. extend these results by giving a weaker notion of equality interpolation, which in contrast to the work of Yorsh et al. also allows interpolation for combinations of non-convex theories. A modular approach for interpolation in data structures, which reduces the theories of data structures to the theories of equality and linear arithmetic, was analyzed by Kapur, Majumdar and Zarba in [68]. They prove that every recursively enumerable theory is interpolating (i.e. an interpolant always exists), and if in addition the theory allows quantifier elimination, it is quantifier-free interpolating (i.e. a quantifier-free interpolant always exists). They conclude, in particular, that the theories of integer, rational and real linear arithmetic are quantifier-free interpolating, and that the theory of arrays is interpolating, but not quantifier-free interpolating. Independently, Sofronie-Stokkermans in [91, 93] analyzed possibilities of computing interpolants hierarchically, and in [98, 100] proposed a method of hierarchical symbol elimination used for interpolant computation.

Property-directed symbol elimination is also used in the context of constraint generation, i.e. computing from given formulae F and G such that $F \wedge G$ is satisfiable a constraint Γ such that $\Gamma \wedge F \wedge G$ is unsatisfiable. In [98, 100] the proposed hierarchical symbol elimination method is not only used for interpolation, but also for inferring constraints on parameters. Already [97] mentions the possibility to infer constraints on parameters by hierarchical reasoning followed by quantifier elimination.

Research question 2: Under which conditions can we use hierarchical reasoning for property-directed symbol elimination in extensions and combinations of theories?

We discuss three areas of application relevant to this thesis with regard to property-directed symbol elimination: verification of parametric systems, knowledge representation (description logics) and geometric graph theory.

Automated Verification of Parametric Systems

In the verification of parametric systems it is important to show that a certain property holds for all states reachable from the initial state. One way to solve such problems is to identify an inductive invariant entailing the property to be proved. Finding suitable inductive invariants is non-trivial, the problem is undecidable in general. Solutions have been proposed for specific cases: In [67], Kapur proposes methods for invariant generation in theories such as Presburger arithmetic, real closed fields, and for polynomial equations and inequations with solutions in an algebraically closed field. The main idea is to use templates for the invariant (polynomials with undetermined coefficients), and solve con-

straints for all paths and initial values to determine the coefficients. A similar idea was used by Beyer et al. in [16] for constraints in linear real or rational arithmetic; it is shown that if an invariant is expressible with a given template, then it will be computed. Symbol elimination has been used for interpolation and invariant generation in many papers. The methods proposed in [67], where quantifier elimination or Gröbner bases computation are used for symbol elimination, are one class of examples. Quantifier elimination is also used by Dillig et al. in [38]. Applications of symbol elimination to invariant generation (briefly mentioned in [73]) are explored in detail in, among others, [72, 60] – there Vampire is used to generate a large set of invariants using symbol elimination; only invariants not implied by the theory axioms or by other invariants are kept (some of these tasks are undecidable). In [56], Gleiss et al. analyze functional and temporal properties of loops. For this, extended expressions (introduced in [72]) are used; symbol elimination as in [73] is used to synthesize invariants using quantification over iterations.

Various papers address the problem of strengthening a given formula to obtain an inductive invariant. In [23], Bradley proposes a goal-oriented invariant generation method for boolean/numeric transition systems, relying on finding counterexamples. Such methods were implemented in IC3 [21]. For programs using only integers and propositional variables, Dillig et al. in [38] use quantifier elimination to obtain increasingly more precise approximations of inductive invariants (termination is not guaranteed), expressed as arbitrary boolean combinations of linear integer constraints. In [44], Falke and Kapur analyze various ways of strengthening the formulae. Depending on how strengthening is attempted, their procedure may also determine whether the original formula is not an invariant. Situations in which termination is guaranteed are identified. This is the case for instance for logics in which only finitely many non-equivalent formulae over a fixed language can be built or for the quantifier-free logic of UTVPI-constraints. In [69], Karbyshev et al. propose a method to generate universal invariants in theories with the finite model property using diagram-based abstraction for invariant strengthening. Padon et al. in [80] identify sufficient conditions for the decidability of inferring inductive invariants in a given language and also present undecidability results; they prove decidability of inferring universal invariants for programs manipulating singly-linked-lists and show undecidability of inferring alternation-free invariants in the same setting, as well as undecidability of inferring universal invariants in general systems (beyond linked lists). They also propose methods for constructing in a systematic way new classes of systems with decidable invariant inference from already established classes. Invariant synthesis for array-based systems is studied by Ghilardi et al. in [53]; under local finiteness assumptions on the theory of elements and existence of well-quasi-orderings on configurations termination is guaranteed. In [2], Alberti et al. use lazy abstraction with interpolation-based refinement and discuss the applicability to invariant synthesis. A system for verifying safety properties that are "cubes" and invariant generation in array-based systems is described in [30]. In [57], Gurfinkel et al. propose an algorithm extending IC3 to support quantifiers for inferring universal invariants in theories of arrays, combining quantified generalizations (to construct invariants) with quantifier instantiation (to detect convergence). The safety problems regarded are assumed to be described by quantifier-free formulae and the quantification in the generated universal invariants is limited to integer variables.

All the approaches addressing the problem of invariant strengthening have limitations in some way. Some methods restrict the shape of the invariants, for example by using templates, some can only be used for numeric domains or require theories which have the finite model property, and some do not come with correctness or termination guarantees.

Research question 3: Can property-directed symbol elimination in combination with hierarchical reasoning be used for inductive invariant synthesis? Under which conditions are such approaches correct/terminating?

Finding Explanations in $\mathcal{EL}^+$

Description logics are logics for knowledge representation used in ontologies. They provide a logical basis for modeling and reasoning about objects, classes of objects (concepts), and relationships between them (roles). Based on the operations that are allowed we obtain different descriptions logics. $\mathcal{ALC}$ for example allows intersection, union and negation of concepts, as well as universal and existential role restriction. If we only allow intersection and existential restriction, we obtain the description logic $\mathcal{EL}$, a logic used for reasoning in medical ontologies [103, 102]. $\mathcal{EL}^+$ is an extension of $\mathcal{EL}$ which also allows role inclusions.

One of the problems arising when creating description logic ontologies is ensuring that they do not contain mistakes that could allow to prove subsumptions between concepts that are not supposed to hold. One situation in which this can happen is when already existing ontologies which can be considered trustworthy are extended, or when two ontologies are put together. Even if the new ontology is still consistent, one needs to make sure that no concept inclusions which are not supposed to be true can be derived. It is therefore important to provide simple explanations for concept subsumptions in such combined ontologies (containing, for instance, only symbols that occur both in the original ontology and in the extension).

One method for finding justifications in description logic TBoxes that has been addressed in other work is the so-called axiom pinpointing. The idea is to find a minimal axiom set which already has the consequence in question. Algorithms for computing minimal axiom sets for $\mathcal{ALC}$-terminologies were e.g. given by Baader and Hollunder [7], and by Schlobach and Cornet [88]. They are extensions of the tableau-like satisfiability algorithm for $\mathcal{ALC}$ and the tableau-like consistency algorithm for $\mathcal{ALC}$, respectively, in which they make use of labels to keep track of the axioms that were used during the execution of the algorithms. In contrast to the algorithm in [88], the one in [7] does not compute minimal axiom sets directly, but Boolean formulae from which they can be derived. Possibilities of explaining $\mathcal{ALC}$-subsumption (again based on tableau implementations) are described in [19]. In [10] and [11] Baader et al. give a similar algorithm for axiom pinpointing in the description logics $\mathcal{EL}$ and $\mathcal{EL}^+$, respectively, in which they modify the subsumption algorithm for $\mathcal{EL}$ and $\mathcal{EL}^+$, respectively. Here again labels are used to keep track of the axioms needed and the output is a Boolean formula, from which the axioms can be derived. They show that computing all possible minimal axiom sets may need exponential time, whereas computing one such set can be done in polynomial time. In [11] they consider extensions of TBoxes, i.e. unions of a static TBox (with irrefutable axioms) and a refutable TBox. Possibilities of finding small proofs for description logics have been investigated in [3], which can be considered as a further step in an incremental way of generating explanations.

In [93, 99] Sofronie-Stokkermans investigated methods for interpolation in the description logics $\mathcal{EL}$ and $\mathcal{EL}^+$. There it is shown that concept subsumption tests in $\mathcal{EL}$ and $\mathcal{EL}^+$ can be reduced to checking validity of implications w.r.t. the theory of semilattices with monotone operators and that in this theory the proof tasks can be hierarchically reduced to checking satisfiability in the theory of semilattices. In [46], Fortin et al. analyze interpolation properties for $\mathcal{EL}$ and extensions thereof. They prove that in contrast to $\mathcal{EL}$, most of its extensions, in particular extensions of $\mathcal{EL}$ with role inclusions (i.e. $\mathcal{EL}^+$), do

not enjoy the $\sqsubseteq$-interpolation property, i.e. the property that for two concept descriptions A and B with $A \sqsubseteq B$ there exists a concept description I such that $A \sqsubseteq I$ and $I \sqsubseteq B$ (they refer to this property as Craig interpolation). However, it is shown that if all role symbols occuring in role inclusions are guaranteed to be part of the shared signature, then an interpolant exists.

Research question 4: Can we use property-directed symbol elimination and hierarchical reasoning to find easy explanations for inconsistencies in combined $\mathcal{EL}^+$ ontologies?

Checking Containedness Between Geometric Graph Classes

Checking containedness between geometric graph classes is a topic of interest, as it provides a general tool to check graph properties resulting from distributed graph algorithms. Testing containedness of graph classes has been studied by Böltz and Frey in recent work. In [17] they derive by hand formulae representing necessary and sufficient conditions for containedness between certain types of graph classes. In this thesis we propose automated methods relying on symbol elimination and hierarchical reasoning for checking inclusion without the manual component in [17]. We are not aware of other similar approaches to the area of computational geometric graph theory, combining reasoning about graphs with reasoning about geometric properties of graphs. In his numerous publications on the topic, e.g. [31, 32, 33, 34, 35], Courcelle uses a logical representation of graphs based on monadic second-order logic; many of the results rely on graphs with bounded tree-width or bounded clique-width. In other existing approaches, higher-order theorem provers are used, for instance Isabelle/HOL is used for the verification of graph algorithms in [1], and Coq is used for checking graph theoretical properties in [39]. In [58] an overview on the analysis of graph transformation systems is given.

Research question 5: Can we use general and property-directed symbol elimination in combination with hierarchical reasoning to prove inclusion of geometric graph classes?

1.3 Contributions of the Thesis

For many problems in automated reasoning we need efficient methods for symbol elimination. The challenge is that the tasks become much more difficult for complex theories, for instance for combinations or extensions of theories. In the thesis we look at both situations, but our focus is on extensions of theories. A theory extension consists of a base theory which is extended with additional function symbols whose properties are described by a set of axioms. A general approach we are using to reason in such theory extensions is hierarchical reasoning. This means that we try to reduce the given problem in a hierarchical manner to a problem in the base theory. Provided that the extensions have the right properties and we have the means to reason effectively in the base theory, we can then solve the problems. We therefore have to analyze under which conditions hierarchical reasoning in combination with symbol elimination is possible. An especially nice property of theory extensions is the so-called locality [90]. In the thesis we propose a method for symbol elimination in (local) theory extensions that we used in [81, 82], which is a refined version of the algorithm from [98, 100]. Given a problem which is structured as a theory extension, the idea is to reduce it to the base theory using purification and instantiation, to apply quantifier elimination in the base theory, and to translate the result back to the original language.

This general approach can be used in many different application domains. We will show its applicability by proposing specialized methods for several interesting problems in different areas of automated reasoning. In addition to that we will demonstrate our methods on a wide range of interesting examples from verification and description logics.

The application areas we consider in this thesis are the following:

(1) **Verification of parametric systems:** We use property-directed symbol elimination for the goal-oriented synthesis of inductive invariants of parametric systems.

(2) **Description logics:** We use methods for symbol elimination in propositional logic based on resolution for finding explanations for subsumptions in $\mathcal{EL}^+$ ontologies.

(3) **Geometric graph theory:** We use second-order quantifier elimination combined with property-directed symbol elimination to check inclusion of geometric graph classes and to detect additional conditions under which inclusion holds.

Verification of Parametric Systems

In the thesis we will build upon existing work on automated verification and synthesis in parametric systems [63, 96, 97] by investigating possibilities for automated invariant checking and goal-oriented generation of inductive invariants. The results were published in [81] and [82]. The goal is to verify safety of a system (or program) by showing that a safety property is an inductive invariant of the system or, if this is not the case, to strengthen the property such that the strengthened property can be proved to be an inductive invariant of the system.

Given as input is the description of a system or a program and the property Ψ, expressed by a universally quantified formula, which needs to be checked for invariance. We present a method which starts with the universally quantified formula Ψ and successively strengthens it, using a certain form of abductive reasoning based on symbol elimination (by abductive reasoning we mean inferring additional conditions on parameters). In case of termination it can be proved that we obtain a universal inductive invariant that entails Ψ, or the answer "no such invariant exists". We identify situations in which the method terminates and present some refinements, which we also illustrate on examples.

As described already in Chapter 1.2, there is a lot of existing work in the area of parametric verification, but all the existing approaches have limitations in one way or another. Common limitations are:

(1) Templates are used, which are difficult to use in connection with additional function symbols (for instance linear arithmetic with an additional function symbol a to describe an array). This is the case for instance in the methods proposed in [67, 38].

(2) Only simple theories are covered. The methods of [23, 38, 44] for example can only handle numeric domains.

(3) The finite model property is required, which is the case for instance in the work described in [69, 80].

(4) Correctness and termination are not guaranteed, as e.g. in [57].

(5) The methods are non-deterministic, as is the case for example in [53].

While we rely on methods similar to the ones used in [23, 53, 38, 44, 80, 69, 57], there are several differences between our work and previous work. Our method is not based on templates, it instead allows us to choose the language for the candidate invariants (we can search for invariants not containing certain constants or function symbols). Furthermore, our method is specialized for use with theory extensions and can therefore handle more complex theories than many of the existing work. The finite model property is only required if we want to guarantee termination. In addition, our algorithm is deterministic. We can guarantee correctness under locality assumptions (which, for instance, do not need to be checked separately if updates and properties are in the array property fragment) and our termination results are established for classes of formulae for which only finitely many atomic formulae formed with a fixed number of variables can be generated using quantifier elimination.

Description logics

An application in description logics that we will address in the thesis is the problem of finding simple explanations for unwanted subsumptions in combinations or extensions of ontologies. Previous work on this topic was published in [83]. We investigate this problem for the case in which the ontologies consist of TBoxes or CBoxes. We restrict to the description logics $\mathcal{EL}$ and $\mathcal{EL}^+$. We use the encoding of TBox subsumption for $\mathcal{EL}$ as a uniform word problem in classes of semilattices with monotone operators and the $\leq$-interpolation property in these classes of algebras (it essentially says that if $A \leq B$ can be derived, then there exists an intermediate term t containing only symbols shared by A and B such that $A \leq t \wedge t \leq B$), as well as extensions to these results in the presence of role inclusions. A subset of the axioms needed for deriving a concept inclusion can be determined using an unsatisfiable core computation or pinpointing [10, 11]. For computing the $\leq$-interpolating terms we use a translation to propositional logic and methods for computing Craig interpolants in propositional logic. The result is translated back to the description logic language. We regard the translated $\leq$-interpolating terms as high-level explanations for the subsumption. Such easy explanations are important for debugging ontologies, since full proofs may be too complicated in practice. Our approach to $\sqsubseteq$-interpolation is different from the one proposed in [46], in which a proof technique based on simulations is used. The method we propose in this thesis is based on hierarchical reasoning in theory extensions and a formalization of the $\sqsubseteq$-interpolation problem for $\mathcal{EL}^+$ as a $\leq$-interpolation problem for the theory of semilattices with monotone operators.

Geometric graph theory

In this thesis we devise methods for checking containedness between geometric graph classes. We focus on graph classes that can be described with universally quantified axioms. When defining such graph classes we use two kinds of predicates. On the one hand we have edge predicates, describing that an edge between two vertices in the graph exists, on the other hand we have inclusion, exclusion and transfer predicates, describing geometric conditions for the existence of an edge, usually based on a distance or cost function.

Transformations can be applied to such graphs in order to make the graphs symmetric, for example by removing all directed edges whose reverse edge is missing or by adding all the missing reverse edges. This way we can define further graph classes. When checking inclusion between graph classes described using transformations – since the transformations update the sets of edges – we need to check entailment of second-order formulae. In

addition, many such graph class descriptions are parametric in nature, so the goal is, in fact, to obtain (weakest) conditions on the parameters used in such descriptions that guarantee that graph classes are non-empty or that inclusions hold. This can be achieved by eliminating "non-parametric" constants or function symbols used in the description of such classes.

Our goal is to prove that one graph class is contained in another graph class, where both are possibly defined using transformations. This can be done in two steps. In a first step, we use general symbol elimination for eliminating existentially quantified edge predicates. In a second step we check unsatisfiability of the obtained formula to prove containedness. If containedness cannot be proved, the methods we use allow us to generate counterexamples or conditions on parameters such that the inclusion holds. For the latter we use property-directed symbol elimination.

The main application area we consider in this thesis is the analysis of inclusions between graph classes arising in wireless network research. The approach we propose (and the tools we use for this) can be a good instrument in theoretical graph and network theory, which would allow the user to test whether certain generalizations of concept are problematic and to locate possible problems with the general formalizations. Our approach is orthogonal to other approaches in this area such as [33], [1] or [39]. It relies on methods for second-order quantifier elimination which allow a reduction of many problems to satisfiability modulo a suitable theory for which state of the art SMT solvers can be used. The procedure is sound and our approach allows us to identify situations in which completeness can be guaranteed. Many of the results in [31, 32, 33, 34, 35] rely on graphs with bounded tree-width or bounded clique-width. We do not impose such restrictions on the graphs we analyze. In contrast to the approaches proposed in [1] or [39], in this thesis we rely on methods which avoid the use of higher-order logic systems such as Isabelle/HOL or Coq.

Overview

In the following the contributions of this thesis are stated, each of them corresponding to one of the research questions from Section 1.2.

Contribution 1: We establish results on quantifier elimination in combinations of theories.

Contribution 2: We propose a refinement to the method for symbol elimination in local theory extensions.

Contribution 3: We use methods for symbol elimination to obtain a method for inductive invariant synthesis in parametric systems based on hierarchical reasoning in extensions of theories.

Contribution 4: We obtain methods based on hierarchical reasoning and symbol elimination for finding high-level explanations for subsumption in combinations of $\mathcal{EL}^+$ ontologies.

Contribution 5: We use general symbol elimination and property-directed symbol elimination in combination with hierarchical reasoning to prove containedness of geometric graph classes.

1.4 Publications

The contributions stated in this thesis are based on former publications of papers in conferences or workshops. In the following we list these papers in chronological order, from oldest to newest:

- Dennis Peuter and Viorica Sofronie-Stokkermans: On Inductive Verification and Synthesis. In Selected Student Contributions and Workshop Papers of LuxLogAI 2018, volume 10 of Kalpa Publications in Computing, pages 1-8. EasyChair, 2018.

- Dennis Peuter and Viorica Sofronie-Stokkermans: On Invariant Synthesis for Parametric Systems. In Proceedings of the 27th International Conference on Automated Deduction (CADE-27), volume 11716 of Lecture Notes in Computer Science, pages 385-405. Springer, 2019.

- Dennis Peuter and Viorica Sofronie-Stokkermans: Finding High-Level Explanations for Subsumption w.r.t. Combinations of CBoxes in $\mathcal{EL}$ and $\mathcal{EL}^+$. In Proceedings of the 33rd International Workshop on Description Logic (DL 2020) co-located with the 17th International Conference on Principles of Knowledge Representation and Reasoning (KR 2020), volume 2663 of CEUR Workshop Proceedings. CEUR-WS.org, 2020.

- Lucas Böltz, Hannes Frey, Dennis Peuter, and Viorica Sofronie-Stokkermans: On Testing Containedness Between Geometric Graph Classes using Second-order Quantifier Elimination and Hierarchical Reasoning (Short Paper). In Proceedings of the Second Workshop on Second-Order Quantifier Elimination and Related Topics (SOQE 2021) associated with the 18th International Conference on Principles of Knowledge Representation and Reasoning (KR 2021), volume 3009 of CEUR Workshop Proceedings, pages 37-45. CEUR-WS.org, 2021.

- Dennis Peuter and Viorica Sofronie-Stokkermans: Symbol Elimination and Applications to Parametric Entailment Problems (Abstract). In Proceedings of the Second Workshop on Second-Order Quantifier Elimination and Related Topics (SOQE 2021) associated with the 18th International Conference on Principles of Knowledge Representation and Reasoning (KR 2021), volume 3009 of CEUR Workshop Proceedings, pages 83-91. CEUR-WS.org, 2021.

- Dennis Peuter and Viorica Sofronie-Stokkermans: Symbol Elimination and Applications to Parametric Entailment Problems. In Proceedings of the 13th International Symposium on Frontiers of Combining Systems (FroCoS 2021), volume 12941 of Lecture Notes in Computer Science, pages 43-62. Springer, 2021.

- Dennis Peuter, Viorica Sofronie-Stokkermans, and Sebastian Thunert: On P-Interpolation in Local Theory Extensions and Applications to the Study of Interpolation in the Description Logics $\mathcal{EL}$, $\mathcal{EL}^+$. In Proceedings of the 29th International Conference on Automated Deduction (CADE-29), volume 14132 of Lecture Notes in Computer Science, pages 419-437. Springer, 2023.

Table 1.9 also lists these papers in chronological order and denotes which chapter of the thesis the contents of the papers is related to.

	Title	Authors	Event	Related to	Ref.
1	On Inductive Verification and Synthesis	Dennis Peuter, Viorica Sofronie-Stokkermans	LuxLogAI 2018	Chapter 4	[81]
2	On Invariant Synthesis for Parametric Systems	Dennis Peuter, Viorica Sofronie-Stokkermans	CADE 2019	Chapters 3 and 4	[82]
3	Finding High-Level Explanations for Subsumption w.r.t. Combinations of CBoxes in $\mathcal{EL}$ and $\mathcal{EL}^+$	Dennis Peuter, Viorica Sofronie-Stokkermans	DL 2020	Chapter 5	[83]
4	On Testing Containedness Between Geometric Graph Classes using Second-order Quantifier Elimination and Hierarchical Reasoning (Short Paper)	Lucas Böltz, Hannes Frey, Dennis Peuter, Viorica Sofronie-Stokkermans	SOQE 2021	Chapter 6	[18]
5	Symbol Elimination and Applications to Parametric Entailment Problems (Abstract)	Dennis Peuter, Viorica Sofronie-Stokkermans	SOQE 2021	Chapter 6	[85]
6	Symbol Elimination and Applications to Parametric Entailment Problems	Dennis Peuter, Viorica Sofronie-Stokkermans	FroCoS 2021	Chapter 6	[84]
7	On P-Interpolation in Local Theory Extensions and Applications to the Study of Interpolation in the Description Logics $\mathcal{EL}$, $\mathcal{EL}^+$	Dennis Peuter, Viorica Sofronie-Stokkermans, Sebastian Thunert	CADE 2023	Chapter 5	[86]

Figure 1.9: List of publications.

1.5 Structure of the Thesis

In **Chapter 2** we present the preliminaries that are needed for the understanding of this thesis. We first describe the syntax and semantics of first-order logic and some related notions such as unsatisfiable cores, ordered resolution and Craig interpolation. Afterwards we define theories and theory extensions and introduce the notion of locality. For local theory extensions we show how to recognize them and present a method for hierarchical reasoning. We then define the notion of quantifier elimination and present methods for eliminating quantifiers in different theories: first, in the theory of real closed fields, and second, in the theory of an infinite set.

In **Chapter 3** we present our results on symbol elimination. We first analyze possibilities for quantifier elimination in combinations of theories, in particular in the combination of the theory of real closed fields and the theory of an infinite set. Afterwards we explain an algorithm for symbol elimination in (local) theory extensions proposed in [98]. We then propose a refinement of the algorithm and illustrate both the algorithm and its refinement on an example.

In **Chapter 4** we show applications of hierarchical reasoning and symbol elimination to the verification of parametric systems. We first describe the problems of invariant checking, constraint synthesis and invariant generation for transition constraint systems. We then present a method for invariant generation that is based on hierarchical reasoning and symbol elimination. We analyze the correctness of the method, identify situations in which termination is guaranteed and show some refinements that decrease complexity. We illustrate the algorithm and its refinements on examples.

In **Chapter 5** we present a method for generating high-level explanations for subsumptions in the description logics $\mathcal{EL}$ and $\mathcal{EL}^+$. We first introduce the description logics $\mathcal{EL}$ and $\mathcal{EL}^+$, explain their algebraic semantics and prove a locality property related to $\mathcal{EL}^+$. Then we define the P-interpolation property w.r.t. a set P of binary predicate symbols and show that the theory of semilattices with monotone operators satisfies this property for $P \in \{\leq, \approx\}$. Based on this property for semilattices we then propose an algorithm for applying a form of interpolation in $\mathcal{EL}$ and $\mathcal{EL}^+$, i.e. for computing intermediate terms for concept subsumptions, which is used for generating the explanations. We illustrate the method on several examples.

In **Chapter 6** we illustrate how to use symbol elimination in local theory extensions for checking containedness of geometric graph classes whose descriptions usually refer to distances between vertices. For this we first show that the extension of the theory of equality with certain distance functions is local. We then show on several examples how containedness of one graph class in another class can be checked, or in case of parametrically described classes, how to compute weakest properties on the parameters such that the class inclusion holds.

In **Chapter 7** we conclude by summarizing and analyzing the results that were achieved in the thesis.

2 Preliminaries

In this chapter we describe the notions and methods that are important for this thesis. We start in Section 2.1 by introducing the syntax and semantics for first-order logic and related notions. In Section 2.2 we explain what theories and theory extensions are, and describe the locality property for theory extensions. In Section 2.3 quantifier elimination is presented, in particular the virtual substitution method for quantifier elimination in the theory of real numbers (i.e. the theory of real closed fields).

2.1 First-order Logic

We assume known standard definitions from first-order logic and refer the reader to [45] for more information. We briefly present the syntax and semantics of first-order logic, then continue by giving a description of ordered resolution and interpolation. Since we only need ordered resolution for propositional logic in this thesis, we will only describe the method for propositional logic and not for first-order logic.

Notation: Throughout this thesis, we usually denote variables with x, y, z, constants with a, b, c and terms with t, s. Propositional variables are denoted with P, Q, R, S, formulae with F, G, H, I and clauses with C, D. We usually denote sets of formulae or sets of clauses with N. We may use all the symbols mentioned above also in combination with indices, for example $x_1, ..., x_n$ for variables or F_1, F_2 for formulae. For a tuple of variables $x_1, ..., x_n$ we sometimes use the notation $\overline{x}$, and analogously for constants and terms. We often refer to finite conjunctions of formulae as "sets of formulae". For instance, if N_1 and N_2 are finite sets of formulae, then $N_1 \cup N_2$ stands for the conjunction of all formulae in the set $N_1 \cup N_2$. We use $\approx$ as a notation for the equality predicate to distinguish it from the usual equality sign (denoted by $=$). To save space, we sometimes write $a < b < c$ for a conjunction $a < b \wedge b < c$ (and similarly for other relation symbols or larger conjunctions).

2.1.1 Syntax and Semantics

We start by defining the syntax and semantics of first-order logic in the case of one-sorted signatures; the extension to many-sorted signatures is shown at the end of the section.

Syntax

We consider *signatures* $\Pi = (\Sigma, \mathsf{Pred})$, where Σ is a set of *function symbols* and Pred a set of *predicate symbols* such that for each function symbol and predicate symbol the arity is specified. A function symbol with arity zero is called a *constant* and a predicate symbol with arity zero is called a *propositional variable*. We consider a fixed countably infinite set X of *variables*.

Remark: If the set Σ of function symbols is empty and all predicates in Pred have arity zero (i.e. Π only contains propositional variables), then we are in propositional logic.

The set of Π-*terms* $T_\Pi(X)$ over X is defined as the smallest set containing

- all variables in X;
- all constants in Π;
- $f(t_1, ..., t_n)$ if f is a function in Π with arity n and every t_i is a Π-term for $i \in \{1, ..., n\}$.

In a term $t = f(t_1, ..., t_n)$ we call f the *root* and t_i the *subterms* of t. A term is called a *ground term* if it does not contain variables. We denote the set of ground terms by T_Π.

A Π-*atom* is an equality $t_1 \approx t_2$ between terms or an expression $P(t_1, ..., t_n)$, where P is a predicate symbol of arity n and t_i is a Π-term for every $i \in \{1, ..., n\}$. A Π-*literal* is a Π-atom A (positive literal) or a negated Π-atom $\neg A$ (negative literal).

The set of Π-*formulae* $F_\Pi(X)$ with variables in X is defined as the smallest set containing

- $\top$, $\bot$, and all Π-atoms;
- $\neg F$ if F is a Π-formula;
- $F_1 \circ F_2$ if F_1 and F_2 are Π-formulae and $\circ \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$;
- $\forall x F$ and $\exists x F$ if F is a Π-formula and $x \in X$.

A formula not containing any quantifiers is called *quantifier-free*. A formula not containing any variables is called a *ground formula*. We denote the set of ground formulae with F_Π.

A Π-*clause* is a disjunction of Π-literals. A clause is called a *ground clause* if it does not contain variables. A *Horn-clause* is a clause which contains at most one positive literal.

Remark: If we only have equality as a predicate, then for terms or clauses containing only function symbols in a set Σ we sometimes refer to as Σ-*terms* or Σ-*clauses*. We denote the set of all Σ-terms by T_Σ.

A formula is in *conjunctive normal form* (CNF) if it is a conjunction of clauses (i.e. if it is a conjunction of disjunctions of literals); it is in *disjunctive normal form* (DNF) if it is a disjunction of conjunctions of literals.

In QxF with $Q \in \{\exists, \forall\}$ we call F the *scope* of the quantifier Qx. A variable x is called a *bound variable* of a formula F if it appears in the scope of a quantifier Qx in F. A variable that is not bound by a quantifier is called a *free variable* of F. A Π-formula is called a Π-*sentence* if every variable is bound by a quantifier. It is in *prenex normal form* if it consists of a string of quantifiers and bound variables followed by a quantifier-free formula. It is called a *universal formula* if it is in prenex normal form and contains only universal quantifiers.

Semantics

A Π-*structure* for a signature $\Pi = (\Sigma, \mathsf{Pred})$ is a tuple $\mathcal{A} = (|\mathcal{A}|, \Sigma^{\mathcal{A}}, \mathsf{Pred}^{\mathcal{A}})$, where

- $|\mathcal{A}|$ is a non-empty set called the *universe* of $\mathcal{A}$,
- $\Sigma^{\mathcal{A}}$ is a set containing a function $f_{\mathcal{A}} : |\mathcal{A}|^n \rightarrow |\mathcal{A}|$ for every $f \in \Sigma$ with arity n, and
- $\mathsf{Pred}^{\mathcal{A}}$ is a set containing a predicate $P_{\mathcal{A}} \subseteq |\mathcal{A}|^m$ for every $P \in \mathsf{Pred}$ with arity m.

A *valuation* over a Π-structure $\mathcal{A}$ is a total map $\beta : X \to |\mathcal{A}|$, assigning to each variable an element of the universe. By $\beta[x \to a]$ we denote the valuation that maps x to a and every other variable $y \in X$ to $\beta(y)$. The value $\mathcal{A}(\beta)(t)$ of a term t in $\mathcal{A}$ w.r.t. β is defined inductively by:

- $\mathcal{A}(\beta)(t) = \beta(t)$ if $t \in X$,
- $\mathcal{A}(\beta)(t) = c_{\mathcal{A}}$ if t is a constant c,
- $\mathcal{A}(\beta)(t) = f_{\mathcal{A}}(\mathcal{A}(\beta)(t_1), ..., \mathcal{A}(\beta)(t_n))$ if $t = f(t_1, ..., t_n)$.

The set of truth values is $\{0, 1\}$, where 0 stands for *false* and 1 for *true*. If $\mathcal{A}$ is a Π-structure and β a valuation, then $\mathcal{A}(\beta)(F)$, the truth value of a formula F in $\mathcal{A}$ w.r.t. β, is defined inductively over the structure of F by

- $\mathcal{A}(\beta)(\top) = 1$,
- $\mathcal{A}(\beta)(\bot) = 0$,
- $\mathcal{A}(\beta)(P(t_1, ..., t_n)) = 1$ if and only if $(\mathcal{A}(\beta)(t_1), ..., \mathcal{A}(\beta)(t_n)) \in P_{\mathcal{A}}$,
- $\mathcal{A}(\beta)(\neg F) = 1 - \mathcal{A}(\beta)(F)$,
- $\mathcal{A}(\beta)(F \wedge G) = \min(\mathcal{A}(\beta)(F), \mathcal{A}(\beta)(G))$,
- $\mathcal{A}(\beta)(F \vee G) = \max(\mathcal{A}(\beta)(F), \mathcal{A}(\beta)(G))$,
- $\mathcal{A}(\beta)(F \to G) = 0$ if and only if $\mathcal{A}(\beta)(F) = 1$ and $\mathcal{A}(\beta)(G) = 0$,
- $\mathcal{A}(\beta)(\exists x F) = \max\limits_{a \in |\mathcal{A}|}(\mathcal{A}(\beta[x \to a])(F))$,
- $\mathcal{A}(\beta)(\forall x F) = \min\limits_{a \in |\mathcal{A}|}(\mathcal{A}(\beta[x \to a])(F))$.

A structure $\mathcal{A}$ is called a *model* of a formula F, denoted $\mathcal{A} \models F$, if and only if $\mathcal{A}(\beta)(F) = 1$ for every valuation $\beta : X \to |\mathcal{A}|$. F is called *valid*, denoted $\models F$, if and only if $\mathcal{A} \models F$ for every structure $\mathcal{A}$. F is called *satisfiable* if and only if there exist $\mathcal{A}$ and β such that $\mathcal{A}(\beta)(F) = 1$, otherwise it is called *unsatisfiable*. If F and G are formulae, we say that F entails G, denoted $F \models G$, if and only if for all structures $\mathcal{A}$ and all valuations β it holds that $\mathcal{A}(\beta)(F) = 1$ implies $\mathcal{A}(\beta)(G) = 1$. $F \models \bot$ means that F is unsatisfiable. A set of formulae N is satisfiable if and only if there exist $\mathcal{A}$ and β such that $\mathcal{A}(\beta)(F) = 1$ for all $F \in N$, otherwise it is called unsatisfiable. We call two formulae F and G *equivalent*, denoted $F \equiv G$, if $F \models G$ and $G \models F$.

Unsatisfiable Cores

If we have an unsatisfiable set of formulae N, often not every formula in N is responsible for the unsatisfiability of N. To make reasoning more efficient it can therefore be of advantage to compute a subset of N that is still unsatisfiable. Such an unsatisfiable subset of N is called an unsatisfiable core of N.

Definition 2.1 ((Minimal) Unsatisfiable Core). *An* unsatisfiable core *of an unsatisfiable set of formulae N is a subset $U \subseteq N$ such that U is also unsatisfiable. An unsatisfiable core U is called* minimal *if every proper subset $V \subset U$ is satisfiable.*

Example 2.2. We consider the set $N = \{F_1, F_2, F_3, F_4, F_5\}$ of formulae, where:

$$F_1 = P$$
$$F_2 = P \to Q$$
$$F_3 = \neg Q \wedge R$$
$$F_4 = R \to S$$
$$F_5 = \neg R$$

Then $\{F_1, F_2, F_3\}$ and $\{F_3, F_5\}$ are minimal unsatisfiable cores of N. ■

The Many-sorted Case

Sometimes we need signatures which have more than one sort with functions whose domains range over different sorts. We then consider a *many-sorted signature* $\Pi = (S, \Sigma, \mathsf{Pred})$, where

- S is a set of sorts,
- Σ is a set of function symbols such that for every $f \in \Sigma$ its arity $a(f) = s_1...s_n \to s$ is specified, and
- Pred is a set of predicate symbols such that for every $p \in \mathsf{Pred}$ its arity $a(p) = s_1...s_m$ is specified,

where $s_1, ..., s_n, s_m, s$ are sorts in S.

Let $X = \{X_s | s \in S\}$, where every X_s is a countably infinite set of variables of sort s.

Let the set $T^s_\Pi(X)$ of terms of sort s be recursively defined as follows:

- Every variable $x \in X_s$ is a term of sort s.
- Every constant c with $a(c) =\to s$ is a term of sort s.
- If $a(f) = s_1...s_n \to s$ and t_i are terms of sort s_i for all $i \in \{1, ..., n\}$, then $f(t_1, ..., t_n)$ is a term of sort s.

We denote by $\{T^s_\Pi(X)\}_{s \in S}$ the set of *many-sorted* Π*-terms* over X with sorts in S.

For *many-sorted* Π*-atoms* $p(t_1, ..., t_m)$ with arity $a(p) = s_1...s_m$ we require that t_i is a term of sort s_i for all $i \in \{1, ..., m\}$.

A *many-sorted* Π*-structure* is a tuple $\mathcal{A} = (\{|\mathcal{A}|_s\}_{s \in S}, \Sigma^{\mathcal{A}}, \mathsf{Pred}^{\mathcal{A}})$, where

- every $|\mathcal{A}|_s$ is a non-empty set called the *universe* of $\mathcal{A}$ of sort s,
- $\Sigma^{\mathcal{A}}$ is a set containing a function $f_{\mathcal{A}} : |\mathcal{A}|_{s_1} \times ... \times |\mathcal{A}|_{s_n} \to |\mathcal{A}|_s$ for every $f \in \Sigma$ with arity $s_1...s_n \to s$, and
- $\mathsf{Pred}^{\mathcal{A}}$ is a set containing a predicate $p_{\mathcal{A}} \subseteq |\mathcal{A}|_{s_1} \times ... \times |\mathcal{A}|_{s_m}$ for every $p \in \mathsf{Pred}$ with arity $s_1...s_m$.

A *valuation over a many-sorted* Π*-structure* is a set $\beta = \{\beta_s\}_{s \in S}$, where every β_s is a map $\beta_s : X_s \to |\mathcal{A}|_s$.

2.1.2 Proof Calculi and Interpolation

In this section we describe important notions related to first-order logic and propositional logic, which are needed later in the thesis. For reasoning in propositional or first-order logic it is often necessary to show satisfiability or unsatisfiability of logical formulae. There exist several proof calculi for deciding satisfiability, for example semantic tableau calculi and the resolution calculus, which both have variants for propositional logic as well as first-order logic. In the following we decribe the ordered resolution calculus.

Ordered Resolution

Ordered resolution is a method for checking satisfiability of formulae in conjunctive normal form. We here present the ordered resolution calculus only for propositional logic, because we will only need it for propositional logic and not for first-order logic in the thesis. We denote the ordered resolution calculus by $Res^{\prec}$.

Definition 2.3 (Literal Ordering). *Let $\prec$ be a total and well-founded ordering on propositional variables. We can extend $\prec$ to an ordering on literals:*

- *If $P \prec Q$, then also $P \prec \neg Q$, $\neg P \prec Q$ and $\neg P \prec \neg Q$.*
- *$P \prec \neg P$ for every propositional variable P.*

A literal L is called *maximal w.r.t.* $\prec$ in a clause $L_1 \vee ... \vee L_n \vee L$ if there is no $L' \in \{L_1, ..., L_n\}$ such that $L \prec L'$, i.e. if for all $L' \in \{L_1, ..., L_n\}$ it holds that either $L' \prec L$ or $L' = L$. A literal L is called *strictly maximal w.r.t.* $\prec$ in $L_1 \vee ... \vee L_n \vee L$ if for all $L' \in \{L_1, ..., L_n\}$ it holds that $L' \prec L$.

Definition 2.4 (Ordered Resolution Rules). *Let C, D be clauses and A be an atom. Consider the following inference rules:*

- *Ordered resolution:* $\dfrac{C \vee A \quad \neg A \vee D}{C \vee D}$ *if A is strictly maximal w.r.t. $\prec$ in $C \vee A$ and $\neg A$ is maximal w.r.t. $\prec$ in $\neg A \vee D$.*
- *Ordered (positive) factorization:* $\dfrac{C \vee A \vee A}{C \vee A}$ *if A is maximal w.r.t. $\prec$ in its clause.*

The clause $C \vee D$ generated by the resolution rule is called a resolvent *of $C \vee A$ and $\neg A \vee D$. The clause $C \vee A$ generated by the factorization rule is called a* factor *of $C \vee A \vee A$.*

Given a formula F in conjunctive normal form, described as a set of clauses, we can check its satisfiability by generating new resolvents and factors in a systematic way. A set of clauses N is saturated under $Res^{\prec}$ if every application of an inference rule yields a resolvent or factor that is already in N.

Theorem 2.5 (Soundness/Completeness [13]). *Let F be a formula in conjunctive normal form, described as a set of clauses.*

(1) The ordered resolution calculus is sound, i.e. if the empty clause ($\bot$) can be derived by $Res^{\prec}$, then F is unsatisfiable.

(2) The ordered resolution calculus is refutationally complete, i.e. if the empty clause is not included in the saturated set N_{sat} obtained by $Res^{\prec}$, then F is satisfiable.

Example 2.6 (Ordered Resolution). Let $R \prec Q \prec P$ be an atom ordering and let F be the CNF-formula described by the following set of clauses:

$$C_1 = P \vee Q \vee \neg R$$
$$C_2 = \neg P \vee Q$$
$$C_3 = \neg Q \vee \neg R$$
$$C_4 = R$$

- Resolution of C_1 and C_2 yields: $C_5 = Q \vee Q \vee \neg R$
- Factorization of C_5 yields: $C_6 = Q \vee \neg R$
- Resolution of C_3 and C_6 yields: $C_7 = \neg R \vee \neg R$
- Resolution of C_4 and C_7 yields: $C_8 = \neg R$
- Resolution of C_4 and C_8 yields: $C_9 = \bot$

Since $\bot$ was derived by $Res^{\prec}$ and the calculus is sound, we can conclude that F is unsatisfiable. ■

One can also define a resolution calculus without ordering, by discarding the conditions on the ordering for the resolution and factorization rules. In Chapter 5 we use a variant of resolution without ordering, where in addition we only allow inferences in which one clause is a positive unit clause (i.e. an atom). We refer to this variant of resolution as *positive unit resolution*. It is sound and complete for sets of Horn clauses (cf. [59]).

Interpolation

Interpolation in logic is a way of computing intermediate terms between two formulae in the common signature of the formulae. In the following we define the notion of Craig interpolation. When we talk about non-logical symbols we refer to all symbols except for logical operators and parantheses, i.e. non-logical symbols include variables, constants, function symbols and predicate symbols.

Definition 2.7 (Craig Interpolant). *Let F and G be formulae such that $F \to G$ is valid. A formula I is called a* Craig interpolant *if*

(i) $F \to I$ *is valid,*

(ii) $I \to G$ *is valid, and*

(iii) Every non-logical symbol in I occurs in both F and G.

In [36] Craig proved that for any two formulae of first-order logic there exists a Craig interpolant. It follows that the same holds for any two propositional formulae. Interpolants in propositional logic can be computed using ordered resolution. We now give an example of a Craig interpolant in propositional logic.

Example 2.8. Consider the following formulae in propositional logic:

$$F = \neg((P \wedge Q) \to R)$$
$$G = (\neg P \to S) \wedge (\neg Q \to S)$$

Clearly $F \to G$ is valid. Since $F \to P \wedge Q$ and $P \wedge Q \to (\neg P \to S) \wedge (\neg Q \to S)$ are valid, a Craig interpolant for F and G is $I = P \wedge Q$. ■

2.2 Theories

A logical theory can be seen as a collection of sentences, defining which formulae are supposed to be true in the theory. This can for instance be done by stating a set of axioms and considering all logical consequences of the axioms or by giving a set of models in which the statements of the theory hold. The formal definitions are given below.

A Π-*theory* $\mathcal{T}$ is a set of Π-sentences. A set $Ax_{\mathcal{T}}$ of Π-sentences is an *axiomatization* of a Π-theory $\mathcal{T}$ if and only if $\mathcal{T}$ and $Ax_{\mathcal{T}}$ have the same consequences, i.e. if for every Π-sentence F it holds that $\mathcal{T} \models F$ if and only if $Ax_{\mathcal{T}} \models F$. The elements of $Ax_{\mathcal{T}}$ are called *axioms*.

If F and G are formulae we write $F \models_{\mathcal{T}} G$ (also written as $\mathcal{T} \cup F \models G$) to say that every model of F which is also a model of $\mathcal{T}$ is a model of G. $F \models_{\mathcal{T}} \bot$ means that there is no model of $\mathcal{T}$ in which F is true. If there is a model of $\mathcal{T}$ which is also a model of F we say that F is $\mathcal{T}$-*consistent* (or satisfiable w.r.t. $\mathcal{T}$). If $F \models_{\mathcal{T}} G$ and $G \models_{\mathcal{T}} F$, we say that F and G are equivalent w.r.t. $\mathcal{T}$, denoted by $F \equiv_{\mathcal{T}} G$.

A theory $\mathcal{T}$ is called *locally finite* if for every finite set of variables X the set of terms $T_\Pi(X)$ contains only finitely many terms $t_1, ..., t_n$ with $\models_{\mathcal{T}} t_i \not\approx t_j$ for $i \neq j$.

Examples of theories are the theories of rational and real linear arithmetic ($\mathsf{LI}(\mathbb{Q})$, $\mathsf{LI}(\mathbb{R})$) and the theory of real closed fields (which will be defined in Section 2.3.1). These theories are not locally finite. An example of a locally finite theory is the theory of semilattices (which will be defined in Chapter 5).

Often we need to consider combinations or extensions of theories. Combining two theories $\mathcal{T}_1$ and $\mathcal{T}_2$ means that we combine the corresponding axioms of the theories. For instance, a theory of arrays can be regarded as the combination of an index theory (usually the theory of integers) and an element theory (for instance the theory of real numbers).

Sometimes we may have a base theory which is extended with additional function symbols which have certain properties (decribed by universal formulae). This is then called a theory extension. This could for instance be an extension of the theory of real numbers with a function $f : \mathbb{R} \to \mathbb{R}$ which has the property that it is monotone, i.e. it satisfies the formula

$$\forall x, y \ (x \leq y \to f(x) \leq f(y)).$$

Of special interest are theory extensions that have a "locality" property. In the following we define what a local theory extension is, show how to recognize them and describe a method for hierarchical reasoning in theory extensions.

2.2.1 Local Theory Extensions

The notion of *locality* was first introduced by Givan and McAllester in [54, 55], who looked into so-called *local inference relations* for which checking validity of ground Horn clauses can be done in polynomial time. Ganzinger continued this work in [48] by studying so-called *local theories* and established links between embeddability properties and locality. In [90] Sofronie-Stokkermans further generalized these results by introducing the notion of *local theory extensions*. In [63] Sofronie-Stokkermans et al. introduced the notion of Ψ-*local theory extensions*, where Ψ is a closure operator on ground terms, which were further studied by Ihlemann and Sofronie-Stokkermans in [64]. In the following these two notions will be defined.

Let $\Pi_0{=}(\Sigma_0, \mathsf{Pred})$ be a signature and $\mathcal{T}_0$ be a *base theory* with signature Π_0 (we call functions in Σ_0 *base functions*). We consider a *theory extension* $\mathcal{T}_1 \coloneqq \mathcal{T}_0 \cup \mathcal{K}$ of $\mathcal{T}_0$ with new function symbols in a set Σ disjoint from Σ_0 (which we call *extension functions*). The properties of the extension functions are axiomatized using a set $\mathcal{K}$ of clauses in the extended signature $\Pi_1 = (\Sigma_0 \cup \Sigma, \mathsf{Pred})$, which contain function symbols in Σ. If G is a finite set of ground Π_1^C-clauses (where $\Pi_1^C = (\Sigma_0 \cup \Sigma \cup \Sigma_C, \mathsf{Pred})$ is the extension of Π_1 with constants in a countable set Σ_C of fresh constants) and $\mathcal{K}$ a set of Π-clauses, we will denote by $\mathsf{st}(\mathcal{K}, G)$ the set of all ground terms which occur in G or $\mathcal{K}$, and by $\mathsf{est}(\mathcal{K}, G)$ the set of all extension ground terms (i.e. ground terms whose root is in Σ) which occur in G or $\mathcal{K}$. We here regard every finite set G of ground clauses as the ground formula $\bigwedge_{C \in G} C$.

We denote by $\mathcal{K}[G]$ the set of instances of $\mathcal{K}$ in which all terms starting with an extension function are ground terms in the set $\mathsf{est}(\mathcal{K}, G)$, i.e.

$$\begin{aligned}\mathcal{K}[G] = \{ C\sigma \mid\ & C \in \mathcal{K} \text{ and } \sigma : X \to X \cup T_{\Sigma \cup \Sigma_C} \text{ is such that} \\ & \text{(i) for each extension term } f(t) \text{ in } C, f(t)\sigma \in \mathsf{est}(\mathcal{K}, G) \\ & \text{(ii) } x\sigma = x \text{ if } x \text{ does not occur in an extension term}\}.\end{aligned}$$

Example 2.9. Let $\mathcal{T}_0$ be the theory of integers with signature $\Pi_0 = \{\Sigma_0, \mathsf{Pred}\}$, where $\Sigma_0 = \mathbb{Z}$ and $\mathsf{Pred} = \{\approx, \leq\}$. Let $\Pi_1 = (\Sigma_0 \cup \Sigma \cup \Sigma_C, \mathsf{Pred})$ be the extended signature, where $\Sigma = \{f, g\}$ and $\Sigma_C = \{a, b, c\}$. We consider the following set of clauses $\mathcal{K}$ and set of ground terms G:

$$\begin{aligned}\mathcal{K} &= \{\ \forall x\ f(x) \leq g(x),\ \ \forall x, y\ x \leq y \to f(x) \leq f(y)\ \} \\ G &= \{\ a \leq b, b \leq c, f(a) \approx 1, f(b) \approx 2, g(a) \approx g(c)\ \}\end{aligned}$$

Then the set of extension ground terms looks as follows:

$$\mathsf{est}(\mathcal{K}, G) = \{\ f(a), f(b), g(a), g(c)\ \}$$

We compute the set of instances $\mathcal{K}[G]$ as follows:

- Since $f(a) \in \mathsf{est}(\mathcal{K}, G)$ and $g(a) \in \mathsf{est}(\mathcal{K}, G)$, we have $f(a) \leq g(a)$ as an instance of the first clause in $\mathcal{K}[G]$.
- Since $f(a) \in \mathsf{est}(\mathcal{K}, G)$ and $f(b) \in \mathsf{est}(\mathcal{K}, G)$, we have four instances of the second clause in $\mathcal{K}[G]$, all instances of the form $x \leq y \to f(x) \leq f(y)$, where $x, y \in \{a, b\}$.

Therefore, we obtain the following set of instances $\mathcal{K}[G]$:

$$\begin{aligned}\mathcal{K}[G] = \{\ & f(a) \leq g(a),\ a \leq a \to f(a) \leq f(a), \\ & a \leq b \to f(a) \leq f(b), \\ & b \leq a \to f(b) \leq f(a), \\ & b \leq b \to f(b) \leq f(b)\ \}\end{aligned}$$

Definition 2.10 (Local Theory Extension). *A theory extension $\mathcal{T}_0 \cup \mathcal{K}$ is* local *if the condition ($\mathsf{Loc_f}$) is satisfied:*

($\mathsf{Loc_f}$) *For every finite set G of ground clauses in the signature Π_1^C it holds that $\mathcal{T}_0 \cup \mathcal{K} \cup G \models \bot$ if and only if $\mathcal{T}_0 \cup \mathcal{K}[G] \cup G$ is unsatisfiable.*

It is always true that $\mathcal{T}_0 \cup \mathcal{K}[G] \cup G \models \bot$ implies $\mathcal{T}_0 \cup \mathcal{K} \cup G \models \bot$. However, in general the satisfiability of $\mathcal{T}_0 \cup \mathcal{K}[G] \cup G$ does not imply the satisfiability of $\mathcal{T}_0 \cup \mathcal{K} \cup G$. If this is the case, i.e. we can restrict to a "local" part of the problem for checking (un-)satisfiability, we say that the theory extension has the locality property. Sometimes the instances $\mathcal{K}[G]$ computed from the set of ground terms $\mathsf{est}(\mathcal{K}, G)$ may not be sufficient for guaranteeing satisfiabilty. In this case we can try to find a suitable closure operator Ψ to compute a larger set of ground terms. If this guarantees satisfiability, then we call this property Ψ-locality.

Definition 2.11 (Term Closure Operator [64]). *Let Ψ be a map associating with $\mathcal{K}$ and a set of Π^C-ground terms T a set $\Psi_\mathcal{K}(T)$ of Π^C-ground terms. We call $\Psi_\mathcal{K}$ a* term closure operator *if the following holds for all sets of ground terms T and T':*

(1) $\mathsf{est}(\mathcal{K}, T) \subseteq \Psi_\mathcal{K}(T)$,

(2) If $T \subseteq T'$, *then* $\Psi_\mathcal{K}(T) \subseteq \Psi_\mathcal{K}(T')$,

(3) $\Psi_\mathcal{K}(\Psi_\mathcal{K}(T)) = \Psi_\mathcal{K}(T)$,

(4) for any map $h : C \to C$ *it holds that* $\overline{h}(\Psi_\mathcal{K}(T)) = \Psi_\mathcal{K}(\overline{h}(T))$, *where* $\overline{h}$ *is the canonical extension of* h *to extension ground terms.*

The important case, in which this criterion is helpful, is the case in which for every finite set T of ground terms the set $\Psi_\mathcal{K}(T)$ is also finite. This is the case we consider in what follows. Let the set $\Psi_\mathcal{K}(\mathsf{est}(\mathcal{K}, G))$ of ground terms derived using the closure operator $\Psi_\mathcal{K}$ be denoted by $\Psi_\mathcal{K}(G)$.

Definition 2.12 (Ψ-local Theory Extension [63, 64]). *A theory extension $\mathcal{T}_0 \cup \mathcal{K}$ is* Ψ-local *if the condition (Loc_f^Ψ) is satisfied:*

(Loc_f^Ψ) *For every finite set G of ground clauses in the signature Π_1^C it holds that $\mathcal{T}_0 \cup \mathcal{K} \cup G \models \bot$ if and only if $\mathcal{T}_0 \cup \mathcal{K}[\Psi_\mathcal{K}(G)] \cup G$ is unsatisfiable.*

Sometimes we require that the clauses in $\mathcal{K}$ and/or G are *flat* or *linear* w.r.t. Σ-functions. In the definition we make a distinction between ground and non-ground clauses.

Definition 2.13 (Flat, Linear [100]). *A non-ground clause D is* Σ-flat *if all symbols below a Σ-function symbol in D are variables. A non-ground clause D is* Σ-linear *if whenever a variable occurs in two terms of D whose root symbol is in Σ, the two terms are identical, and no term which starts with a Σ-function contains two occurrences of the same variable. A ground clause D is* Σ-flat *if all symbols below a Σ-function in D are constants. A ground clause D is* Σ-linear *if whenever a constant occurs in two terms of D whose root symbol is in Σ, the two terms are identical, and no term which starts with a Σ-function contains two occurrences of the same constant.*

2.2.2 Recognizing Local Theory Extensions

In [90] it was shown that (Ψ-)local extensions can be recognized by showing that certain partial models embed into total ones. Especially well-behaved are the theory extensions with the *completability* property, stating that partial models can be made total without changing the universe of the model. In the following we will define the embeddability and completability properties formally.

Definition 2.14 (Partial Structure [100]). *Let* $\Pi = (\Sigma, \mathsf{Pred})$ *be a signature. A* partial Π-structure *is a tuple* $(|\mathcal{A}|, \{f_{\mathcal{A}}\}_{f\in\Sigma}, \{P_{\mathcal{A}}\}_{P\in\mathsf{Pred}})$*, where*

- $|\mathcal{A}|$ *is a non-empty set,*
- *for every* n*-ary function symbol* $f \in \Pi$*,* $f_{\mathcal{A}}$ *is a partial function from* $|\mathcal{A}|^n$ *to* $|\mathcal{A}|$*,*
- *constants (0-ary function symbols) are always defined, and*
- *for every* k*-ary predicate symbol* $P \in \mathsf{Pred}$*,* $P_{\mathcal{A}}$ *is a subset of* $|\mathcal{A}|^k$*.*

The structure is a (total) algebra *if all functions* $f_{\mathcal{A}}$ *are total.*

Definition 2.15 (Weak Satisfiability [90]). *Let* $\mathcal{A}$ *be a partial* Π*-structure and let* $\beta : X \to |\mathcal{A}|$ *be a valuation for its variables.* $(\mathcal{A}, \beta)$ weakly satisfies a literal L *(notation:* $(\mathcal{A}, \beta) \models_w L$*) if and only if*

- $L = P(t_1, ..., t_n)$ *and either* $(\beta(t_1), ..., \beta(t_n)) \in P_{\mathcal{A}}$ *or* $\beta(t_i)$ *is undefined for some* i*;*
- $L = \neg P(t_1, ..., t_n)$ *and either* $(\beta(t_1), ..., \beta(t_n)) \notin P_{\mathcal{A}}$ *or* $\beta(t_i)$ *is undefined for some* i*.*

$(\mathcal{A}, \beta)$ weakly satisfies a clause C *(notation:* $(\mathcal{A}, \beta) \models_w C$*) if and only if it weakly satisfies one of the literals in* C*.*

Let $\mathcal{T}_0 \subseteq \mathcal{T}_1 = \mathcal{T}_0 \cup \mathcal{K}$ be a theory extension where the signature of the base theory $\mathcal{T}_0$ is $\Pi_0 = (\Sigma_0, \mathsf{Pred})$ and the extended signature of $\mathcal{T}_1$ is $\Pi_1 = (\Sigma_0 \cup \Sigma_1, \mathsf{Pred})$. Given a (partial or total) Π-structure $\mathcal{A}$ and $\Pi_0 \subseteq \Pi$ we denote by $\mathcal{A}|_{\Pi_0}$ the reduct of $\mathcal{A}$ to Π_0, i.e. the Π_0-structure obtained from $\mathcal{A}$ by forgetting the interpretations of function and predicate symbols from $\Pi \setminus \Pi_0$.

Definition 2.16 (Weak Partial Model [100]). *A partial* Π_1*-structure* $\mathcal{A}$ *is a* weak partial model *of* $\mathcal{T}_1$ *with totally defined* Σ_0*-functions if*

- $\mathcal{A}|_{\Pi_0}$ *is a total model of* $\mathcal{T}_0$ *and*
- $\mathcal{A}$ *weakly satisfies all clauses in* $\mathcal{K}$*.*

We denote by $\mathsf{PMod}_w(\Sigma_1, \mathcal{T}_1)$ *the set of all weak partial models of* $\mathcal{T}_1$ *with totally defined* Σ_1*-functions.*

Definition 2.17 (Weak Embedding [100]). *Let* $\Pi = (\Sigma, \mathsf{Pred})$*. A* weak Π-embedding *between two partial* Π*-structures* $\mathcal{A}$ *and* $\mathcal{B}$ *is a total injective map* $h : |\mathcal{A}| \to |\mathcal{B}|$ *such that:*

(i) For all $P \in \mathsf{Pred}$ *it holds that* $P_{\mathcal{A}}(a_1, ..., a_n)$ *if and only if* $P_{\mathcal{B}}(h(a_1), ..., h(a_n))$*.*

(ii) For all $f \in \Sigma$ *it holds that whenever* $f_{\mathcal{A}}(a_1, ..., a_n)$ *is defined (in* $\mathcal{A}$*), then* $f_{\mathcal{B}}(h(a_1), ..., h(a_n))$ *is defined (in* $\mathcal{B}$*) and* $h(f_{\mathcal{A}}(a_1, ..., a_n)) = f_{\mathcal{B}}(h(a_1), ..., h(a_n))$*.*

We now define the embeddability and completability properties (and also versions related to a closure operator Ψ) in a formal way:

($\mathsf{Emb_f}$) Every $\mathcal{A} \in \mathsf{PMod}_w(\Sigma_1, \mathcal{T}_1)$ where all functions have a finite domain of definition weakly embeds into a total model of $\mathcal{T}_1$.

($\mathsf{Comp_f}$) Every $\mathcal{A} \in \mathsf{PMod}_w(\Sigma_1, \mathcal{T}_1)$ where all functions have a finite domain of definition weakly embeds into a total model $\mathcal{B}$ of $\mathcal{T}_1$ such that $\mathcal{A}|_{\Pi_0}$ and $\mathcal{B}|_{\Pi_0}$ are isomorphic.

($\mathsf{Emb}_\mathsf{f}^\Psi$) Every $\mathcal{A} \in \mathsf{PMod}_w(\Sigma_1, \mathcal{T}_1)$ where $\{f(a_1, ..., a_n) \mid a_i \in A, f \in \Sigma_1, f_\mathcal{A}(a_1, ..., a_n)$ is defined$\}$ is finite and closed under Ψ weakly embeds into a total model of $\mathcal{T}_1$.

($\mathsf{Comp}_\mathsf{f}^\Psi$) Every $\mathcal{A} \in \mathsf{PMod}_w(\Sigma_1, \mathcal{T}_1)$ where $\{f(a_1, ..., a_n) \mid a_i \in A, f \in \Sigma_1, f_\mathcal{A}(a_1, ..., a_n)$ is defined$\}$ is finite and closed under Ψ weakly embeds into a total model $\mathcal{B}$ of $\mathcal{T}_1$ such that $\mathcal{A}|_{\Pi_0}$ and $\mathcal{B}|_{\Pi_0}$ are isomorphic.

Since completability is a stronger property than embeddability, it clearly follows that (Comp_f) implies (Emb_f) and ($\mathsf{Comp}_\mathsf{f}^\Psi$) implies ($\mathsf{Emb}_\mathsf{f}^\Psi$).

In [90] it was shown that under certain conditions (Emb_f) implies (Loc_f).

Theorem 2.18 ([90]). *Let $\mathcal{K}$ be a set of Σ_1-flat and Σ_1-linear clauses and let $\mathcal{T}_1 = \mathcal{T}_0 \cup \mathcal{K}$. Assume that $\mathcal{T}_0$ is a locally finite theory and $\mathcal{K}$ only contains finitely many ground subterms. If the extension $\mathcal{T}_0 \subseteq \mathcal{T}_1$ satisfies (Emb_f), then it satisfies (Loc_f).*

In [99] it was proved that ($\mathsf{Emb}_\mathsf{f}^\Psi$) implies ($\mathsf{Loc}_\mathsf{f}^\Psi$).

Theorem 2.19 ([99]). *Let $\mathcal{K}$ be a set of Σ_1-flat and Σ_1-linear clauses. If the extension $\mathcal{T}_0 \subseteq \mathcal{T}_1 = \mathcal{T}_0 \cup \mathcal{K}$ satisfies ($\mathsf{Emb}_\mathsf{f}^\Psi$), where Ψ is a term closure operator, then the extension satisfies ($\mathsf{Loc}_\mathsf{f}^\Psi$).*

Remark 2.20 ([99], [86]). *The Σ_1-linearity restriction in Theorem 2.19 can be relaxed. We can allow a variable x to occur below two unary function symbols g and h in a clause if Ψ has the property that for every constant c it holds that: $g(c) \in \Psi(G)$ if and only if $h(c) \in \Psi(G)$.*

Examples of Local Theory Extensions

The link between embeddability of partial models into total models and locality can be used to identify many classes of local theory extensions. In the following we present some examples.

Example 2.21 (Extensions with Free/Monotone Functions [90, 63]). The following types of extensions of a theory $\mathcal{T}_0$ are local:

(1) Any extension of $\mathcal{T}_0$ with uninterpreted function symbols (the extension satisfies condition (Comp_f)).

(2) Any extension of a theory $\mathcal{T}_0$ for which $\leq$ is a partial order with functions monotone w.r.t. $\leq$, i.e. functions satisfiying monotonicity axioms of the form

$$\mathsf{Mon}(f) = \forall x, y(x \leq y \rightarrow f(x) \leq f(y))$$

(the extension satisfies condition (Comp_f) if all models of $\mathcal{T}_0$ are lattices w.r.t. $\leq$). ■

Remark 2.22. *For the set of monotonicity axioms for functions in a set Σ we use the following notation:*

$$\mathit{Mon}(\Sigma) := \bigcup_{f \in \Sigma} \mathit{Mon}(f).$$

Example 2.23 (Extensions with Definitions [66, 63]). Any extension of a theory $\mathcal{T}_0$ with a new function symbol f is local if f is defined by case distinction with axioms of the form

$$\mathsf{Def}_f \coloneqq \{\forall \overline{x}(\phi_i(\overline{x}) \to F_i(f(\overline{x}), \overline{x})) \mid i = 1, \ldots, m\},$$

where ϕ_i and F_i with $i = 1, \ldots, m$ are formulae over the signature of $\mathcal{T}_0$ such that the following conditions hold:

(i) $\phi_i(\overline{x}) \wedge \phi_j(\overline{x}) \models_{\mathcal{T}_0} \bot$ for $i \neq j$ and

(ii) $\mathcal{T}_0 \models \forall \overline{x}(\phi_i(\overline{x}) \to \exists y(F_i(y, \overline{x})))$ for all $i \in \{1, \ldots, m\}$.

In particular the following types of extensions are local (and satisfy ($\mathsf{Comp_f}$)):

(1) Any extension with a function f defined by axioms of the form

$$\mathsf{D}_f \coloneqq \{\forall \overline{x}(\phi_i(\overline{x}) \to f(\overline{x}) \approx t_i) \mid i = 1, \ldots, n\} \ ,$$

where ϕ_i are formulae over the signature of $\mathcal{T}_0$ and t_i terms over the signature of $\mathcal{T}_0$ such that (i) holds.

(2) Any extension of $\mathcal{T}_0 \in \{\mathsf{LI}(\mathbb{Q}), \mathsf{LI}(\mathbb{R})\}$ with functions satisfying axioms

$$\mathsf{Bound}_f \coloneqq \{\forall \overline{x}(\phi_i(\overline{x}) \to s_i \leq f(\overline{x}) \leq t_i) \mid i = 1, \ldots, n\},$$

where ϕ_i are formulae over the signature of $\mathcal{T}_0$, s_i, t_i are Π_0-terms, condition (i) holds and $\models_{\mathcal{T}_0} \forall \overline{x}(\phi_i(\overline{x}) \to s_i \leq t_i)$. ■

Example 2.24 (The Array Property Fragment [24, 63]). In [24] a decidable fragment of the theory of arrays is studied, namely the *array property fragment*. Arrays are regarded as functions with arguments of sort index and values of sort elem_m with $m > 0$. The index theory $\mathcal{T}_i$ is Presburger arithmetic, with signature $\Pi_{\mathbb{Z}} = (\{0, 1, +, -\}, \{\approx, <, \leq\})$. The element theories $\mathcal{T}^1_{\mathsf{elem}}, ..., \mathcal{T}^m_{\mathsf{elem}}$ are parametric, with $\Pi^1_{\mathsf{elem}}, ..., \Pi^m_{\mathsf{elem}}$ being the corresponding signatures. The theory of arrays $\mathcal{T}_A$ has the signature $\Pi_A = \Pi_{\mathbb{Z}} \cup \bigcup_k \Pi^k_{\mathsf{elem}} \cup \{\cdot[\cdot], \cdot\{\cdot \leftarrow \cdot\}\}$ where $a[i]$ describes an array read, i.e. it returns the value of the array a at position i, and $a\{i \leftarrow e\}$ describes an array write, i.e. it returns the modified array a with value e at position i.

An *array property formula* has the form $\forall \bar{i}(\varphi_I(\bar{i}) \to \varphi_V(\bar{i}))$, where:

- φ_I is called the index guard, and it is a positive Boolean combination of atoms of the form $t \leq u$ or $t \approx u$, where t and u are either a variable or a ground term of sort index.
- φ_V is called the value constraint, and it has the property that any universally quantified variable x of sort index only occurs in a direct array read $a[x]$ in φ_V and there are no nested array reads.

The *array property fragment* consists of all existentially closed Boolean combinations of quantifier-free formulae and array property formulae. In [24] it is shown that formulae in the array property fragment have complete instantiation for satisfiability checking. In [63] it is shown that this fragment satisfies a Ψ-locality condition with the closure operator $\Psi(G) = \{a(i_1, ..., i_n) \mid a \text{ is an array name}, \ i_1, ..., i_n \in \mathcal{I}\}$ where $\mathcal{I}$ is the set of index terms used for complete instantiation in [24], i.e. the union of the set $\mathcal{R}$ of index terms at which some array is read and the set $\mathcal{B}$ of index terms that define boundaries on some array of an array property (additionally, if $\mathcal{R} = \mathcal{B} = \emptyset$, then $\mathcal{I} = \{0\}$). ■

2.2.3 Hierarchical Reasoning in Local Theory Extensions

For (Ψ)-local theory extensions hierarchical reasoning is possible. Here we discuss only the case of local theory extensions, but similar results hold also for Ψ-local theory extensions under the additional assumption that for every finite set G of ground clauses $\Psi_{\mathcal{K}}(G)$ is finite. If $\mathcal{T}_0 \cup \mathcal{K}$ is a local extension of $\mathcal{T}_0$ and G is a set of ground $\Sigma_0 \cup \Sigma_1 \cup \Sigma_C$-clauses, then $\mathcal{T}_0 \cup \mathcal{K} \cup G$ is unsatisfiable if and only if $\mathcal{T}_0 \cup \mathcal{K}[G] \cup G$ is unsatisfiable. We can reduce this last satisfiability test to a satisfiability test w.r.t. $\mathcal{T}_0$. The idea is to purify $\mathcal{K}[G] \cup G$ by applying the following steps:

(i) Introduce in a bottom-up manner new constants c_t for subterms $t = f(g_1, \ldots, g_n)$, where $f \in \Sigma$ and g_i are ground $\Sigma_0 \cup \Sigma_C$-terms, and replace the terms t with the constants c_t.

(ii) Add the definitions $c_t \approx t$ to a set Def.

We denote by $\mathcal{K}_0 \cup G_0 \cup \mathsf{Def}$ the set of formulae obtained this way. Then G is satisfiable w.r.t. $\mathcal{T}_0 \cup \mathcal{K}$ if and only if $\mathcal{K}_0 \cup G_0 \cup \mathsf{Con}_0$ is satisfiable w.r.t. $\mathcal{T}_0$, where Con_0 is the following set of instances of congruence axioms:

$$\mathsf{Con}_0 = \{(\bigwedge_{i=1}^{n} c_i \approx d_i) \rightarrow c \approx d \mid c \approx f(c_1, \ldots, c_n), d \approx f(d_1, \ldots, d_n) \in \mathsf{Def}\}.$$

Theorem 2.25 ([90]). *If $\mathcal{T}_1 = \mathcal{T}_0 \cup \mathcal{K}$ is a local theory extension of $\mathcal{T}_0$ and G is a finite set of ground clauses, then we can reduce the problem of checking whether G is satisfiable w.r.t. $\mathcal{T}_0 \cup \mathcal{K}$ to checking the satisfiability w.r.t. $\mathcal{T}_0$ of the set of formulae $\mathcal{K}_0 \cup G_0 \cup \mathsf{Con}_0$ constructed as explained above. If $\mathcal{K}_0 \cup G_0 \cup \mathsf{Con}_0$ belongs to a decidable fragment of $\mathcal{T}_0$, we can use the decision procedure for this fragment to decide whether $\mathcal{T}_0 \cup \mathcal{K} \cup G$ is unsatisfiable.*

As the size of $\mathcal{K}_0 \cup G_0 \cup \mathsf{Con}_0$ is polynomial in the size of G (for a given $\mathcal{K}$), locality allows us to express the complexity of the ground satisfiability problem w.r.t. $\mathcal{T}_1$ as a function of the complexity of the satisfiability of formulae w.r.t. $\mathcal{T}_0$.

In the following we illustrate the method for hierarchical reasoning in theory extensions described before on a small example.

Example 2.26. Let $\mathcal{T}_1 = \mathcal{T}_0 \cup \mathcal{K}$ be a theory extension, where $\mathcal{T}_0 = \mathsf{LI}(\mathbb{R})$ and

$$\begin{aligned}\mathcal{K} = \mathsf{Mon}(f, g) = \{&\forall x, y(x \leq y \rightarrow f(x) \leq f(y)),\\ &\forall x, y(x \leq y \rightarrow g(x) \leq g(y))\}.\end{aligned}$$

We want to show that

$$\mathcal{T}_0 \cup \mathcal{K} \models \forall x, y(x \leq y \rightarrow f(x) + g(x) \leq f(y) + g(y)).$$

For this we have to show that $\mathcal{T}_0 \cup \mathcal{K} \cup G$ is unsatisfiable, where

$$G = a \leq b \wedge (f(a) + g(a) > f(b) + g(b))$$

is the skolemized negation of the formula $\forall x, y(x \leq y \rightarrow f(x) + g(x) \leq f(y) + g(y))$.

From Example 2.21 we already know that the theory extension $\mathcal{T}_1 = \mathsf{LI}(\mathbb{R}) \cup \mathsf{Mon}(f, g)$ is local. We can therefore use hierarchical reasoning as described above. We show the reduction step by step:

Step 1: Use locality

From locality we know that $\mathcal{T}_0 \cup \mathcal{K} \cup G$ is unsatisfiable if and only if $\mathcal{T}_0 \cup \mathcal{K}[G] \cup G$ is unsatisfiable. The extension terms appearing in G are $f(a), f(b), g(a)$ and $g(b)$, therefore we have the following set of instances of $\mathsf{Mon}(f, g)$:

$$\begin{aligned}\mathsf{Mon}(f,g)[G] = \{\ & (a \leq b \rightarrow f(a) \leq f(b)),\\ & (a \leq b \rightarrow g(a) \leq g(b)),\\ & (b \leq a \rightarrow f(b) \leq f(a)),\\ & (b \leq a \rightarrow g(b) \leq g(a)),\\ & (a \leq a \rightarrow f(a) \leq f(a)),\\ & (a \leq a \rightarrow g(a) \leq g(a)),\\ & (b \leq b \rightarrow f(b) \leq f(b)),\\ & (b \leq b \rightarrow g(b) \leq g(b))\ \}.\end{aligned}$$

Remark: Since the relation $\leq$ on $\mathbb{R}$ is reflexive, the last four instances in the set above are trivially always true and are therefore not needed. Thus, we will omit these instances in the following steps. In other examples in this thesis similar redundant instances are always omitted from the start.

Step 2: Flattening and purification

We flatten and purify $\mathsf{Mon}(f, g)[G]$ by replacing every ground term whose root is an extension function (i.e. f or g) with a new constant. We obtain a set of definitions Def and a conjunction of formulae in the base signature, $\mathcal{T}_0 \cup \mathcal{K}_0 \cup G_0$, where:

$$\begin{aligned}\mathsf{Def} &= \{a_1 \approx f(a), a_2 \approx g(a), b_1 \approx f(b), b_2 \approx g(b)\}\\ \mathcal{K}_0 &= (a \leq b \rightarrow a_1 \leq b_1) \wedge (a \leq b \rightarrow a_2 \leq b_2) \wedge\\ &\quad\ (b \leq a \rightarrow b_1 \leq a_1) \wedge (b \leq a \rightarrow b_2 \leq a_2)\\ G_0 &= a \leq b \wedge (a_1 + a_2 > b_1 + b_2)\end{aligned}$$

Step 3: Reduction to testing satisfiability in $\mathcal{T}_0$

We know that G is unsatisfiable w.r.t. $\mathcal{T}_0 \cup \mathcal{K}$ if and only if $\mathcal{K}_0 \cup G_0 \cup \mathsf{Con}_0$ is unsatisfiable w.r.t. $\mathcal{T}_0$, where Con_0 consists of the instances of the congruence axioms containing only terms starting with f and g which occur in Def.

$$\mathsf{Con}_0 = \{a \approx b \rightarrow a_1 \approx b_1, a \approx b \rightarrow a_2 \approx b_2\}$$

Remark: For extensions with monotone functions the congruence axioms are not needed, since from $a \leq b \rightarrow f(a) \leq f(b)$ and $b \leq a \rightarrow f(b) \leq f(a)$ it immediately follows that $a \approx b \rightarrow f(a) \approx f(b)$. Therefore we can omit the congruence axioms whenever we have an extension with monotonicity axioms.

It is easy to see that $\mathcal{K}_0 \cup G_0 \cup \mathsf{Con}_0$ is unsatisfiable w.r.t. $\mathcal{T}_0$, as from $a \leq b$ it follows that $a_1 \leq b_1$ and $a_2 \leq b_2$, but then we clearly have $a_1 + a_2 \leq b_1 + b_2$, which contradicts $a_1 + a_2 > b_1 + b_2$. ■

Sometimes theories can be structured such that we have a chain of theory extensions $\mathcal{T}_0 \subseteq \mathcal{T}_1 \subseteq ... \subseteq \mathcal{T}_n$.

Chains of Theory Extensions

Consider the chain of theory extensions

$$\mathcal{T}_0 \subseteq \mathcal{T}_1 = \mathcal{T}_0 \cup \mathcal{K}_1 \subseteq \mathcal{T}_2 = \mathcal{T}_0 \cup \mathcal{K}_1 \cup \mathcal{K}_2 \subseteq ... \subseteq \mathcal{T}_n = \mathcal{T}_0 \cup \mathcal{K}_1 \cup ... \cup \mathcal{K}_n$$

with signatures

$$\Pi_0 = (\Sigma_0, \mathsf{Pred}) \subseteq \Pi_1 = (\Sigma_0 \cup \Sigma_1, \mathsf{Pred}) \subseteq ... \subseteq \Pi_n = (\Sigma_0 \cup \Sigma_1 \cup ... \cup \Sigma_n, \mathsf{Pred}).$$

If every theory $\mathcal{T}_i$ with $i \in \{1, ..., n\}$ defines a local theory extension of $\mathcal{T}_{i-1}$ and every variable in $\mathcal{K}_i$ occurs under a function symbol in Σ_i, then hierarchical reasoning is possible and we can reduce the satisfiability test w.r.t. $\mathcal{T}_n$ in a top-down manner to a satisfiabilty test w.r.t. the base theory $\mathcal{T}_0$ in n steps.

Example 2.27 (Hierarchical Reasoning in Chains of Local Theory Extensions). Let $\mathcal{T}_1 = \mathcal{T}_0 \cup \mathcal{K}_1 \cup \mathcal{K}_2$ be a theory extension with signature $\Pi_1 = (\Sigma_1 \cup \Sigma_2, \mathsf{Pred})$, where

$$\begin{aligned}
\mathcal{T}_0 &= \mathsf{LI}(\mathbb{R}),\\
\mathcal{K}_1 &= \mathsf{Mon}(f) = \{\forall x, y(x \leq y \rightarrow f(x) \leq f(y)\},\\
\mathcal{K}_2 &= \mathsf{D}_g = \{\forall x(x < c \rightarrow g(x) \approx c, \forall x(x \geq c \rightarrow g(x) \approx f(x)\},\\
\Sigma_1 &= \{f\},\\
\Sigma_2 &= \{g\},\\
\mathsf{Pred} &= \{\approx, <, \leq, >, \geq\}.
\end{aligned}$$

We want to show that $\mathcal{T}_0 \cup \mathcal{K}_1 \cup \mathcal{K}_2 \models f(c) \geq c \rightarrow \forall x(g(x) \geq c)$. For this we have to show that $\mathcal{T}_0 \cup \mathcal{K}_1 \cup \mathcal{K}_2 \cup G$ is unsatisfiable, where

$$G = f(c) \geq c \wedge g(a) < c$$

is the skolemized negation of the formula $f(c) \geq c \rightarrow \forall x(g(x) \geq c)$. From Example 2.21 and Example 2.23 we know that the theory extensions $\mathcal{T}_1 = \mathsf{LI}(\mathbb{R}) \cup \mathsf{Mon}(f)$ and $\mathcal{T}_2 = \mathcal{T}_1 \cup \mathsf{D}_g$ are local. We can therefore use hierarchical reasoning. We first consider the theory extension $\mathcal{T}_1 \subseteq \mathcal{T}_2 = \mathcal{T}_1 \cup \mathsf{D}_g$:

From locality we know that $\mathcal{T}_1 \cup \mathsf{D}_g \cup G$ is unsatisfiable if and only if $\mathcal{T}_1 \cup \mathsf{D}_g[G] \cup G$ is unsatisfiable. The only extension term appearing in G is $g(a)$, therefore we have the following set of instances of D_g:

$$\mathsf{D}_g[G] = \{(a < c \rightarrow g(a) \approx c), (a \geq c \rightarrow g(a) \approx f(a))\}.$$

We flatten and purify $\mathsf{D}_g[G]$ by replacing every ground term whose root is an extension function (i.e. g) with a new constant. We obtain:

$$\begin{aligned}
\mathsf{Def} &= \{a_2 \approx g(a)\}\\
\mathsf{D}_{g_1} &= (a < c \rightarrow a_2 \approx c) \wedge (a \geq c \rightarrow a_2 \approx f(a))\\
G_1 &= f(c) \geq c \wedge a_2 < c
\end{aligned}$$

The instances of the congruence axioms Con_1 are not needed here because we have only one extension term $g(a)$. We can now reduce it further to $\mathcal{T}_0$ by using hierarchical reasoning on the theory extension $\mathcal{T}_0 \subseteq \mathcal{T}_1 = \mathcal{T}_0 \cup \mathsf{Mon}(f)$:

Let $G' = G_1 \cup \mathsf{D}_{g_1}$. From locality we know that $\mathcal{T}_0 \cup \mathsf{Mon}(f) \cup G'$ is unsatisfiable if and only if $\mathcal{T}_0 \cup \mathsf{Mon}(f)[G'] \cup G'$ is unsatisfiable. The only extension terms appearing in G' are $f(a)$ and $f(c)$, therefore we have the following set of non-redundant instances of $\mathsf{Mon}(f)$:

$$\mathsf{Mon}(f)[G'] = \{(a \leq c \rightarrow f(a) \leq f(c)), (c \leq a \rightarrow f(c) \leq f(a))\}.$$

We flatten and purify $\mathsf{Mon}(f)[G']$ by replacing every ground term whose root is an extension function (i.e. f) with a new constant. We obtain:

$$\begin{aligned}
\mathsf{Def} &= \{a_2 \approx g(a)\} \\
\mathsf{Def}' &= \{a_1 \approx f(a), c_1 \approx f(c)\} \\
\mathsf{Mon}(f)_0 &= (a \leq c \rightarrow a_1 \leq c_1), (c \leq a \rightarrow c_1 \leq a_1) \\
G'_0 &= c_1 \geq c \wedge a_2 < c \wedge (a < c \rightarrow a_2 \approx c) \wedge (a \geq c \rightarrow a_2 \approx a_1)
\end{aligned}$$

We know that G is unsatisfiable w.r.t. $\mathcal{T}_0 \cup \mathsf{Mon}(f)$ if and only if $\mathsf{Mon}(f)_0 \cup G_0 \cup \mathsf{Con}_0$ is unsatisfiable w.r.t. $\mathcal{T}_0$, where Con_0 consists of the congruence axioms containing only terms starting with f which occur in Def'.

$$\mathsf{Con}_0 = \{a \approx c \rightarrow a_1 \approx c_1\}$$

We can show that $\mathsf{Mon}(f)_0 \cup G_0 \cup \mathsf{Con}_0$ is unsatisfiable w.r.t. $\mathcal{T}_0$ by case distinction:

Case 1: $a < c$. Then from G'_0 we immediately obtain $a_2 \approx c$, which is in contradiction to $a_2 < c$.

Case 2: $a \geq c$. Then from G'_0 we obtain $a_2 \approx a_1$ and from $\mathsf{Mon}(f)_0$ we get $c_1 \leq a_1$, so we have $c_1 \leq a_2$, which together with $c_1 \geq c \wedge a_2 < c$ from G'_0 yields a contradiction.

An automated proof is also possible, by using a prover for the base theory $\mathcal{T}_0$. ■

Hierarchical reasoning is used to reduce a (satisfiability) problem to a problem in the given base theory. The reduced problem can then be solved if efficient reasoning methods for the base theory exist. Let $\forall \overline{x} F(\overline{x}, c_1, ..., c_n)$ be the formula obtained after the hierarchical reduction. The following are equivalent:

(i) $\forall \overline{x} F(\overline{x}, c_1, ..., c_n)$ is unsatisfiable

(ii) $\exists c_1, ..., c_n \ \forall \overline{x} F(\overline{x}, c_1, ..., c_n) \equiv \bot$

Note that if every variable in $\mathcal{K}$ occurs below an extension function, then we reduce the reasoning task to the task of checking satifiability w.r.t. $\mathcal{T}_0$ of a ground formula, i.e. we have $F(c_1, ..., c_n)$ instead of $\forall \overline{x} F(\overline{x}, c_1, ..., c_n)$ in (i) and (ii).

Provided that the base theory allows quantifier elimination we can check (ii) using quantifier elimination, so we can check satisfiability of the reduced formula and therefore of the original problem. Besides checking satisfiability, quantifier elimination can als be used for generating constraints on parameters, i.e. if the formula $\forall \overline{x} F(\overline{x}, c_1, ..., c_n)$ is satisfiable, we can use it to find a formula Γ such that $\Gamma \wedge F(\overline{x}, c_1, ..., c_n)$ is unsatisfiable. In the following we present the details of quantifier elimination.

2.3 Quantifier Elimination

The goal of quantifier elimination (QE) is to eliminate all quantifiers and quantified variables of a logical formula to obtain a simpler formula that is equivalent to the original one. Quantifier elimination can for instance be used for satisfiability checking.

Definition 2.28 (Quantifier Elimination). *A theory $\mathcal{T}$ over signature* Π allows quantifier elimination *if for every formula $\phi \in F_\Pi(X)$ there exists a quantifier-free formula $\phi^* \in F_\Pi(X)$ which is equivalent to ϕ modulo $\mathcal{T}$.*

Examples of theories which allow quantifier elimination are rational and real linear arithmetic ($\mathsf{LI}(\mathbb{Q})$, $\mathsf{LI}(\mathbb{R})$), the theory of real closed fields, and the theory of absolutely free data structures. Presburger arithmetic with an additional divisibility predicate allows quantifier elimination as well (cf. [22]). Of particular interest for the applications considered in this thesis are quantifier elimination methods for the real numbers, i.e. the theory of real closed fields, which we will refer to simply as real quantifier elimination.

Example 2.29. In the real numbers the following hold:

(1) The formula $\exists x(a \cdot x + b \approx 0)$ is equivalent to the quantifier-free formula

$$(a \approx 0 \wedge b \approx 0) \vee (a \not\approx 0).$$

(2) The formula $\exists x(a \cdot x^2 + b \cdot x + c \approx 0)$ is equivalent to the quantifier-free formula

$$(a \not\approx 0 \wedge b^2 - 4ac \geq 0) \vee (a \approx 0 \wedge b \not\approx 0) \vee (a \approx 0 \wedge b \approx 0 \wedge c \approx 0).$$

(3) The formula $\forall x(a \cdot x - b \cdot x \approx 0)$ is equivalent to the quantifier-free formula

$$a - b \approx 0.$$ ■

Remark: If we have a formula φ with nested quantifiers we can use the following transformations on φ to simplify the quantifier elimination problem:

(1) Transform φ to prenex normal form: $Q_1 x_1 ... Q_n x_n(\varphi')$, where φ' is quantifier free.

(2) Eliminate the quantifiers one after the other, starting with the innermost (Q_n).

(3) If it is a universal quantifier, transform it: $\forall x(\varphi') \rightsquigarrow \neg\exists x(\neg\varphi')$.

The problem is reduced to eliminating x from $\exists x(\varphi'')$ (where φ'' is either φ' or $\neg\varphi'$).

(4) Transform the quantifier-free formula φ'' to disjunctive normal form: $\bigvee_i \bigwedge_j \varphi_{ij}$.

(5) Bring the existential quantifier inside the disjunction: $\bigvee_i \exists x \bigwedge_j \varphi_{ij}$.

With these transformations we have reduced the problem to the case where only existential quantifiers in front of a conjunction of literals need to be eliminated. Note that not all transformations lead to a simplification of the problem all the time. Especially steps (1) and (4) can potentially cost more effort than they save.

In what follows we will first present a method for real quantifier elimination, then discuss quantifier elimination in theories with equality.

2.3.1 Real Quantifier Elimination

Since in many applications we have problems including real numbers, we need a suitable method for quantifier elimination in this case. There exist several methods for real quantifier elimination: In 1948 the first one was presented by Tarski [107], in 1975 Collins introduced cylindrical algebraic decomposition [29], and in 1988 Weispfenning proposed the virtual substitution [111]. In this thesis we present the method of Weispfenning for linear formulae in detail, as this method is implemented in the software which will be used throughout the thesis, namely the Redlog package in Reduce. Note that we will describe the idea of virtual substitution for the case of linear equations and disequations, but methods for handling the quadratic case and heuristics for even higher degrees do exist and are implemented in Redlog [112, 42, 105].

Definition 2.30 (Theory of Real Closed Fields [61]). *A field K is called a* real closed field *if in addition to the field axioms the following axioms hold:*

(1) $\forall x_1, ..., x_n \in K : x_1^2 + ... + x_n^2 \not\approx -1$ *for every positive natural number n*

(2) $\forall x \in K\ \exists y \in K : (x \approx y^2 \vee -x \approx y^2)$

(3) $\forall a_1, ..., a_n \in K\ \exists x \in K : x^n + a_1 \cdot x^{n-1} + ... + a_{n-1} \cdot x + a_n \approx 0$ *for every number n that is odd*

The theory axiomatized by the above axioms together with the field axioms is called the theory of real closed fields *and denoted by $\mathcal{T}_{\mathcal{RCF}}$.*

The field of real numbers $\mathbb{R}$ with the usually defined operations is a real closed field, since all axioms hold. The field of complex numbers $\mathbb{C}$ is not a real closed field, since axiom (1) is not satisfied. The field of rational numbers $\mathbb{Q}$ is not a real closed field either, since axioms (2) and (3) are not satisfied.

Definition 2.31 (Signature of Ordered Rings [104]). *The* signature of ordered rings *is* $S_{OR} = (\{0/0,\ \ 1/0,\ \ +/2,\ \ -/1,\ \ \cdot/2\}, \{\approx/2,\ \ \leq/2\})$.

In the signature S_{OR} and the theory $\mathcal{T}_{\mathcal{RCF}}$ also the relations $\not\approx$, $<$, $>$, $\geq$ and a division function div can be defined:

(1) $x \not\approx y$ if and only if $\neg(x \approx y)$

(2) $x < y$ if and only if $x \leq y \wedge x \not\approx y$

(3) $x > y$ if and only if $\neg(x \leq y)$

(4) $x \geq y$ if and only if $\neg(x < y)$

(5) $\mathsf{div}(x, y, z) \coloneqq y \not\approx 0 \wedge z \cdot y \approx x$

Virtual Substitution

We assume that we have a formula $\varphi = \exists x(F)$ with F being quantifier-free and only containing linear equalities ($\approx$), disequalities ($\not\approx$) and inequalities ($<, \leq, >, \geq$). The idea of virtual substitution is to replace $\exists x(F)$, which can be regarded as an infinite disjunction over the real numbers, by a finite disjunction of formulae not containing x. For this we decompose the set $\mathbb{R}$ of real numbers into a finite number of intervals such that the truth

value of the formula can only change on the endpoints of the intervals, i.e. inside an interval the truth value remains the same. Then we only need to take one suitable test point from each interval and can replace the quantified variables in the formula with those. The method is named after this (virtual or modified) substitution.

We first give definitions for the main notions of virtual substitution together with examples. We consider $\varphi = \exists x(F(x, a_1, ..., a_n))$, where $a_1, ..., a_n$ are free variables in F and F is a quantifier-free formula. We assume that the atomic formulae of F are linear equalities, disequalities and inequalities over the real numbers, i.e. we have the set of predicates $P = \{\approx, \not\approx, <, \leq, >, \geq\}$. Every (linear) equality, disequality and inequality $t \rhd s$ with $\rhd \in P$ can equivalently be written in the form $p \rhd 0$, where p is a (linear) polynomial, so we assume atoms to be of this form.

A formula is called *positive* if it does not contain any negation. Without loss of generality we can assume F to be a positive formula, since every non-positive formula over the real numbers can easily be transformed to a positive formula. For this we just have to bring F to negation normal form and eliminate the negations in front of atoms by changing the predicate accordingly. For instance, $\neg(x \not\approx 3) \equiv x \approx 3$ and $\neg(x < y) \equiv x \geq y$.

Definition 2.32 (Solution Candidate [40]). *The zeroes of the polynomials occurring in the atomic formulae of F are called* solution candidates.

Remark 2.33. *Since we need a distinction between the different predicates used in an atomic formula later on, our set of solution candidates contains the zeroes in the form of equalities, disequalities and inequalities, i.e. for an atomic formula $p(x) \rhd 0$, where $p(x)$ is a polynomial in the variable x and $\rhd \in \{\approx, \not\approx, <, \leq, >, \geq\}$, the zero z of $p(x)$ is given in the form $x \rhd z$.*

Example 2.34. Let $F = 2x - 4 \approx 0 \wedge x - 1 \geq 0 \wedge x + a < 0$ and x the quantified variable in $\exists x F(x, a)$. Then 2, 1 and $-a$ are the solution candidates. Therefore, based on the remark above, we obtain the following set of solution candidates:

$$C = \{x \approx 2, x \geq 1, x < -a\}.$$ ■

The solution candidates are the endpoints of the intervals, so if we have n solution candidates, this describes a decomposition of the set of real numbers into $n + 1$ intervals. From each interval we need one test point. Test points may contain the additional symbols ∞ and ε. We call a term containing one of these symbols a *non-standard term*.

Definition 2.35 (Test Point, Guard [40]). *A* test point *is a term (this includes non-standard terms) that the quantified variable is replaced with. A* guard *is a quantifier-free formula describing a condition which assures that the test point is always defined.*

Example 2.36. Test points could for instance be 2, $3a$, $-a + b$ or $\frac{5}{a-1}$. The first three test points do not need guards. However, the term $\frac{5}{a-1}$ is only defined if the denominator is not zero, therefore we need $\gamma = a \not\approx 1$ as a guard. $-\infty$, ∞, $5 + \varepsilon$, $\frac{1}{a} - \varepsilon$ are also possible test points, of which the latter needs $\gamma = a \not\approx 0$ as a guard. Those four test points are non-standard terms, as they contain the symbols ∞ or ε. How to compute test points from the solution candidates is explained in Definition 2.39. ■

Definition 2.37 (Test Term [40]). *A* test term *is a tuple* (γ, t)*, where* γ *is a guard and* t *is a test point.*

Example 2.38. $(a \neq 1, \frac{5}{a-1})$ is a test term. $(\top, 5 + \varepsilon)$ is also a test term. Since no guard is needed for the test point, we set $\gamma = \top$. ■

Definition 2.39 (Elimination Set [40]). *An* elimination set *is a set of* test terms*, which is derived from the solution candidates as follows:*

Let $A = \{x \rhd_i s_i \mid i \in I\}$ *be the set of all solution candidates with* $\rhd_i \in \{\approx, \not\approx, \leq <, \geq, >\}$*. Then the set of test terms is*

$$T \coloneqq \{-\infty\} \cup \{s_i \mid \rhd_i \in \{\geq, \approx\}\} \cup \{s_i + \varepsilon \mid \rhd_i \in \{>, \not\approx\}\}.$$

Together with the corresponding guards for each test point we derive the elimination set.

The computation of the elimination set described in Definition 2.39 is based on traversing the real numbers from $-\infty$ to $+\infty$ and taking for every interval its leftmost point. These are the points where the truth value of the formula can change from true to false or conversely. It is also possible to traverse the real numbers in the opposite direction, from $+\infty$ to $-\infty$, and take the rightmost point from each interval, which then leads to the following set of test points:

$$T' \coloneqq \{+\infty\} \cup \{s_i \mid \rhd_i \in \{\leq, \approx\}\} \cup \{s_i - \varepsilon \mid \rhd_i \in \{<, \not\approx\}\}.$$

Let F be a formula, x a variable, $\overline{a}$ parameters and t a term, then with $\sigma = F(x, \overline{a})[x//t]$ we denote the virtual substitution of x with t. The difference to standard substitution is that we may also substitute with non-standard terms. For distinction we use the notation "//" for virtual substitution in contrast to "/" for standard substitution. When substituting test terms (including non-standard terms) into atomic formulae the following conditions have to hold for the substitution result σ (cf. [40]):

(1) σ is a quantifier-free formula over the signature of ordered rings.

(2) σ is equivalent to the formal substitution result w.r.t. the extended signature (i.e. the signature of ordered rings extended by the non-standard terms ∞ and ε).

To obtain formulae satisfying both conditions we need to eliminate the non-standard symbols ∞ and ε from the formal substitution result. For this we use the following simplification rules:

Definition 2.40 (Simplification Rules [89]). *After the substitution of the quantified variables the symbols* ∞ *and* ε *can occur in the formula. We use the following rules to eliminate these symbols:*

(1) $(x < s)[x//-\infty] \coloneqq \lim_{r \to -\infty} (r < s) = \top$

(2) $(x \leq s)[x//-\infty] \coloneqq \lim_{r \to -\infty} (r \leq s) = \top$

(3) $(x > s)[x//-\infty] \coloneqq \lim_{r \to -\infty} (r > s) = \bot$

(4) $(x \geq s)[x//-\infty] \coloneqq \lim_{r \to -\infty} (r \geq s) = \bot$

(5) $(x \approx s)[x//-\infty] \coloneqq \lim_{r\to-\infty}(r \approx s) = \bot$

(6) $(x \not\approx s)[x//-\infty] \coloneqq \lim_{r\to-\infty}(r \not\approx s) = \top$

(7) $(x < s)[x//\, u+\varepsilon] \coloneqq \lim_{\varepsilon\to 0,\varepsilon>0}(u+\varepsilon < s) = (u < s)$

(8) $(x \leq s)[x//\, u+\varepsilon] \coloneqq \lim_{\varepsilon\to 0,\varepsilon>0}(u+\varepsilon \leq s) = (u < s)$

(9) $(x > s)[x//\, u+\varepsilon] \coloneqq \lim_{\varepsilon\to 0,\varepsilon>0}(u+\varepsilon > s) = (u \geq s)$

(10) $(x \geq s)[x//\, u+\varepsilon] \coloneqq \lim_{\varepsilon\to 0,\varepsilon>0}(u+\varepsilon \geq s) = (u \geq s)$

(11) $(x \approx s)[x//\, u+\varepsilon] \coloneqq \lim_{\varepsilon\to 0,\varepsilon>0}(u+\varepsilon \approx s) = \bot$

(12) $(x \not\approx s)[x//\, u+\varepsilon] \coloneqq \lim_{\varepsilon\to 0,\varepsilon>0}(u+\varepsilon \not\approx s) = \top$

In Algorithm 1 we present the algorithm for virtual substitution in the linear case.

Algorithm 1 Virtual Substitution [40]

Input: Formula $\varphi = \exists x F(x, \overline{a})$, where F is a quantifier-free formula containing only linear equalities and disequalities

Output: Quantifier-free formula φ' which is euivalent to φ

Step 1 Let A be the set of atomic formulae of F.

Step 2 Compute the set of solution candidates C from A (i.e. the zeroes of the polynomials).

Step 3 Compute the elimination set E from C according to Definition 2.39.

Step 4 Virtually substitute the terms from E in F (using the rules in Definition 2.40):

$$\exists x F(x, \overline{a}) \leftrightarrow \bigvee_{(\gamma,t)\in E} \gamma \wedge F(x, \overline{a})[x//t].$$

We now show for a given formula with two existential quantifiers how to compute an equivalent quantifier-free formula using Algorithm 1.

Example 2.41. Consider the formula $\exists x_2 \exists x_1 F(x_1, x_2, a)$, where a is a parameter and $F(x_1, x_2, a)$ is the following quantifier-free formula:

$$F(x_1, x_2, a) = (x_1 - x_2 \leq 0) \ \wedge \ (a x_2 \approx 1) \ \wedge \ \neg(x_2 > 1) \ \wedge \ (-x_1 + 2x_2 + 3 < 0)$$

Our goal is to eliminate the quantifiers and the quantified variables x_1 and x_2 in F. We start with the innermost quantifier, i.e. $\exists x_1 F(x_1, x_2, a)$:

(1) All subformulae of the conjunction except for $\neg(x_2 > 1)$ are atomic formulae. This formula is equivalent to the atomic formula $x_2 \leq 1$. We obtain:

$$A = \{x_1 - x_2 \leq 0, \quad a x_2 \approx 1, \quad x_2 \leq 1, \quad -x_1 + 2x_2 + 3 < 0\}.$$

(2) We solve the equations and disequations from A for x_1. Atoms which do not contain x_1 can be disregarded. We obtain the following set of solution candidates :

$$C = \{x_1 \leq x_2, \quad x_1 > 2x_2 + 3\}$$

(3) $-\infty$ is always a test point. The test point $2x_2+3+\varepsilon$ is derived from the second solution candidate. Since for these two test points no additional conditions are needed, we can set the guards to $\gamma = \top$. We obtain the following elimination set:

$$E = \{(\top, -\infty), (\top, 2x_2 + 3 + \varepsilon)\}$$

(4) We substitute, form the disjunction and simplify the obtained formula:

$$\begin{aligned}
F'(x_2, a) \coloneqq\ & (\top \ \wedge\ (-\infty \leq x_2) \ \wedge\ (ax_2 \approx 1) \ \wedge\ (x_2 \leq 1) \ \wedge\ (-\infty > 2x_2 + 3)) \vee \\
& (\top \ \wedge\ (2x_2 + 3 + \varepsilon \leq x_2) \ \wedge\ (ax_2 \approx 1) \ \wedge\ (x_2 \leq 1) \ \wedge \\
& (2x_2 + 3 + \varepsilon > 2x_2 + 3)) \\
\equiv\ & (\top \ \wedge\ (ax_2 \approx 1) \ \wedge\ (x_2 \leq 1) \ \wedge\ \bot) \vee \\
& ((2x_2 + 3 < x_2) \ \wedge\ (ax_2 \approx 1) \ \wedge\ (x_2 \leq 1) \ \wedge\ (2x_2 + 3 \geq 2x_2 + 3)) \\
\equiv\ & \bot \vee ((x_2 + 3 < 0) \ \wedge\ (ax_2 \approx 1) \ \wedge\ (x_2 \leq 1) \ \wedge\ \top) \\
\equiv\ & (x_2 + 3 < 0) \ \wedge\ (ax_2 \approx 1) \ \wedge\ (x_2 \leq 1)
\end{aligned}$$

Therefore we have $\exists x_2 \exists x_1 F(x_1, x_2, a) \equiv \exists x_2 F'(x_2, a)$.

We now eliminate the quantifier in $\exists x_2\ ((x_2 + 3 < 0) \ \wedge\ (ax_2 \approx 1) \ \wedge\ (x_2 \leq 1))$.

(1) All subformulae in the conjunction above are atomic, so we have the following set of atomic formulae:

$$A = \{x_2 + 3 < 0, \quad ax_2 \approx 1, \quad x_2 \leq 1\}$$

(2) Solving the equations and disequations for x_2 we obtain the following set of solution candidates:

$$C = \{x_2 < -3, \quad x_2 \approx \tfrac{1}{a}, \quad x_2 \leq 1\}$$

(3) Besides $-\infty$ we obtain from the second solution candidate the test point $\frac{1}{a}$. We need the guard guard $\gamma = a \not\approx 0$ to ensure that this term is always defined. Therefore the elimination set is:

$$E = \{(\top, -\infty), (a \not\approx 0, \tfrac{1}{a})\}$$

(4) Virtual substitution and simplification yields the following:

$$\begin{aligned}
F''(a) =\ & (\top \wedge\ (-\infty < -3) \ \wedge\ \left(-\infty \approx \frac{1}{a}\right) \ \wedge\ (-\infty \leq 1)) \vee \\
& ((a \not\approx 0) \ \wedge\ \left(\frac{1}{a} < -3\right) \ \wedge\ \left(\frac{1}{a} \approx \frac{1}{a}\right) \ \wedge\ \left(\frac{1}{a} \leq 1\right)) \\
\equiv\ & (\top \ \wedge\ \bot \ \wedge\ \top) \vee \\
& ((a \not\approx 0) \ \wedge\ \left(\frac{1}{a} < -3\right) \ \wedge\ \top \ \wedge\ \left(\frac{1}{a} \leq 1\right)) \\
\equiv\ & (a \not\approx 0) \ \wedge\ (3a^2 + a < 0) \ \wedge\ (a \leq a^2)
\end{aligned}$$

Both quantifiers and the corresponding variables are eliminated and as a result we have $\exists x_2 \exists x_1 F(x_1, x_2, a) \ \equiv\ F''(a) = a \not\approx 0 \ \wedge\ 3a^2 + a < 0 \wedge a \leq a^2$. ■

Complexity

Time complexity for quantifier elimination over the real numbers is in general (i.e. for arbitrary alternations of quantifiers and arbitrary degrees of the formulae) doubly exponential in the size of the input formula [37]. The method of Tarski is not elementary recursive. Cylindrical algebraic decomposition is doubly exponential in the number of all variables [25]. Virtual substitution for linear formulae is doubly exponential in the number of quantifier alternations [111]. In the applications we consider in this thesis we usually have only existential quantifiers, i.e. no quantifier alternations, and often only linear formulae. This makes the virtual substitution method efficient for our purposes.

2.3.2 Quantifier Elimination in Theories with Equality

In this section we analyze quantifier elimination results for theories with equality. We start with the theory of pure equality, denoted by $\mathcal{E}$. This theory has the signature $\Pi = (\emptyset, \{\approx\})$, i.e. it has no function symbols and equality $\approx$ as the only predicate symbol, and contains no non-logical axioms. In this theory we only have atoms of the form $x \approx y$, where x and y are constants or variables. Formulae can be built in the usual way, as boolean combinations of the atoms and with universal and existential quantification. Unfortunately, the theory $\mathcal{E}$ does not allow quantifier elimination, as the following example shows.

Example 2.42. Consider the formula $F = \exists x(\neg(x \approx c_1) \wedge ... \wedge \neg(x \approx c_n))$. There does not exist a quantifier-free formula F' which is equivalent to F w.r.t. $\mathcal{E}$, since the validity of F in a model $\mathcal{A}$ of $\mathcal{E}$ depends on the size of the universe of $\mathcal{A}$. If the universe only contains the n elements c_1,...,c_n, then F is false in $\mathcal{A}$, but if it contains more than n elements, then F is true in $\mathcal{A}$. ■

While the theory of pure equality does not allow quantifier elimination, there is a closely related theory which does. We consider the theory of an infinite set, denoted by $\mathcal{T}_{\mathcal{IS}}$, which has the same signature as $\mathcal{E}$ (i.e. no function symbols and $\approx$ as the only predicate symbol), but has as models only Π-structures with an infinite universe. Hence, every model of $\mathcal{T}_{\mathcal{IS}}$ has infinitely many elements, so the formula $F = \exists x(\neg(x \approx c_1) \vee ... \vee \neg(x \approx c_n))$ from Example 2.42 is equivalent to $F' = \top$ w.r.t. $\mathcal{T}_{\mathcal{IS}}$.

In the following we define what a model completion is, then we show that the model completion $\mathcal{T}_{\mathcal{IS}}$ of $\mathcal{E}$ admits quantifier elimination.

Definition 2.43 (Diagram of a Structure [61]). *The* diagram $\Delta(\mathcal{A})$ *of a* Π*-structure* $\mathcal{A}$ *is the set of ground* $\Pi^{|\mathcal{A}|}$*-literals that are true in* $\mathcal{A}$ *(where* $\Pi^{|\mathcal{A}|}$ *is the signature obtained from* Π *by adding to it new constants naming the elements of* $|\mathcal{A}|$*).*

Definition 2.44 (Complete Theory [61]). *A theory* $\mathcal{T}$ *is called* complete *if it has models and every two models of* $\mathcal{T}$ *are elementary equivalent, i.e. satisfy the same first-order formulae (this is the same as saying that for every formula* F *in the language of* $\mathcal{T}$ *either* F *or* $\neg F$ *is a consequence of* $\mathcal{T}$*).*

Definition 2.45 (Embedding [61]). *Let* $\Pi = \{\Sigma, \mathsf{Pred}\}$ *and let* $\mathcal{A}$ *and* $\mathcal{B}$ *be* Π*-structures. A map* $h : |\mathcal{A}| \to |\mathcal{B}|$ *is called a* Π-embedding *if and only if it is an injective homomorphism and for all* $P \in \mathsf{Pred}$ *with arity* n *and all* $(a_1, ..., a_n) \in |\mathcal{A}|^n$ *the following holds:*

$$(a_1, ..., a_n) \in P_{\mathcal{A}} \text{ if and only if } (h(a_1), ..., h(a_n)) \in P_{\mathcal{B}}.$$

Definition 2.46 (Elementary Embedding [61]). *A* Π*-embedding that preserves the truth of all first-order formulae over* Π *is called an* elementary embedding.

Definition 2.47 (Model-complete [61]). *A theory is called* model-complete *if every embedding between its models is elementary.*

Definition 2.48 (Model Companion [61]). *Let* $\mathcal{T}$ *be a* Π*-theory. We call* $\mathcal{T}^*$ *a* model companion *of* $\mathcal{T}$ *if and only if*

(1) $\mathcal{T}^*$ *is model-complete, and*

(2) every model of $\mathcal{T}$ *can be extended to a model of* $\mathcal{T}^*$ *and vice versa.*

Definition 2.49 (Model Completion [61]). *Let* $\mathcal{T}$ *be a* Π*-theory and let* $\mathcal{T}^*$ *with* $\mathcal{T} \subseteq \mathcal{T}^*$ *be a further* Π*-theory. We say that* $\mathcal{T}^*$ *is a* model completion *of* $\mathcal{T}$ *if and only if*

(1) $\mathcal{T}^*$ *is a model companion of* $\mathcal{T}$*, and*

(2) for every model $\mathcal{A}$ *of* $\mathcal{T}$ *we have that* $\mathcal{T}^* \cup \Delta(\mathcal{A})$ *is a complete theory.*

There is a relation between model completions and quantifier elimination, as shown in the following theorem.

Theorem 2.50 ([51]). *Let* $\mathcal{T}$ *be a* Π*-theory and let* $\mathcal{T}^*$ *with* $\mathcal{T} \subseteq \mathcal{T}^*$ *be a further* Π*-theory.* $\mathcal{T}^*$ *is a model completion of* $\mathcal{T}$ *if*

(1) every model of $\mathcal{T}$ *can be embedded into a model of* $\mathcal{T}^*$*, and*

(2) $\mathcal{T}^*$ *admits quantifier elimination.*

$\mathcal{T}_{\mathcal{IS}}$ is the model completion of $\mathcal{E}$ and it admits quantifier elimination. We now analyze how quantifiers can be eliminated in the theory $\mathcal{T}_{\mathcal{IS}}$.

Let $F(x, \overline{y})$ be a quantifier-free formula in negation normal form. We only consider the case in which the existential quantifier in $\exists x F(x, \overline{y})$ needs to be eliminated (if we have a universal quantifier we can use the equivalence $\forall x F(x, \overline{y}) \equiv \neg\exists x \neg F(x, \overline{y})$). By extending our signature with $\not\approx$ we can assume that F does not contain any negation (as the equivalences $\neg(c \approx d) \equiv c \not\approx d$ and $\neg(c \not\approx d) \equiv c \approx d$ hold). Therefore, F contains only atoms of the form $c \approx d$ and $c \not\approx d$, where c and d are constants or variables.

We first present a simple method for quantifier elimination in $\mathcal{T}_{\mathcal{IS}}$ which is typically used, afterwards we show that virtual substitution can be used alternatively.

The Typical Method for Quantifier Elimination

In what follows, we denote literals of the form $\neg(c \approx d)$ with $c \not\approx d$. Let $\overline{y} = y_1, ..., y_k$ and $F(x, \overline{y})$ be a formula containing only atoms of the form $x \rhd x$, $x \rhd y_j$, $y_j \rhd x$ and $y_{j_1} \rhd y_{j_2}$, where $\rhd \in \{\approx, \not\approx\}$ and $j, j_1, j_2 \in \{1, ..., k\}$. We can eliminate x in $\exists x F(x, \overline{y})$ by applying the following steps:

(1) We bring the formula F to disjunctive normal form and distribute the existential quantifier over the disjuncts: $\exists x F(x, \overline{y}) \equiv \bigvee_i \exists x C_i(x, \overline{y})$, where C_i are conjunctions of atoms of the form $c \approx d$ or $c \not\approx d$ with $c, d \in \{x, y_1, ..., y_k\}$.

(2) For each formula $\exists x C_i(x, \overline{y})$ we compute an equivalent quantifier-free formula $C_i'(\overline{y})$. If C_i contains an atom of the form $x \not\approx x$, then $C_i'(\overline{y}) = \bot$. Otherwise we do the following:

- We replace in C_i every atom of the form $x \approx x$ with $\top$.
- If C_i contains at least one literal of the form $x \approx y_j$ or $y_j \approx x$ with $j \in \{1, ..., k\}$, then we substitute every occurrence of x in C_i with y_j. The result is an equivalent quantifier-free formula $C_i'(\overline{y})$.
- If in C_i all atoms containing x are of the form $x \not\approx y_j$ or $y_j \not\approx x$ with $j \in \{1, ..., k\}$, then we replace every such atom with $\top$. The result is an equivalent quantifier-free formula $C_i'(\overline{y})$.

(3) The formula $F'(\overline{y}) = \bigvee_i C_i'(\overline{y})$ is a quantifier-free formula that is equivalent to $\exists x F(x, \overline{y})$ w.r.t. $\mathcal{T}_{\mathcal{IS}}$.

Note that in the last case of Step 2 every literal of the form $x \not\approx y_i$ can be replaced by $\top$ because we can only have a finite number of atoms but we have an infinite universe, so it is guaranteed that there exists an element different from every y_i such that all atoms $x \not\approx y_i$ become true.

We now illustrate the application of this quantifier elimination procedure on an example.

Example 2.51. Let $F(x, \overline{y}) = (x \approx y_1 \wedge x \approx y_2) \vee (x \not\approx y_3 \wedge y_1 \approx y_3)$. We eliminate the existential quantifier in $\exists x F(x, \overline{y})$ by following the steps above:

(1) The formula $F(x, \overline{y})$ is already in disjunctive normal form, so

$$\begin{aligned} \exists x F(x, \overline{y}) &\equiv \exists x C_1(x, \overline{y}) \vee \exists x C_2(x, \overline{y}), \text{ where} \\ C_1(x, \overline{y}) &= (x \approx y_1 \wedge x \approx y_2) \text{ and} \\ C_2(x, \overline{y}) &= (x \not\approx y_3 \wedge y_1 \approx y_3). \end{aligned}$$

(2) Neither F_1 nor F_2 contain atoms of the form $x \not\approx x$ or $x \approx x$. Since in C_1 we have the atom $x \approx y_1$, we can replace every occurrence of x in F_1 with y_1. We obtain the following:

$$\begin{aligned} \exists x C_1(x, \overline{y}) &\equiv (y_1 \approx y_1 \wedge y_1 \approx y_2) \\ &\equiv (\top \wedge y_1 \approx y_2) \\ &\equiv y_1 \approx y_2 = C_1'(\overline{y}) \end{aligned}$$

Since in C_2 the atom $x \not\approx y_1$ is the only atom containing x, we can replace it by $\top$. We obtain the following:

$$\begin{aligned} \exists x C_2(x, \overline{y}) &\equiv (\top \wedge y_1 \approx y_3) \\ &\equiv y_1 \approx y_3 = C_2'(\overline{y}) \end{aligned}$$

(3) The quantifier-free formula $F'(\overline{y}) = C_1'(\overline{y}) \vee C_2'(\overline{y}) = (y_1 \approx y_2) \vee (y_1 \approx y_3)$ is equivalent to $\exists x F(x, \overline{y}) = \exists x (x \approx y_1 \wedge x \approx y_2) \vee (x \not\approx y_3 \wedge y_1 \approx y_3)$ w.r.t. $\mathcal{T}_{\mathcal{IS}}$. ■

Quantifier Elimination Using Virtual Substitution

We can also eliminate x in $\exists x F(x, \overline{y})$ using the virtual substitution method. Without loss of generality we can assume that all atoms in $F(x, \overline{y})$ which contain x are of the form $x \approx d$ where $d \in \{x, \overline{y}\}$, i.e. x always occurs on the left hand side of the equality (or on both sides). In the following we describe what happens in Steps 1-5 of Algorithm 1 (virtual substitution) when being applied on formulae in the theory $\mathcal{T}_{\mathcal{IS}}$.

- In Step 1 we transform every literal of the form $\neg(x \approx y)$ to $x \not\approx y$ (for this we extend the signature by the predicate symbol $\not\approx$).
- In Step 2 we compute the solution candidates. As in the atoms $x \approx y$ and $x \not\approx y$ the variables x and y are always defined, we do not need guards.
- In Step 3 we add $-\infty$ as a test point and in Step 4 we replace every occurrence of x with $-\infty$. Note that this test point would not be necessary in this case, as it is only needed for satisfying atoms of the form $x \leq t$.
- For each atom $x \approx y$ we add in Step 3 y as a test point and replace in Step 4 every occurrence of x with y.
- For each atom $x \not\approx y$ we add in Step 3 $y + \varepsilon$ as a test point, do the substitution in Step 4 and simplify in Step 5 the formula by replacing every atom $y + \varepsilon \approx z$ with $\bot$ and every atom $y + \varepsilon \not\approx z$ with $\top$.

For comparison with the first quantifier elimination method, we now apply virtual substitution on the same example.

Example 2.52. Let $F(x, \overline{y}) = (x \approx y_1 \wedge x \approx y_2) \vee (\neg(x \approx y_3) \wedge y_1 \approx y_3)$. We want to eliminate x in $\exists x F(x, \overline{y})$. We follow the steps described above:

(1) We transform the non-atomic formula $\neg(x \approx y_3)$ to $x \not\approx y_3$ and obtain the following set of atomic formulae: $A = \{x \approx y_1, x \approx y_2, x \not\approx y_3, y_1 \approx y_3\}$

(2) We do not need guards for the solution candidates, so our set of solution candidates looks as follows: $C = \{x \approx y_1, x \approx y_2, x \not\approx y_3\}$

(3) The elimination set E contains $-\infty$ and the test points derived from the solution candidates: $E = \{-\infty, y_1, y_2, y_3 + \varepsilon\}$

(4) We virtually substitute the terms from E in F and use the simplification rules:

$$
\begin{aligned}
\exists x F(x, \overline{y}) \equiv\ & ((-\infty \approx y_1 \wedge -\infty \approx y_2) \vee (-\infty \not\approx y_3 \wedge y_1 \approx y_3)) \vee \\
& ((y_1 \approx y_1 \wedge y_1 \approx y_2) \vee (y_1 \not\approx y_3 \wedge y_1 \approx y_3)) \vee \\
& ((y_2 \approx y_1 \wedge y_2 \approx y_2) \vee (y_2 \not\approx y_3 \wedge y_1 \approx y_3)) \vee \\
& ((y_3 + \varepsilon \approx y_1 \wedge y_3 + \varepsilon \approx y_2) \vee (y_3 + \varepsilon \not\approx y_3 \wedge y_1 \approx y_3)) \\
\equiv\ & ((\bot \wedge \bot) \vee (\top \wedge \bot)) \vee\ ((\top \wedge y_1 \approx y_2) \vee \bot) \vee \\
& ((y_2 \approx y_1 \wedge \top) \vee (y_2 \not\approx y_3 \wedge y_1 \approx y_3)) \vee \\
& ((\bot \wedge \bot) \vee (\top \wedge y_1 \approx y_3)) \\
\equiv\ & (y_1 \approx y_2) \vee (y_2 \approx y_1) \vee (y_2 \not\approx y_3 \wedge y_1 \approx y_3) \vee (y_1 \approx y_3) \\
\equiv\ & (y_1 \approx y_2) \vee (y_1 \approx y_3)
\end{aligned}
$$

Thus, the quantifier-free formula $F'(\overline{y}) = (y_1 \approx y_2) \vee (y_1 \approx y_3)$ is equivalent w.r.t. $\mathcal{T}_{\mathcal{IS}}$ to $\exists x F(x, \overline{y}) = \exists x (x \approx y_1 \wedge x \approx y_2) \vee (\neg(x \approx y_3) \wedge y_1 \approx y_3)$. ■

As one can see, we obtained the same result with the two presented methods for quantifier elimination in $\mathcal{T}_{\mathcal{IS}}$. We now give an analysis of why virtual substitution yields a correct result when being applied for this theory.

Since virtual substitution yields an equivalent quantifier-free formula F' for an arbitrary formula F, it is sufficient to show that it computes the right result for a formula in disjunctive normal form that is equivalent to F. For this we show that applying virtual substitution to a formula in disjunctive normal form always yields the same result as the other method.

Let $\overline{y} = y_1, ..., y_k$ and $F(x, \overline{y}) = C_1(x, \overline{y}) \vee ... \vee C_n(x, \overline{y})$ be a formula in disjunctive normal form with C_1,...,C_n being conjunctions of atoms of the form $x \rhd x$, $x \rhd y_j$ and $y_{j_1} \rhd y_{j_2}$, where $\rhd \in \{\approx, \not\approx\}$ and $j, j_1, j_2 \in \{1, ..., k\}$. Let x be the existentially quantified variable to be eliminated. We compare the way virtual substitution works with Step 2 of the other elimination method.

- Note first that, with the exception of $-\infty$, we can only have two kinds of test points: atoms of the form $x \approx y$ yield y as a test point, while atoms of the form $x \not\approx y$ yield $y + \varepsilon$ as a test point.
- If an atom of the form $x \approx x$ is present in a conjunction C_i and the variable x is replaced by some test point t, then this yields $t \approx t$ and simplifies to $\top$. This is the same as in the other method.
- If an atom of the form $x \not\approx x$ is present in a conjunction C_i and x is replaced by a test point t, then this yields $t \not\approx t$, which simplifies to $\bot$ and makes the whole conjunction evaluate to $\bot$. This is the same as in the other method.
- Consider a conjunction C_i in which an atom of the form $x \approx y$ occurs. It yields y as a test point, so in the result of the virtual substitution there will be a conjunction C_i' resulting from replacing every occurrence of x in C_i by y. The same conjunction appears in the result when we use the other method.
- Consider a conjunction C_i in which only atoms of the form $x \not\approx y$ occur. It yields terms of the form $y + \varepsilon$ as test points. Replacing x with such a test point yields atoms of the form $y + \varepsilon \not\approx z$, which are then simplified to $\top$. Therefore, we obtain a conjunction C_i' where all atoms of the form $x \not\approx y$ are replaced by $\top$. The same conjunction appears in the result when we use the other method.

Let F_1' be the result obtained by applying the typical method for quantifier elimination and F_2' be the result obtained by applying the method of virtual substution. The considerations above show that every conjunction C_i' that occurs in F_1' also occurs in F_2'. However, without additional simplifications F_2' will in addition contain many conjunctions that are not present in F_1' for the following reasons:

(1) We always have $-\infty$ as an additional test point for virtual substitution.

(2) From one conjunction C_i we may get several test points for virtual substitution. For instance, if we have the conjunction $C_i = x \approx y_1 \wedge x \approx y_2$, this yields two test points y_1 and y_2 for virtual substitution, while in the typical method x only has to be replaced with one of the two.

(3) In virtual substitution we always replace x with a test point t in the whole formula, while in the typical method we replace x only in the relevant conjunction.

We show that the additional conjunctions arising from (1) to (3) are not relevant for the formula.

Consider (1):

In conjunctions containing $-\infty$ all atoms of the form $-\infty \approx y$ are simplified to $\bot$ and all atoms of the form $-\infty \not\approx y$ are simplified to $\top$. Therefore, if a conjunction C_i' contains at least one atom of the form $-\infty \approx y$, then C_i' is false and therefore not relevant for the disjunction. If C_i' contains only atoms of the form $-\infty \not\approx y$, then each of these atoms is replaced by $\top$, which yields the same conjunction that is obtained by replacing x with the test point $y + \varepsilon$, so it adds nothing new to the formula.

Consider (2):

If all atoms in C_i containing x are of the form $x \not\approx y$, then for each test point we obtain the same conjunction C_i', in which all such atoms are replaced by $\top$. Assume now that C_i contains also atoms of the form $x \approx y$, i.e. it is of the form

$$C_i = x \approx y_{i_1} \wedge ... \wedge x \approx y_{i_n} \wedge x \not\approx y_{j_1} \wedge ... \wedge x \not\approx y_{j_m} \wedge \overline{C}(\overline{y})$$

where $n, m \geq 0$ and $\overline{C}(\overline{y})$ is a conjunction of atoms not containing x.

We obtain $n + m$ test points $y_{i_1}, ..., y_{i_n}, y_{j_1} + \varepsilon, ..., y_{j_m} + \varepsilon$ and therefore after substitution the following $n + m$ conjunctions:

$$\begin{aligned}
C'_{i_1} &= y_{i_1} \approx y_{i_1} \wedge ... \wedge y_{i_1} \approx y_{i_n} \wedge y_{i_1} \not\approx y_{j_1} \wedge ... \wedge y_{i_1} \not\approx y_{j_m} \wedge \overline{C}(\overline{y}) \\
&\vdots \\
C'_{i_n} &= y_{i_n} \approx y_{i_1} \wedge ... \wedge y_{i_n} \approx y_{i_n} \wedge y_{i_n} \not\approx y_{j_1} \wedge ... \wedge y_{i_n} \not\approx y_{j_m} \wedge \overline{C}(\overline{y}) \\
C'_{j_1} &= y_{j_1} + \varepsilon \approx y_{i_1} \wedge ... \wedge y_{j_1} + \varepsilon \approx y_{i_n} \wedge y_{j_1} + \varepsilon \not\approx y_{j_1} \wedge ... \wedge y_{j_1} + \varepsilon \not\approx y_{j_m} \wedge \overline{C}(\overline{y}) \\
&\vdots \\
C'_{j_m} &= y_{j_m} + \varepsilon \approx y_{i_1} \wedge ... \wedge y_{j_m} + \varepsilon \approx y_{i_n} \wedge y_{j_m} + \varepsilon \not\approx y_{j_1} \wedge ... \wedge y_{j_m} + \varepsilon \not\approx y_{j_m} \wedge \overline{C}(\overline{y})
\end{aligned}$$

The conjunctions $C'_{i_1}, ..., C'_{i_n}$ are equivalent, since for each of them the conjunction of the first n atoms is equivalent to $y_{i_1} \approx y_{i_2} \approx ... \approx y_{i_n}$. The conjunctions $C'_{j_1}, ..., C'_{j_m}$ evaluate to $\bot$, as they contain atoms of the form $y_1 + \varepsilon \not\approx y_2$, which are simplified to $\bot$ according to the simplication rules for virtual substitution. Thus, replacing x with more than one test point in a conjunction C_i does not add any relevant conjunctions.

Consider (3):

Let C_1 and C_2 be two conjunctions in F of the form

$$\begin{aligned}
C_1 &= x \approx y_{i_1^1} \wedge ... \wedge x \approx y_{i_n^1} \wedge x \not\approx y_{j_1^1} \wedge ... \wedge x \not\approx y_{j_m^1} \wedge \overline{C}_1(\overline{y}) \text{ and} \\
C_2 &= x \approx y_{i_1^2} \wedge ... \wedge x \approx y_{i_k^2} \wedge x \not\approx y_{j_1^2} \wedge ... \wedge x \not\approx y_{j_l^2} \wedge \overline{C}_2(\overline{y}),
\end{aligned}$$

where $n, m, k, l \geq 0$ and $\overline{C}_1(\overline{y})$ and $\overline{C}_2(\overline{y})$ are conjunctions of atoms not containing x. We show that substituting x in C_2 with test points from C_1 yields only conjunctions which are not relevant for the formula.

From C_1 we derive the set of test points $\{y_{i_1^1}, ..., y_{i_n^1}, y_{j_1^1} + \varepsilon, y_{j_m^1} + \varepsilon\}$. We make a case distinction on the form of the test point we are substituting with.

Case 1: We substitute x in C_2 with the test points $y_{i_1^1}, ..., y_{i_n^1}$. We obtain the following n conjunctions:

$$C'_{2i_1^1} = y_{i_1^1} \approx y_{i_1^2} \wedge ... \wedge y_{i_1^1} \approx y_{i_k^2} \wedge y_{i_1^1} \not\approx y_{j_1^2} \wedge ... \wedge y_{i_1^1} \not\approx y_{j_l^2} \wedge \overline{C}_2(\overline{y})$$
$$\vdots$$
$$C'_{2i_n^1} = y_{i_n^1} \approx y_{i_1^2} \wedge ... \wedge y_{i_n^1} \approx y_{i_k^2} \wedge y_{i_n^1} \not\approx y_{j_1^2} \wedge ... \wedge y_{i_n^1} \not\approx y_{j_l^2} \wedge \overline{C}_2(\overline{y})$$

Note that from the substitution of x with $y_{i_1^2}$ (one of the test points obtained from C_2) in C_2 we have the following conjunction:

$$C'_{2i_1^2} = y_{i_1^2} \approx y_{i_1^2} \wedge ... \wedge y_{i_1^2} \approx y_{i_k^2} \wedge y_{i_1^2} \not\approx y_{j_1^2} \wedge ... \wedge y_{i_1^2} \not\approx y_{j_l^2} \wedge \overline{C}_2(\overline{y})$$

It is easy to see that the following hold:

$$C'_{2i_1^1} \equiv \overline{C}_2(\overline{y}) \wedge y_{i_1^2} \not\approx y_{j_1^2} \wedge ... \wedge y_{i_1^2} \not\approx y_{j_l^2} \wedge y_{i_1^2} \approx y_{i_2^2} \wedge ... \wedge y_{i_1^2} \approx y_{i_k^2} \wedge y_{i_1^2} \approx y_{i_1^1}$$
$$C'_{2i_1^2} \equiv \overline{C}_2(\overline{y}) \wedge y_{i_1^2} \not\approx y_{j_1^2} \wedge ... \wedge y_{i_1^2} \not\approx y_{j_l^2} \wedge y_{i_1^2} \approx y_{i_2^2} \wedge ... \wedge y_{i_1^2} \approx y_{i_k^2}$$

It follows that $C'_{2i_1^1} \models C'_{2i_1^2}$ and thus the conjunction $C'_{2i_1^1}$ is not needed in the disjunction resulting from virtual substitution. Similarly, the same can be shown for conjunctions $C'_{2i_2^1}, ..., C'_{2i_n^1}$.

Case 2: We substitute x in C_2 with the test points $y_{j_1^1} + \varepsilon, ..., y_{j_m^1} + \varepsilon$. We obtain the following m conjunctions:

$$C'_{2j_1^1} = y_{j_1^1} + \varepsilon \approx y_{i_1^2} \wedge ... \wedge y_{j_1^1} + \varepsilon \approx y_{i_k^2} \wedge y_{j_1^1} + \varepsilon \not\approx y_{j_1^2} \wedge ... \wedge y_{j_1^1} + \varepsilon \not\approx y_{j_l^2} \wedge \overline{C}_2(\overline{y})$$
$$\vdots$$
$$C'_{2j_m^1} = y_{j_m^1} + \varepsilon \approx y_{i_1^2} \wedge ... \wedge y_{j_m^1} + \varepsilon \approx y_{i_k^2} \wedge y_{j_m^1} + \varepsilon \not\approx y_{j_1^2} \wedge ... \wedge y_{j_m^1} + \varepsilon \not\approx y_{j_l^2} \wedge \overline{C}_2(\overline{y})$$

According to the simplification rules of virtual substitution, all atoms of the form $y+\varepsilon \approx z$ are replaced by $\bot$ and all atoms of the form $y+\varepsilon \not\approx z$ are replaced by $\top$. Therefore, if $k \geq 1$, then the conjunctions $C'_{2j_1^1}, ..., C'_{2j_m^1}$ are equivalent to $\bot$. If $k = 0$, then all conjunctions are equivalent to $\overline{C}_2(\overline{y})$, which is the same conjunction one obtains by replacing x with the test point $y_{j_1^2} + \varepsilon$ in C_2, so it does not add anything new to the formula.

In conclusion, we have shown that using virtual substitution on a formula in disjunctive normal form yields a disjunction of conjunctions which includes all the conjunctions that are obtained with the typical method and many additional conjunctions which are not relevant for the formula. Therefore, for a formula F in disjunctive normal form, the formula F'_2 obtained by using virtual substitution is equivalent to the formula F'_1 obtained by using the typical method. It follows that also for arbitrary input formulae the results of the two methods will always be equivalent.

The fact that virtual substitution can be used for quantifier elimination in the theory of an infinite set will be used in Section 3.1 to show that virtual substitution can be used for quantifier elimination in the combination of the theory of real closed fields and the theory of an infinite set as well.

2.4 Provers

In this section we introduce the software that was used for the work presented in the thesis. This includes:

- the logic system **Redlog**, which is used for quantifier elimination in real closed fields or Presburger arithmetic.
- the SMT solver **Z3**, which is used for satisfiability checking and unsatisfiable core computation.
- the theorem prover **SPASS**, which is used for applying resolution on propositional formulae.
- the program **H-PILoT**, which is used for hierarchical reduction and reasoning in theory extensions.
- the program **SEH-PILoT**, which is used for property-directed symbol elimination and invariant strengthening.

We here give a brief description of the provers and what they are used for in this thesis. We provide references for readers interested in more details about the provers.

Redlog

Redlog (short for Reduce logic system) is an extension of the computer algebra system Reduce to a computer logic system, where several algorithms can be applied to first-order formulae, including several forms of quantifier elimination [41]. Redlog needs a context specifying the language and theory which is used. Since in our applications we usually have real numbers, we need a context specifying the real closed fields, which Redlog offers in the form of a context called OFSF (which stands for ordered fields standard form). Sometimes we may also use the context PASF, which stands for Presburger arithmetic standard form and is used for linear integer arithmetic. Given a context and an input formula with quantifiers, we can use Redlog to eliminate quantifiers and obtain an equivalent formula not containing any of the quantified variables. For the context OFSF Redlog has the virtual substitution method described in Section 2.3.1 implemented for the linear and quadratic case, for formulae of higher degrees Redlog uses heuristics [112, 105]. In this thesis, Redlog is the main software used for any kind of quantifier elimination over the real numbers or the integers.

Z3

Z3 is a theorem prover developed by Microsoft Research [79]. It is used to check the satisfiability of logical formulae over one or more theories. Its input syntax is based on the SMT-LIB 2.0 standard. If Z3 detects satisfiability of a formula, it can also generate a model. If Z3 detects unsatisfiability of a formula, then it can also display a proof. However, these proofs are often not very comprehensible for a human and therefore this option is not used in the thesis. If an unsatisfiable set of formulae is given in Z3, then it also offers the option to compute a (minimal) unsatisfiable core. In this thesis, Z3 is used for satisfiability checking, model generation and unsatisfiable core computation.

SPASS

SPASS is a theorem prover for first-order logic with equality that is developed at the Max Planck Institute for Informatics in Saarbrücken, Germany [110]. It tries to determine whether a formula is valid or not by negating the formula, adding it to the premises and applying reduction and inference rules based on the superposition calculus. For valid formulae a proof can be given. For formulae which are not valid a saturated set of clauses is returned, from which a model can be built (i.e. a countereaxample). SPASS offers many options on which kind of inference and simplification rules are used. It is in particular possible to restrict to resolution and factorization rules. It is also possible to set a precedence on the used symbols, which determines an order for using ordered resolution. In this thesis, we use SPASS mainly for resolution.

H-PILoT

H-PILoT (short for Hierarchical Proving by Instantiation in Local Theory extensions) is a program for performing hierarchical reasoning in theory extensions, developed in the research group of Prof. Dr. Viorica Sofronie-Stokkermans [65]. It reduces a problem in the theory extension (this can also be a chain of theory extensions) to a problem in the base theory. Specialized provers or SMT solvers, such as Redlog or Z3, are then used to test satisfiability of the formula obtained after reduction. If the answer is unsatisfiable, then one knows that the original problem is unsatisfiable as well. If the answer is satisfiable, then satisfiability of the original problem is only guaranteed if all theory extensions are local. In this thesis, we use H-PILoT for hierarchical reasoning and satisfiability checking in theory extensions which are known to be local.

SEH-PILoT

SEH-PILoT (short for Symbol Elimination based on Hierarchical Proving In Local Theory Extensions) is an extension of H-PILoT that was written by Philipp Marohn as part of his Bachelor thesis [77] and extended with more features later on. Its main purpose is to perform property-directed symbol elimination in theory extensions (see Section 3.2.2). The program builds upon the hierarchical reduction of H-PILoT with Redlog as a prover, but allows the user to specify which symbols need to be eliminated and after the elimination does the resubstitution of the constants that were introduced by H-PILoT in the reduction process. Additionally, SEH-PILoT offers functionality for invariant strengthening in parametric systems (see Section 4.4). Moreover, it has various (optional) simplification techniques implemented to make the formulae obtained by Redlog as short and comprehensible as possible. In this thesis, SEH-PILoT is used for performing property-directed symbol elimination in local theory extensions and for invariant strengthening in parametric systems.

3 Symbol Elimination

In Section 2.3 we discussed possibilities of eliminating quantified variables from a formula in order to obtain an equivalent quantifier-free formula. However, sometimes we may be interested in eliminating not only quantified variables, but other kinds of symbols as well, e.g. constants, function symbols or predicate symbols. In this case we use the term *symbol elimination*. We distinguish between two variants of symbol elimination. The goal of *general symbol elimination* is to compute from a given formula F a formula F' that is equivalent to F and does not contain certain symbols. The goal of *property-directed symbol elimination* is different. We are given a formula F and an additional formula G and want to compute a formula F' not containing certain symbols that has some desired property related to G. Symbol elimination is a more general notion than quantifier elimination, as quantifier elimination is a special kind of general symbol elimination.

The theories allowing quantifier elimination that were discussed in Section 2.3 were simple theories, like the theory of real closed fields or the theory of an infinite set. However, in many cases we have more complex theories, for example a combination of different theories or extensions of a base theory. Therefore, it is important to have methods for quantifier elimination or, more generally, symbol elimination in theories with such complex structures. We first analyze possibilities of quantifier elimination in combinations of disjoint theories with a concrete example relevant for this thesis. Afterwards we present an algorithm for symbol elimination in theory extensions, discuss the role of locality for this method, present an improvement of the original algorithm and illustrate the algorithm and its improvement on examples.

Parts of the results in this chapter were already published in [81, 82].

3.1 Quantifier Elimination in Combinations of Theories

In Section 2.3 we discussed methods for quantifier elimination in simple theories, but sometimes the problems we consider are structured in a more complex way, for instance as a combination of theories with disjoint signatures. Therefore, we now analyze possibilities of existential quantifier elimination in combinations of disjoint theories.

Theorem 3.1 (Existential quantifier elimination in combinations of disjoint theories). *Let S_1 and S_2 be sets of sorts such that $S_1 \cap S_2 = \emptyset$. Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be theories over the disjoint signatures $\Pi_1 = (S_1, \Sigma_1, \mathsf{Pred}_1)$ and $\Pi_2 = (S_2, \Sigma_2, \mathsf{Pred}_2)$, respectively. Let $\mathcal{T}$ be the two-sorted combination of the theories $\mathcal{T}_1$ and $\mathcal{T}_2$ with signature $\Pi = (S_1 \cup S_2, \Sigma_1 \cup \Sigma_2, \mathsf{Pred}_1 \cup \mathsf{Pred}_2)$, where every n-ary operation $f \in \Sigma_i$ has sort $s_1...s_n \to s$ with $s_1, ..., s_n, s \in S_i$, and every m-ary predicate symbol $p \in \mathsf{Pred}_i$ has arity $s_1...s_m$ with $s_1, ..., s_m \in S_i$. Assume that $\mathcal{T}_1$ and $\mathcal{T}_2$ allow elimination of existential quantifiers. Let $F(x, \overline{y})$ be a quantifier-free Π-formula. Then $\exists x F(x, \overline{y})$ is equivalent w.r.t. $\mathcal{T}$ with a quantifier-free Π-formula $G(\overline{y})$.*

Proof: For eliminating the existential variable x from $\exists x F(x, \overline{y})$ we first bring F to disjunctive normal form, $F(x, \overline{y}) \equiv \bigvee_{i=1}^{n} D_i(x, \overline{y})$, where every conjunction D_i can be written as a conjunction $(D_i^1 \wedge D_i^2)$, where D_i^1 contains only atoms over the signature Π_1 and D_i^2 contains only atoms over the signature Π_2. Then $\exists x F(x, \overline{y}) \equiv \bigvee_{i=1}^{n} \exists x (D_i^1(x, \overline{y}) \wedge D_i^2(x, \overline{y}))$.

- Case 1: The variable x is of sort $s \in S_1$. Then for every i, x does not occur in D_i^2 and $\exists x (D_i^1(x, \overline{y}) \wedge D_i^2(\overline{y})) \equiv \exists x (D_i^1(x, \overline{y})) \wedge D_i^2(\overline{y})$. A method for quantifier elimination in $\mathcal{T}_1$ can be used for computing formulae $\overline{D}_i(\overline{y})$ with $\exists x (D_i^1(x, \overline{y})) \equiv \overline{D}_i(\overline{y})$.
- Case 2: The variable x is of sort $s \in S_2$. Then for every i, x does not occur in D_i^1 and $\exists x (D_i^1(\overline{y}) \wedge D_i^2(x, \overline{y})) \equiv D_i^1(\overline{y}) \wedge \exists x (D_i^2(x, \overline{y}))$. A method for quantifier elimination in $\mathcal{T}_2$ can be used for computing formulae $\overline{D}_i(\overline{y})$ with $\exists x (D_i^2(x, \overline{y})) \equiv \overline{D}_i(\overline{y})$. □

We now consider a special case of a disjoint combination of theories relevant for this thesis and analyze how to eliminate quantifiers using only one method for quantifier elimination.

Lemma 3.2. *Let $\mathcal{T}_{\mathcal{IS}}$ be the theory of an infinite set over signature $\Pi_1 = (\{\mathsf{p}\}, \Sigma_1, \mathsf{Pred}_1)$ and $\mathcal{T}_{\mathcal{RCF}}$ be the theory of real closed fields over signature $\Pi_2 = (\{\mathsf{num}\}, \Sigma_2, \mathsf{Pred}_2)$, where Π_1 and Π_2 are disjoint. Let $\mathcal{T} = \mathcal{T}_{\mathcal{IS}} \cup \mathcal{T}_{\mathcal{RCF}}$ be the disjoint combination the two theories with signature $\Pi = (\{\mathsf{p}, \mathsf{num}\}, \Sigma_1 \cup \Sigma_2, \mathsf{Pred}_1 \cup \mathsf{Pred}_2)$. Let $F(x, \overline{y})$ be a quantifier-free Π-formula. Then we can compute a Π-formula equivalent w.r.t. $\mathcal{T}$ to $\exists x F(x, \overline{y})$ using the virtual substitution method (Algorithm 1).*

Proof: Let $\exists x F(x, \overline{a})$ be the formula from which we want to eliminate x. Since we are in the disjoint combination of the two theories, each atom contains either only symbols from Π_1 or from Π_2. Note that all Π_1-atoms are of the form $p \approx q$, where p and q are variables in Σ_1. We consider two cases:

- Case 1: x is of sort num. Then x only occurs in Π_2-atoms. Then clearly by Algorithm 1 a quantifier-free formula F' equivalent to $\exists x F(x, \overline{a})$ w.r.t. $\mathcal{T}_{\mathcal{RCF}}$ is computed. Since the parts of the formula which do not contain x are not important for computing the test points, the existence of Π_1-atoms in $F(x, \overline{a})$ does not influence the equivalence between the original formula and the result of the virtual substitution, so F' is also equivalent to $\exists x F(x, \overline{a})$ w.r.t. $\mathcal{T}_{\mathcal{IS}} \cup \mathcal{T}_{\mathcal{RCF}}$.
- Case 2: x is of sort p. Then x only occurs in Π_1-atoms. Let L be the set of all Π_1-literals. We analyze what happens when we apply Algorithm 1.

  Step 1: The set A of atomic formulae is computed by transforming every formula $\neg(p \approx q)$ (where p and q are of sort p) to $p \not\approx q$ (we assume w.l.o.g. that $\not\approx \in \mathsf{Pred}_1$).

  $$A = \{p \approx q \mid p \approx q \in L\} \cup \{p \not\approx q \mid \neg(p \approx q) \in L\}$$

  Step 2: The set C of solution candidates is computed by separating x on the left-hand side of each equation and disequation. We obtain atoms of the form $x \approx y$ and $x \not\approx y$, where x and y are of sort p.

  $$C = \{x \approx y \mid x \approx y \in A\} \cup \{x \not\approx y \mid x \not\approx y \in A\}$$

  Step 3: The elimination set E is computed, which contains $-\infty$ (which is in fact not needed because there are no atoms of the form $x < y$ or $x \leq y$), y for every atom of the form $x \approx y$ in C, and $y + \varepsilon$ for every atom of the form $x \not\approx y$ in C. For these types of atoms no guards are needed, therefore we can always set $\gamma = \top$.

  $$E = \{(\top, -\infty)\} \cup \{(\top, y) \mid x \approx y \in C\} \cup \{(\top, y + \varepsilon) \mid x \not\approx y \in C\}$$

Step 4: The virtual substitution of x with elements in E is performed, which leads to the following formula:

$$\bigvee_{(\top,t)\in E} \top \wedge F(x,\overline{a})[x//t] \;\equiv\; \bigvee_{(\top,t)\in E} F(x,\overline{a})[x//t]$$

In the virtual substitution $F(x,\overline{a})[x//t]$ the simplification rules in Definition 2.40 are applied and yield the following:

- For $t = -\infty$ and $t = y + \varepsilon$, atoms of the form $x \approx z$ in F are replaced by $\bot$.
- For $t = -\infty$ and $t = y + \varepsilon$, atoms of the form $x \not\approx z$ in F are replaced by $\top$.
- For $t = y$, atoms of the form $x \approx z$ and $x \not\approx z$ are replaced by $y \approx z$ and $y \not\approx z$, respectively.

The formula obtained this way is quantifier-free and equivalent to $\exists x F(x,\overline{a})$ w.r.t. $\mathcal{T}_{\mathcal{IS}}$. Since the parts of the formula which do not contain x are not important for computing the test points, the existence of Π_2-atoms in $F(x,\overline{a})$ does not influence the equivalence between the original formula and the result of the virtual substitution, so the obtained formula is also equivalent to $\exists x F(x,\overline{a})$ w.r.t. $\mathcal{T}_{\mathcal{IS}} \cup \mathcal{T}_{\mathcal{RCF}}$.

In both cases by applying Algorithm 1 we obtain a quantifier-free formula that is equivalent to $\exists x F(x,\overline{a})$ w.r.t. $\mathcal{T}_{\mathcal{IS}} \cup \mathcal{T}_{\mathcal{RCF}}$. □

The result in Lemma 3.2 is very useful, as it simplifies the choice of tools for quantifier elimination in the combination of $\mathcal{T}_{\mathcal{RCF}}$ and $\mathcal{T}_{\mathcal{IS}}$. There is no need to implement the method described in Theorem 3.1, we can instead just use virtual substitution.

Examples in which Lemma 3.2 is used, i.e. in which virtual substitution is used for quantifier elimination in the combination of $\mathcal{T}_{\mathcal{RCF}}$ and $\mathcal{T}_{\mathcal{IS}}$, are shown in Chapter 6.

3.2 Symbol Elimination in Theory Extensions

The reasoning tasks we consider in the thesis are usually stated in such a way that we have to check unsatisfiability of a given formula. If for instance we want to verify that a certain system is always safe, then we try to prove that it cannot happen that the safety condition for the system is violated, which corresponds to checking that a formula specifying such a violation is unsatisfiable.

The theories we consider are often complex and structured as an extension of a base theory or even as a chain of theory extensions, as in the following example.

Example 3.3. Let $\mathcal{T}_0 = \mathsf{LI}(\mathbb{R})$ be a base theory which is extended by a set of additional function symbols $\{\overline{f}\}$, where $\overline{f} = f_1, ..., f_n$. Let $\mathsf{Safe}(\overline{x},\overline{f})$ be a formula describing a safety condition for the system (depending on variables $\overline{x}$ and extension functions $\overline{f}$) and $\mathsf{Update}(\overline{x},\overline{x'},\overline{f},\overline{f'})$ be a formula describing an update of the system, where primed symbols denote the values of the corresponding variables and functions after an update of the system. To prove safety of the system, one has to show that it is impossible that the safety condition holds before an update, but does not hold anymore after the update. This can be achieved by proving that the formula $\mathsf{Safe}(\overline{x},\overline{f}) \wedge \mathsf{Update}(\overline{x},\overline{x'},\overline{f},\overline{f'}) \wedge \neg\mathsf{Safe}(\overline{x'},\overline{f'})$ is unsatisfiable. ■

If $\mathsf{Safe}(\overline{x}, \overline{f})$ in the example above is a universal formula and we have a chain of theory extensions $\mathcal{T}_0 \subseteq \mathcal{T}_0 \cup \mathsf{Safe}(\overline{x}, \overline{f}) \subseteq \mathcal{T}_0 \cup \mathsf{Safe}(\overline{x}, \overline{f}) \cup \mathsf{Update}(\overline{x}, \overline{x'}, \overline{f}, \overline{f'})$ and all quantified variables occur below an extension function, then by hierarchical reasoning we can reduce such a satisfiability problem in a theory extension to a proof task in the base theory and then check satisfiability of the reduced problem. If one obtains unsatisfiability of the reduced formula, it is clear that the original formula is unsatisfiable as well. If all the theory extensions in the chain above are local, then also satisfiability of the reduced formula implies that the original formula is satisifable. If we obtain satisfiability it means that we could not prove that the safety condition is preserved under updates, for example that the system is not safe. In this case it can be of interest to find conditions over a given subsignature such that unsatisfiability is guaranteed, for example finding conditions which guarantee the safety of a system. Since the problems we consider are often parametric in some way, we usually are interested in finding additional constraints on the parameters that guarantee unsatisfiabilty of the original formula.

We now present a method for symbol elimination in theory extensions which allows us to generate such constraints. We first give the idea: Let Σ_0 be a set of base functions and Σ_1 be a set of extension functions disjoint from Σ_0. Assume given a theory extension $\mathcal{T}_0 \subseteq \mathcal{T}_0 \cup \mathcal{K}$, a set of ground clauses G containing ground terms in a set T and a set Σ_p of "parameters" with $\Sigma_p \subseteq \Sigma_1$. The task is to compute a constraint Γ containining only symbols in Σ_p such that $\mathcal{T}_0 \cup \Gamma \cup \mathcal{K} \cup G \models \bot$. We start with the set $\mathcal{K}[T] \cup G$ obtained after instantiation and use purification as described in Theorem 2.25 by introducing new constants for the terms appearing in G. Among these newly introduced constants we identify which ones need to be eliminated and which ones not. The constants corresponding to a symbol in Σ_p or corresponding to an argument of a function in Σ_p are not supposed to be eliminated, all other constants will be eliminated. For eliminating these (existentially quantified) constants we can use a quantifier elimination method for $\mathcal{T}_0$. In the result we then have to replace back the constants introduced during the purification step with the terms they stand for and finally negate the obtained formula.

In the following we describe the details of the algorithm. Let $\Pi_0 = (\Sigma_0, \mathsf{Pred})$. Let $\mathcal{T}_0$ be a Π_0-theory and Σ_p be a set of parameters (function and constant symbols) with $\Sigma_p \cap \Sigma_0 = \emptyset$. Let Σ be a signature such that $\Sigma \cap (\Sigma_0 \cup \Sigma_p) = \emptyset$. We consider the theory extension $\mathcal{T}_0 \subseteq \mathcal{T}_0 \cup \mathcal{K}$, where $\mathcal{K}$ is a set of clauses in the signature $\Pi = \Pi_0 \cup \Sigma_p \cup \Sigma$ in which all variables occur also below functions in $\Sigma_1 = \Sigma_p \cup \Sigma$. In Algorithm 2 we describe a method for symbol elimination proposed in [98, 100]. For a given theory extension $\mathcal{T}_0 \subseteq \mathcal{T}_0 \cup \mathcal{K}$ and a set of flat and linear ground clauses G such that $\mathcal{T}_0 \cup \mathcal{K} \cup G$ is satisfiable, it computes a universal formula Γ over $\Pi_0 \cup \Sigma_p$ such that $\mathcal{T}_0 \cup \mathcal{K} \cup \Gamma \cup G$ is unsatisfiable. The following theorems state that the computed formula is always a constraint that guarantees unsatisfiability of the input formula in the extended theory (Theorem 3.4) and if we have theory extensions satisfying ($\mathsf{Comp_f}$) it is even the weakest such constraint (Theorem 3.5).

Theorem 3.4 ([98]). *Assume that the base theory $\mathcal{T}_0$ allows quantifier elimination. For every finite set of flat and linear ground Π^C-clauses G, and every finite set T of flat terms over the signature $\Pi_0 \cup \Sigma_p \cup \Sigma \cup \Sigma_C$ with $\mathsf{est}(G) \subseteq T$ and such that $\mathcal{K}[T]$ is ground, Algorithm 2 yields a universally quantified $\Pi_0 \cup \Sigma_p$-formula $\forall \overline{y} \Gamma_T(\overline{y})$ with the following properties:*

(1) For every structure $\mathcal{A}$ with signature $\Pi_0 \cup \Sigma \cup \Sigma_p \cup \Sigma_C$ which is a model of $\mathcal{T}_0 \cup \mathcal{K}$, if $\mathcal{A} \models \forall \overline{y} \Gamma_T(\overline{y})$, then $\mathcal{A} \models \neg G$.

(2) $\mathcal{T}_0 \cup \forall \overline{y} \Gamma_T(\overline{y}) \cup \mathcal{K} \cup G$ is unsatisfiable.

Algorithm 2 Symbol Elimination in Theory Extensions [98, 100]

Input: Theory extension $\mathcal{T}_0 \subseteq \mathcal{T}_0 \cup \mathcal{K}$ with signature $\Pi = \Pi_0 \cup (\Sigma_p \cup \Sigma)$ where Σ_p is a set of parameters and $\mathcal{K}$ a set of flat and linear clauses; set G of flat and linear ground Π^C-clauses; set T of flat ground Π^C-terms such that $\mathcal{K}[T]$ is ground.

Output: Universal $\Pi_0 \cup \Sigma_p$-formula $\forall \overline{y} \Gamma_T(\overline{y})$.

Step 1 Purify $\mathcal{K}[T] \cup G$ as described in Theorem 2.25 (with set of extension symbols $\Sigma_1 = \Sigma_p \cup \Sigma$). Let $\mathcal{K}_0 \cup G_0 \cup \mathsf{Con}_0$ be the set of Π_0^C-clauses obtained this way.

Step 2 Let $G_1 = \mathcal{K}_0 \cup G_0 \cup \mathsf{Con}_0$. Among the constants in G_1, we identify

(i) the constants c_f, $f \in \Sigma_p$, where c_f is a constant parameter or c_f is introduced by a definition $c_f \approx f(c_1, \ldots, c_k)$ in the hierarchical reasoning method,

(ii) all constants $\overline{c}_p$ which are not parameters and occur as arguments of functions in Σ_p in such definitions.

Replace all the other constants $\overline{c}$ with existentially quantified variables $\overline{x}$ (i.e. replace $G_1(\overline{c}_p, \overline{c}_f, \overline{c})$ with $\exists \overline{x} G_1(\overline{c}_p, \overline{c}_f, \overline{x})$).

Step 3 Construct a formula $\Gamma_1(\overline{c}_p, \overline{c}_f)$ equivalent to $\exists \overline{x} G_1(\overline{c}_p, \overline{c}_f, \overline{x})$ w.r.t. $\mathcal{T}_0$ using a method for quantifier elimination in $\mathcal{T}_0$.

Step 4 Replace each constant c_f introduced by a definition $c_f = f(c_1, \ldots, c_k)$ with the term $f(c_1, \ldots, c_k)$ in $\Gamma_1(\overline{c}_p, \overline{c}_f)$. Let $\Gamma_2(\overline{c}_p)$ be the formula obtained this way. Replace $\overline{c}_p$ with existentially quantified variables $\overline{y}$.

Step 5 Let $\forall \overline{y} \Gamma_T(\overline{y})$ be $\forall \overline{y} \neg \Gamma_2(\overline{y})$.

Theorem 3.5 (cf. also [98]). *Assume that $\mathcal{K} = \mathcal{K}_p \cup \mathcal{K}_1$ such that $\mathcal{K}_p$ contains only symbols in $\Sigma_0 \cup \Sigma_p$ and $\mathcal{K}_1$ is a set of Π-clauses such that*

$$\mathcal{T}_0 \subseteq \mathcal{T}_0 \cup \mathcal{K}_p \subseteq \mathcal{T}_0 \cup \mathcal{K}_p \cup \mathcal{K}_1$$

is a chain of theory extensions both satisfying condition ($\mathsf{Comp_f}$) and having the property that all variables occur below an extension function, and such that $\mathcal{K}$ is flat and linear. Let G be a set of flat and linear ground Π^C-clauses, and $\forall \overline{y} \Gamma_G(\overline{y})$ be the formula obtained with Algorithm 2 for $T = \mathsf{est}(\mathcal{K}, G)$. Then $\forall \overline{y} \Gamma_G(\overline{y})$ is entailed by every universal $\Pi_0 \cup \Sigma_p$-formula Γ which entails $\mathcal{K}_p$ and for which $\mathcal{T}_0 \cup \Gamma \cup \mathcal{K} \cup G \models \bot$.

A similar result holds if T is the set of instances obtained from the instantiation of a chain of theory extensions $\mathcal{T}_0 \subseteq \mathcal{T}_0 \cup \mathcal{K}_1 \subseteq \cdots \subseteq \mathcal{T}_0 \cup \mathcal{K}_1 \cup \cdots \cup \mathcal{K}_n$, all satisfying condition ($\mathsf{Comp_f}$), and where $\mathcal{K}_1, \ldots, \mathcal{K}_n$ are all flat and linear [100].

In the following example we illustrate how Algorithm 2 can be applied to compute additional constraints on parameters such that a desired property, expressed by a universal formula, holds.

Example 3.6. In Example 2.26 we showed that if two functions f and g are monotone, then $f + g$ is also monotone. We now assume that we only know that f is monotone and we want to find conditions on g such that the sum of f and g is monotone. Let Π_0 be the signature of $\mathsf{LI}(\mathbb{R})$ and let $\Sigma_1 = \{f, g\}$ and $\Sigma_p = \{g\}$. We have the local theory extension $\mathcal{T}_0 \subseteq \mathcal{T}_1 = \mathcal{T}_0 \cup \mathcal{K}$, where $\mathcal{T}_0 = \mathsf{LI}(\mathbb{R})$ and $\mathcal{K} = \mathsf{Mon}(f) = \{\forall x, y(x \leq y \rightarrow f(x) \leq f(y)\}$. The sum of the functions f and g is monotone if $\mathcal{T}_0 \cup \mathcal{K} \cup G$ is unsatisfiable, where

$$G = a \leq b \wedge (f(a) + g(a) > f(b) + g(b)).$$

However, $\mathcal{T}_0 \cup \mathcal{K} \cup G$ is satisfiable. Let $\Sigma_p = \{g\}$. From Example 2.21 we know that the theory extension $\mathcal{T}_0 \subseteq \mathcal{T}_1 = \mathsf{LI}(\mathbb{R}) \cup \mathsf{Mon}(f)$ is local and we know that $\mathsf{LI}(\mathbb{R})$ allows quantifier elimination, so by Theorem 3.4 we can use Algorithm 2 to compute a $\Pi_0 \cup \Sigma_p$-formula $\forall \overline{y} \Gamma_T(\overline{y})$ such that $\mathcal{T}_0 \cup \forall \overline{y} \Gamma_T(\overline{y}) \cup \mathcal{K} \cup G$ is unsatisfiable, which means that $\forall \overline{y} \Gamma_T(\overline{y})$ is a condition on g which assures monotonicity of $f + g$. Additionally, since the theory extension $\mathcal{T}_0 \subseteq \mathcal{T}_1$ satisfies condition ($\mathsf{Comp_f}$) and $\mathcal{K}$ is flat and linear, from Theorem 3.5 it follows that this must be the weakest such condition.

Step 1: The extension terms starting with g in G are $g(a)$ and $g(b)$, but since we do not have axioms for g, they are not needed for instantiation. The extension terms starting with f in G are $f(a)$ and $f(b)$, therefore we have the following set of non-redundant instances:

$$\mathsf{Mon}(f)[G] = \{(a \leq b \rightarrow f(a) \leq f(b)), (b \leq a \rightarrow f(b) \leq f(a))\}$$

Using purification we obtain:

$$\begin{aligned} \mathsf{Def} &= \{a_1 \approx f(a), b_1 \approx f(b), a_2 \approx g(a), b_2 \approx g(b)\} \\ \mathcal{K}_0 \cup G_0 \cup \mathsf{Con}_0 &= (a \leq b \rightarrow a_1 \leq b_1) \wedge (b \leq a \rightarrow b_1 \leq a_1) \wedge \\ &\quad (a \leq b) \wedge (a_1 + a_2 > b_1 + b_2) \wedge \\ &\quad (a \approx b \rightarrow a_1 \approx b_1) \wedge (a \approx b \rightarrow a_2 \approx b_2) \end{aligned}$$

Step 2: We want to eliminate all constants not related to the parameters $\Sigma_p = \{g\}$. We identify the three sets of constants according to the algorithm:

- a_2 and b_2 are introduced as names for $g(a)$ and $g(b)$, respectively, so $\overline{c}_f = a_2, b_2$.
- a and b appear as arguments in $g(a)$ and $g(b)$, respectively, so $\overline{c}_p = a, b$.
- The remaining constants in $\mathcal{K}_0 \cup G_0 \cup \mathsf{Con}_0$ are a_1 and b_1, so $\overline{c} = a_1, b_1$.

We replace the constants $\overline{c} = a_1, b_1$ with existentially quantified constants (of the same name) and obtain the formula $\exists a_1, b_1 \; G_1(a, b, a_2, b_2, a_1, b_1)$:

$$\begin{aligned} \exists a_1, b_1 (\; & (a \leq b \rightarrow a_1 \leq b_1) \wedge (b \leq a \rightarrow b_1 \leq a_1) \wedge \\ & (a \leq b) \wedge (a_1 + a_2 > b_1 + b_2) \wedge \\ & (a \approx b \rightarrow a_1 \approx b_1) \wedge (a \approx b \rightarrow a_2 \approx b_2) \;) \end{aligned}$$

Step 3: We can use a system such as Redlog for quantifier elimination to obtain the following formula equivalent to $\exists a_1, b_1 \; G_1(a, b, a_2, b_2, a_1, b_1)$:

$$\Gamma_1(a, b, a_2, b_2) = a < b \wedge a_2 > b_2$$

Step 4: By replacing back a_2 with $g(a)$ and b_2 with $g(b)$ and quantifying a and b existentially we get the following:

$$\exists a, b \; \Gamma_2(a, b) = \exists a, b \; (a < b \wedge g(a) > g(b))$$

Step 5: Negating the formula above yields the following result:

$$\forall a, b \; \Gamma_T(a, b) = \forall a, b \; (a \geq b \vee g(a) \leq g(b)) \equiv \forall a, b \; (a < b \rightarrow g(a) \leq g(b))$$

Therefore we know that $\forall a, b \; (a < b \rightarrow g(a) \leq g(b))$, i.e. the monotonicity property for g, is the weakest condition on g such that the sum of g and any monotone function f is also monotone. ■

3.2.1 Improvement of the Algorithm

Algorithm 2 uses a hierarchical approach in which we reduce eliminating symbols in $\Sigma \setminus \Sigma_p$ to quantifier elimination in the base theory $\mathcal{T}_0$. However, quantifier elimination usually has high complexity and can lead to very large formulae. Therefore we are interested in making the algorithm more efficient. One way to improve the algorithm is to apply quantifier elimination not to the whole formula, but only to a smaller part of it. This can often be done, as we have a set of parameters Σ_p which we do not want to eliminate. This means, considering our set of clauses $\mathcal{K}$ can be divided into a part which contains only parameters (function symbols in Σ_p) and a part which also contains non-parametric symbols (function symbols in Σ), i.e. $\mathcal{K} = \mathcal{K}_p \cup \mathcal{K}_1$ where $\mathcal{K}_p$ contains only parameters from Σ_p, we do not need to eliminate any symbols from $\mathcal{K}_p$. It is therefore sufficient to apply quantifier elimination on the rest of the formula, leaving $\mathcal{K}_p$ out of the scope of the quantifiers.

Theorem 3.7. *Assume that $\mathcal{K} = \mathcal{K}_p \cup \mathcal{K}_1$ such that $\mathcal{K}_p$ contains only symbols in $\Sigma_0 \cup \Sigma_p$ and $\mathcal{K}_1$ is a set of Π-clauses such that*

$$\mathcal{T}_0 \subseteq \mathcal{T}_0 \cup \mathcal{K}_p \subseteq \mathcal{T}_0 \cup \mathcal{K}_p \cup \mathcal{K}_1$$

is a chain of theory extensions both satisfying condition (Comp_f) and having the property that all variables occur below an extension function, and such that $\mathcal{K}$ is flat and linear. Let G be a set of flat and linear ground Π^C-clauses. Then the formula $\mathcal{K}_p \wedge \forall \overline{y} \Gamma_1(\overline{y})$, where $\forall \overline{y} \Gamma_1(\overline{y})$ is obtained by applying Algorithm 2 to $\mathcal{T}_0 \cup \mathcal{K}_1 \cup G$, has the property that for every universal formula Γ containing only parameters with $\mathcal{T}_0 \cup (\mathcal{K}_p \cup \Gamma) \cup G \models \bot$, we have $\mathcal{K}_p \wedge \Gamma \models \mathcal{K}_p \wedge \forall \overline{y} \Gamma_1(\overline{y})$.

Proof: Assume that we have the following chain of local theory extensions:

$$\mathcal{T}_0 \subseteq \mathcal{T}_0 \cup \mathcal{K}_p \subseteq \mathcal{T}_0 \cup \mathcal{K}_p \cup \mathcal{K}_1$$

We assume that the clauses in $\mathcal{K}_p \cup \mathcal{K}_1$ are flat and linear and that for each of these extensions any variable occurs below an extension function. For the sake of simplicity we assume that Σ contains only one function symbol which we want to eliminate. For several function symbols the procedure is analogous.

Let G be a set of flat ground clauses. We know that the following are equivalent:

(1) $\mathcal{T}_0 \cup \mathcal{K}_p \cup \mathcal{K}_1 \cup G$ is satisfiable.

(2) $\mathcal{T}_0 \cup \mathcal{K}_p \cup \mathcal{K}_1[G] \cup G$ is satisfiable, where the extension terms used in the instantiation correspond to the set $\mathsf{est}(G) = \{f(d_1), \ldots, f(d_k)\}$, where for every i, $d_i = (d_i^1, \ldots, d_i^{a(f)})$, where $a(f)$ is the arity of f.

(3) The set of formulae obtained after purification (in which each extension term $f(d)$ is replaced with a new constant c_{fd}) and the inclusion of instances of the congruence axioms, $\mathcal{T}_0 \cup \mathcal{K}_p \cup (\mathcal{K}_1[G])_0 \cup G_0 \cup \mathsf{Con}_0(G)$, is satisfiable, where

$$\mathsf{Con}_0(G) = \{ \bigwedge_{j=1}^{a(f)} d_n^j \approx d_m^j \rightarrow c_{fd_n} \approx c_{fd_m} \mid f(d_n), f(d_m) \in \mathsf{est}(G) \}.$$

(4) $\mathcal{T}_0 \cup \mathcal{K}_p[G_1] \cup G_1$ is satisfiable, where $G_1 = (\mathsf{Def}_f[G])_0 \cup G_0 \cup \mathsf{Con}_0(G)$.

(5) The purified form of the formula above, $\mathcal{T}_0 \cup (\mathcal{K}_p[G_1])_0 \cup (G_1)_0 \cup \mathsf{Con}_0(G_1)$, is satisfiable.

We use the following set of ground terms:

$$T = \mathsf{est}(G) \cup \mathsf{est}(G_1) = \{f(d_1), \ldots, f(d_k)\} \cup \{g(c) \mid g \in \Sigma_p, g(c) \in \mathsf{est}(G_1)\}$$

(a set of flat terms in which we isolated the terms starting with the function symbol f).

We apply Algorithm 2:

Step 1: We perform the hierarchical reduction in two steps.
In the first step we introduce a constant c_{fd} for every term $f(d) \in \mathsf{est}(G)$. After this first reduction we obtain $G_1 = \mathcal{K}_1[G] \cup G_0 \cup \mathsf{Con}_0(G)$, where

$$G_1 \;=\; \mathcal{K}_1[G] \;\wedge G_0 \;\wedge \bigwedge_{f(d_n), f(d_m) \in \mathsf{est}(G)} \big(\bigwedge_{j=1}^{a(f)} d_n^j \approx d_m^j \rightarrow c_{fd_n} \approx c_{fd_m} \big).$$

In the second reduction we replace every term $g(c) \in \mathsf{est}(G_1)$, where $g \in \Sigma_p$, with a new constant c_{gc} and add the corresponding congruence axioms. We obtain

$$G_1' = (\mathcal{K}_p[G_1])_0 \wedge (G_1)_0 \wedge \mathsf{Con}_0(G_1).$$

Step 2: We want to eliminate the function symbol f. We assume that all the other function symbols are either parametric or in $\mathcal{T}_0$. We therefore replace the constants c_{fd} with variables x_{fd}.

Step 3: Since $\mathcal{K}_p$ does not contain any function symbol in Σ, neither $(\mathcal{K}_p[G_1])_0$ nor $\mathsf{Con}_0(G_1)$ contain the variables x_{fd}. Therefore:

$$\exists x_{fd_1}, \ldots, x_{fd_k}\; G_1'(x_{fd_1}, \ldots, x_{fd_k}) \;\equiv\; (\mathcal{K}[G_1])_0 \wedge \mathsf{Con}_0(G_1) \wedge \exists x_{fd_1}, \ldots, x_{fd_k}(G_1)_0$$

After we have applied quantifier elimination w.r.t. $\mathcal{T}_0$ we obtain a quantifier-free Π^C-formula $D_0 \equiv \exists x_{fd_1}, \ldots, x_{fd_k}(G_1)_0$, hence

$$\exists x_{fd_1}, \ldots, x_{fd_k}\; G_1'(x_{fd_1}, \ldots, x_{fd_k}) \equiv (\mathcal{K}[G_1])_0 \wedge \mathsf{Con}_0(G_1) \wedge D_0.$$

Step 4: We replace back in the formula obtained this way all constants c_{gc}, where $g \in \Sigma_p$ and $g(c) \in \mathsf{est}(G_1)$, with the terms $g(c)$ and remove $\mathsf{Con}_0(G_1)$. This restores $\mathcal{K}_p[G_1]$ and all formulae which did not contain f in G_1'. We therefore obtain

$$\Gamma_2(\overline{d}) = \mathcal{K}_p[G_1](\overline{d}, \overline{c}, \overline{g(c)}) \wedge D(\overline{d}, \overline{c}, \overline{g(c)}).$$

We replace the constants $\overline{d}, \overline{c}$ which are not parameters with variables $\overline{y}$ and obtain

$$\Gamma_2(\overline{y}) = \mathcal{K}_p[G_1](\overline{y}, \overline{g(y)}) \wedge D(\overline{y}, \overline{g(y)}).$$

If we analyze the formula $\mathcal{K}_p[G_1](\overline{y}, \overline{g(y)})$ obtained from $\mathcal{K}_p[G_1]$ by replacing the constants $\overline{d}, \overline{c}$ with variables $\overline{y}$, we notice that the constants substituted for variables in $\mathcal{K}_p[G_1]$ are replaced back with new variables. Thus, $\mathcal{K}_p[G_1](\overline{y}, \overline{g(y)}) = \bigwedge_{i \in I} \mathcal{K}_p\sigma_i$, which is a finite conjunction, where I is the set of instances of $\mathcal{K}_p[G_1](\overline{y}, \overline{g(y)})$ and for every $i \in I$, $\sigma_i : X \rightarrow X$ is a substitution (not necessarily injective) that might rename the variables in $\mathcal{K}_p$.

Step 5: We negate $\exists \overline{y} \Gamma_2(\overline{y})$ and obtain

$$\neg \exists \overline{y} \Gamma_2(\overline{y}) \equiv \forall \overline{y} \neg \Gamma_2(\overline{y}) \equiv \forall \overline{y} (\bigvee_{i \in I} \neg \mathcal{K}_p\sigma_i \vee \neg D(\overline{y}, \overline{g(y)})).$$

If the goal is to strengthen the already existing constraints $\mathcal{K}_p$ on the parameters with an additional (universally quantified) condition Γ such that $\mathcal{T}_0 \cup \mathcal{K}_p \cup \Gamma \cup \mathcal{K}_1 \cup G \models \bot$, note the following:

$$\begin{array}{rcl} \forall \overline{x} \mathcal{K}_p(\overline{x}) \wedge \forall \overline{y}(\bigvee\limits_{i \in I} \neg \mathcal{K}_p \sigma_i \vee \neg D(\overline{y}, \overline{g(y)})) & \equiv & \forall \overline{y}(\bigvee\limits_{i \in I} (\forall \overline{x} \mathcal{K}_p(\overline{x}) \wedge \neg \mathcal{K}_p \sigma_i) \vee \\ & & \quad (\forall \overline{x} \mathcal{K}_p(\overline{x}) \wedge \neg D(\overline{y}, \overline{g(y)})) \ ) \\ & \equiv & \forall \overline{y}(\ \forall \overline{x} \mathcal{K}_p(\overline{x}) \wedge \neg D(\overline{y}, \overline{g(y)}) \ ) \\ & \equiv & \forall \overline{x} \mathcal{K}_p(\overline{x}) \wedge \forall \overline{y} \neg D(\overline{y}, \overline{g(y)}) \end{array}$$

We now prove that the formula $\Gamma_1 = \forall \overline{y} \neg D(\overline{y})$ is the weakest formula Γ with the property that $\mathcal{T}_0 \cup \mathcal{K}_p \cup \Gamma \cup \mathcal{K}_1 \cup G$, i.e. that for every set Γ of constraints on the parameters, if $\mathcal{T}_0 \cup \mathcal{K}_p \cup \Gamma \cup \mathcal{K}_1 \cup G$ is unsatisfiable, then every model of $\mathcal{T}_0 \cup \mathcal{K}_p \cup \Gamma$ is a model of $\mathcal{T}_0 \cup \mathcal{K}_p \cup \Gamma_1$.

We know that if the extensions $\mathcal{T}_0 \subseteq \mathcal{T}_0 \cup \mathcal{K}_p \subseteq \mathcal{T}_0 \cup \mathcal{K}_p \cup \mathcal{K}_1$ satisfy condition ($\mathsf{Comp_f}$) and Γ contains only symbols in $\Sigma_0 \cup \Sigma_p$, then also the extensions $\mathcal{T}_0 \cup \Gamma \subseteq \mathcal{T}_0 \cup \Gamma \cup \mathcal{K}_p \subseteq \mathcal{T}_0 \cup \Gamma \cup \mathcal{K}_p \cup \mathcal{K}_1$ satisfy condition ($\mathsf{Comp_f}$) [64]. If $\mathcal{K}_p \cup \mathcal{K}_1$ is flat and linear, then the extensions are local. Let $T = \mathsf{est}(\mathcal{K}_1, G)$. By locality, $\mathcal{T}_0 \cup \Gamma \cup \mathcal{K}_p \cup \mathcal{K}_1 \cup G$ is unsatisfiable if and only if $\mathcal{T}_0 \cup \Gamma \cup \mathcal{K}_p \cup \mathcal{K}_1[G] \cup G$ is unsatisfiable, which is the case if and only if $\mathcal{T}_0 \cup \Gamma \cup \mathcal{K}_p \cup (\mathcal{K}_1[G])_0 \cup G_0 \cup \mathsf{Con}_0 \cup \mathsf{Def}$ is unsatisfiable. Let $\mathcal{A}$ be a model of $\mathcal{T}_0 \cup \Gamma \cup \mathcal{K}_p$. Then $\mathcal{A}$ cannot be a model of $(\mathcal{K}_1[G])_0 \cup G_0 \cup \mathsf{Con}_0 \cup \mathsf{Def}$, so (with the notation used in Steps 1–5) $\mathcal{A} \not\models D(\overline{d}, \overline{c}, \overline{g}(\overline{c}))$, i.e. $\mathcal{A} \not\models \exists \overline{y} D(\overline{y}, \overline{g}(\overline{y}))$. It follows that $\mathcal{A} \models \forall \overline{y} \neg D(\overline{y}, \overline{g}(\overline{y}))$. Thus, $\mathcal{A} \models \mathcal{K}_p \wedge \Gamma_1$. □

We illustrate the ideas of the improvement of the symbol elimination algorithm based on Theorem 3.7 on an adaptation of an example first presented in [98].

Example 3.8. Let $\mathcal{T}_0 = \mathsf{LI}(\mathbb{R})$. Consider the extension of $\mathcal{T}_0$ with the function symbols $\Sigma_1 = \{f, g, h, c\}$. Assume $\Sigma_p = \{f, h, c\}$ and $\Sigma = \{g\}$, and the properties of these function symbols are axiomatized by $\mathcal{K}_p \cup \mathcal{K}$, where

$$\begin{array}{lll} \mathcal{K}_p \coloneqq \{ & \forall x, y(c < x \leq y \rightarrow f(x) \leq f(y)), & \forall x, y(x \leq y < c \rightarrow h(x) \leq h(y)) \quad \}, \\ \mathcal{K} \coloneqq \{ & \forall x(x \leq c \rightarrow g(x) \approx f(x)), & \forall x(c < x \rightarrow g(x) \approx h(x)) \quad \}. \end{array}$$

We are interested in generating a set of additional constraints on the functions f and h which ensure that g is monotone, i.e. satisfies

$$\mathsf{Mon}(g): \quad \forall x, y(x \leq y \rightarrow g(x) \leq g(y)).$$

For this we have to generate a set Γ of $\Sigma_0 \cup \Sigma_p$-constraints such that

$$\mathcal{T}_0 \cup \mathcal{K}_p \cup \Gamma \cup \mathcal{K} \cup \{c_1 \leq c_2, g(c_1) > g(c_2)\}$$

is unsatisfiable, where $G = \{c_1 \leq c_2, g(c_1) > g(c_2)\}$ is the Skolemized negation of $\mathsf{Mon}(g)$. We have the following chain of theory extensions:

$$\mathcal{T}_0 \subseteq \mathcal{T}_0 \cup \mathcal{K}_p \subseteq \mathcal{T}_0 \cup \mathcal{K}_p \cup \mathcal{K}$$

Both extensions satisfy the condition ($\mathsf{Comp_f}$), and $\mathcal{T}_0 \cup \mathcal{K}_p \cup \mathcal{K} \cup G$ is satisfiable if and only if $\mathcal{T}_0 \cup \mathcal{K}_p \cup \mathcal{K}[G] \cup G$ is satisfiable, where:

$$\begin{array}{lll} \mathcal{K}[G] \coloneqq \{ & c_1 \leq c \rightarrow g(c_1) \approx f(c_1), & c_2 \leq c \rightarrow g(c_2) \approx f(c_2), \\ & c < c_1 \rightarrow g(c_1) \approx h(c_1), & c < c_2 \rightarrow g(c_2) \approx h(c_2) \quad \} \end{array}$$

We construct Γ as follows:

Step 1 Since we have a chain of two local extensions, we perform a two-step reduction: First we compute $\mathcal{T}_0 \cup \mathcal{K}[G] \cup G$, then we purify it by introducing new constants g_1, g_2 for the terms $g(c_1), g(c_2)$. We obtain

$$\mathsf{Def}_1 = \{g_1 \approx g(c_1), g_2 \approx g(c_2)\} \text{ and}$$

$$\begin{aligned}\mathcal{K}_0 \cup \mathsf{Con}_0 \cup G_0 \coloneqq \{ \quad & c_1 \leq c \to g_1 \approx f(c_1), \quad c_2 \leq c \to g_2 \approx f(c_2), \\ & c < c_1 \to g_1 \approx h(c_1), \quad c < c_2 \to g_2 \approx h(c_2), \\ & c_1 \approx c_2 \to g_1 \approx g_2, \; c_1 \leq c_2, \quad g_1 > g_2 \quad \}.\end{aligned}$$

In the next step we introduce new constants f_1, f_2, h_1 and h_2 for the terms $f(c_1)$, $f(c_2)$, $h(c_1)$ and $h(c_2)$, respectively. We obtain

$$\mathsf{Def}_2 = \{f_1 \approx f(c_1), f_2 \approx f(c_2), h_1 \approx h(c_1), h_2 \approx h(c_2)\}.$$

We do not need to effectively perform the instantiation of the axioms in $\mathcal{K}_p$ or consider the instances of the congruence axioms for the functions in Σ_p. We restrict to computing $(\mathcal{K}_0 \cup \mathsf{Con}_0 \cup G_0)_0$:

$$\begin{aligned}(\mathcal{K}_0 \cup \mathsf{Con}_0 \cup G_0)_0 \coloneqq \{ \quad & c_1 \leq c \to g_1 \approx f_1, \quad c_2 \leq c \to g_2 \approx f_2, \\ & c < c_1 \to g_1 \approx h_1, \quad c < c_2 \to g_2 \approx h_2, \\ & c_1 \approx c_2 \to g_1 \approx g_2, \quad c_1 \leq c_2, \quad g_1 > g_2 \; \}\end{aligned}$$

Step 2 The parameters are contained in the set $\Sigma_p = \{f, h, c\}$. We want to eliminate the function symbol g, so we replace g_1, g_2 with existentially quantified variables z_1, z_2 and obtain an existentially quantified formula:

$$\begin{aligned}\exists z_1, z_2 \big(\quad & (c_1 \leq c \to z_1 \approx f_1) \wedge (c_2 \leq c \to z_2 \approx f_2) \wedge \\ & (c < c_1 \to z_1 \approx h_1) \wedge (c < c_2 \to z_2 \approx h_2) \wedge \\ & (c_1 \approx c_2 \to z_1 \approx z_2) \wedge c_1 \leq c_2 \wedge z_1 > z_2 \; \big)\end{aligned}$$

Step 3 If we use Redlog for quantifier elimination, the obtained formula is quite long and incomprehensible. Therefore we instead use QEPCAD[1], which can often simplify the obtained formulae better than Redlog. Using QEPCAD we obtain the following formula:

$\Gamma_1(c_1, c_2, c, f_1, f_2, h_1, h_2)$:

$$c_2 > c_1 \wedge \big((c_2 \leq c \wedge f_2 < f_1) \vee (c_1 > c \wedge h_2 < h_1) \vee (c_1 \leq c \wedge c_2 > c \wedge h_2 < f_1) \big)$$

Step 4 We construct the formula $\Gamma_2(c_1, c_2, c)$ from Γ_1 by replacing f_i by $f(c_i)$ and h_i by $h(c_i)$, for $i \in \{1, 2\}$. We obtain

$$\begin{aligned}c_2 > c_1 \wedge \big(\quad & (c_2 \leq c \;\wedge\; f(c_2) < f(c_1)) \;\vee\; (c_1 > c \;\wedge\; h(c_2) < h(c_1)) \vee \\ & (c_1 \leq c \;\wedge\; c_2 > c \;\wedge\; h(c_2) < f(c_1)) \big).\end{aligned}$$

After replacing c_1 with a variable x_1 and c_2 with a variable x_2 we obtain

$$\begin{aligned}x_2 > x_1 \wedge \big(\quad & (x_2 \leq c \;\wedge\; f(x_2) < f(x_1)) \;\vee\; (x_1 > c \;\wedge\; h(x_2) < h(x_1)) \vee \\ & (x_1 \leq c \;\wedge\; x_2 > c \;\wedge\; h(x_2) < f(x_1)) \big).\end{aligned}$$

[1] QEPCAD is a tool for quantifier elimination in the real numbers based on the method of cylindrical algebraic decomposition. See https://www.usna.edu/Users/cs/wcbrown/qepcad/B/QEPCAD.html for more information on QEPCAD.

Step 5 After negation and universal quantification of the variables we obtain

$$\forall x_1, x_2 \big(x_2 \leq x_1 \vee \big(\quad \begin{array}{l} (x_2 \leq c \rightarrow f(x_1) \leq f(x_2)) \wedge \\ (x_1 > c \rightarrow h(x_1) \leq h(x_2)) \wedge \\ (x_1 \leq c \ \wedge \ x_2 > c \rightarrow f(x_1) \leq h(x_2)) \ \big) \big) \end{array}$$

which is a set of contraints on f and h that guarantees monotonicity of g. ■

3.2.2 Implementation of the Algorithm

The symbol elimination procedure described in Algorithm 2 was implemented by Philipp Marohn as part of his Bachelor thesis [77] (with me and Prof. Dr. Sofronie-Stokkermans being the supervisors) and updated with more features later on. The program is called SEH-PILoT (short for Symbol Elimination based on Hierarchical Proving In Local Theory Extensions) and is implemented in Python 3.9. Its primary purpose is to perform property-directed symbol elimination in theory extensions, i.e. to generate constraints on parameters such that a given property holds, but it additionally offers other useful features as well.

In the following we give a brief overview on the latest version of SEH-PILoT at the time of writing this thesis. The description is based on a preliminary system description [78], which describes an older version of SEH-PILoT and therefore does not include all the features of the current version. Note that the implementation is still ongoing, which means that things may be changed and new features will be added in the future.

The input for SEH-PILoT is a YAML-file[2] in which a **mode**, various **options**, a **specification type**, a **specification theory** and the **specification** are specified. When SEH-PILoT is used for symbol elimination in general (and not specifically for parametric transition systems), one has to set the specification type to HPILOT. In this case the specification is just a plain input file for H-PILoT. The specification theory can be either REAL_CLOSED_FIELDS or PRESBURGER_ARITHMETIC. One can choose between the three modes GENERATE_CONSTRAINTS, SYMBOL_ELIMINATION and CHECK_SATISFIABILITY. In what follows we first illustrate the workflow of SEH-PILoT when using mode GENERATE_CONSTRAINTS.

For the most part SEH-PILoT in the mode GENERATE_CONSTRAINTS follows the steps of Algorithm 2 and uses Redlog for quantifier elimination. One minor difference is that in SEH-PILoT the negation is done in Step 3 right after the quantifier elimination, i.e. before replacing back the constants in Step 4. In Algorithm 2 the formula is negated after the constants have been replaced, i.e. in a fifth step. However, the order in which the negation and the replacement of the constants is done is not important as it does not change the result.

Specifying parameters in SEH-PILoT can be done in one of two ways. Using the option eliminate we can define a set of symbols which are eliminated. Symbols which are not included in this set are interpreted as parameters. Alternatively, using the option parameter we can do the opposite. We specify a set of parameters and all the symbols not included in this set will be eliminated. Note that in mode GENERATE_CONSTRAINTS and in mode SYMBOL_ELIMINATION exactly one of the two options must be used.

SEH-PILoT offers various options to use simplifiers in order to obtain shorter formulae, as Redlog in many cases does not simplify formulae very well. It is for instance possible to

[2] see https://yaml.org/

use the QEPCAD simplifier SLFQ[3] on the input formula for Redlog and on the result that is obtained after quantifier elimination. For this the options slfq_formula and slfq_query, respectively, can be used.

If SEH-PILoT is used in mode SYMBOL_ELIMINATION, then the specified symbols are eliminated without a negation of the result afterwards. This means that only Steps 1-4 of Algorithm 2 are applied.

If SEH-PILoT is used in mode CHECK_SATISFIABILITY, then the satisfiability of the given specification is checked. This is essentially the same as using H-PILoT on the input. Internally it is done by choosing in Step 2 that all the symbols occuring in the reduced formula (obtained after Step 1) are to be eliminated. Therefore, the options eliminate and parameter may not be used in this mode.

It is possible to specify several tasks in one YAML-file. If several tasks have the same specification, one can set an anchor ("&" plus a string) before the specification is written for the first time and use the alias ("*" plus the same string) in the following tasks instead of writing the whole specification again. The time required for each task (in seconds) is measured separately and displayed in the output file in addition to the total time (in seconds).

We illustrate on two examples the usage of SEH-PILoT with specification type HPILOT in mode CHECK_CONSTRAINTS. We also analyze the benefits of the implemented SLFQ simplifier by comparing the results with and without the option slfq_query.

Example 3.9. Consider again the problem from Example 3.6. We have the theory extension $\mathcal{T}_0 \subseteq \mathcal{T}_0 \cup \mathcal{K}$, where $\mathcal{T}_0 = \mathsf{LI}(\mathbb{R})$ and

$$\mathcal{K} := \{\forall x, y(x \leq y \rightarrow f(x) \approx f(y))\}.$$

We have the signature $\Sigma_1 = \Sigma_p \cup \Sigma$, where $\Sigma_p = \{g\}$ and $\Sigma = \{f\}$. We know that $\mathcal{T}_0 \cup \mathcal{K} \cup G$ is satisfiable, where G is the Skolemized negation of the property to be proved, i.e. $G := a \leq b \wedge (f(a) + g(a) > f(b) + g(b))$.

Using SEH-PILoT we can then generate a constraint containing only symbols occurring in Σ_p (i.e. a constraint on g) which ensures that the sum of f and g is monotone. We have to use SEH-PILoT with specification type HPILOT, specification theory REAL_CLOSED_FIELDS and mode GENERATE_CONSTRAINTS. In this mode it is obligatory to specify either the parameters or, conversely, which symbols are to be eliminated. We here use the option parameter to specify that $\{g\}$ is the set of parameters. The specification is basically an input for H-PILoT. Here we first specifiy the base functions (with corresponding arity), the extension functions (with corresponding arity and level of the extension) as well as the relations (with corresponding arity). It is followed by a set of clauses, which corresponds to the clauses in $\mathcal{K}$, and a query, which corresponds to the Skolemized negation of the property to be proved. We use the anchor "&*spec_ex*3_9" such that we do not have to write the same specification again for the second task.

With the above considerations the input file for SEH-PILoT looks as follows:

[3] see https://www.usna.edu/Users/cs/wcbrown/qepcad/SLFQ/Home.html for more information on SLFQ

```
tasks:
    example_3.9:
        mode: GENERATE_CONSTRAINTS
        options:
            parameter: [g]
            slfq_query: false
        specification_type: HPILOT
        specification_theory: REAL_CLOSED_FIELDS
        specification: &spec_ex3_9
            file: |
                Base_functions := {(+,2), (-,2), (*,2)}
                Extension_functions := {(f,1,2), (g,1,1)}
                Relations := {(<=,2), (<,2), (>=,2), (>,2)}

                Clauses := (FORALL x,y). x <= y --> f(x) <= f(y);

                Query := a <= b;
                         f(a) + g(a) > f(b) + g(b);

    example_3.9_slfq:
        mode: GENERATE_CONSTRAINTS
        options:
            parameter: [g]
            slfq_query: true
        specification_type: HPILOT
        specification_theory: REAL_CLOSED_FIELDS
        specification: *spec_ex3_9
```

We run SEH-PILoT in command line using the command sehpilot example_3.9.yaml where "example_3.9.yaml" is the name of the input file. We obtain the following output:

```
Metadata:
    Date: '2023-07-14 13:53:26'
    Number of Tasks: 2
    Runtime Sum: 0.4934
example_3.9:
    Runtime: 0.1065
    Result: (FORALL a, b). NOT(AND(g(a) - g(b) > _0, a - b < _0))
example_3.9_slfq:
    Runtime: 0.3869
    Result: (FORALL a, b). OR(g(a) - g(b) <= _0, a - b >= _0)
```

First of all, it can be seen that the result we obtain is the same that we obtained by hand in Example 3.6. It expresses the property that g is a monotone function, since the following are equivalent:

$$\neg(g(a) - g(b) > 0 \land a - b < 0) \equiv g(a) - g(b) \leq 0 \lor a - b \geq 0 \equiv a < b \rightarrow g(a) \leq g(b)$$

We now compare the results obtained with and without the SLFQ simplifier. Since we already obtain a very simple formula without the SLFQ simplifier, the difference between the two is rather small. With SLFQ we obtain a slightly shorter formula, in which the negation was already eliminated. This came with the cost of taking almost 4 times as long to compute (0.3869 seconds with simplification and 0.1065 seconds without simplification). We conclude that in this example the simplifier was not necessary. However, often the SLFQ simplifier will make a significant difference in the length and simplicity of the output formula, as the following example shows. ■

Example 3.10. Consider again the problem from Example 3.8. We have the theory extension $\mathcal{T}_0 \subseteq \mathcal{T}_0 \cup \mathcal{K}_p \subseteq \mathcal{T}_0 \cup \mathcal{K}_p \cup \mathcal{K}$, where $\mathcal{T}_0 = \mathsf{LI}(\mathbb{R})$ and

$$\mathcal{K}_p \coloneqq \{ \quad \forall x, y(c < x \leq y \rightarrow f(x) \leq f(y)), \quad \forall x, y(x \leq y < c \rightarrow h(x) \leq h(y)) \quad \},$$

$$\mathcal{K} \coloneqq \{ \quad \forall x(x \leq c \rightarrow g(x) \approx f(x)), \qquad \forall x(c < x \rightarrow g(x) \approx h(x)) \quad \}.$$

We have the signature $\Sigma_1 = \Sigma_p \cup \Sigma$, where $\Sigma_p = \{f, h, c\}$ and $\Sigma = \{g\}$. Since we want to prove that g is monotone, our formula G, the Skolemized negation of the property to be proved, is as follows:

$$G \coloneqq c_1 \leq c_2 \wedge g(c_1) > g(c_2)$$

Using SEH-PILoT we can generate a constraint containing only symbols occurring in Σ_p which ensures that g is monotone. We use the improvement of the algorithm, which means that we can disregard $\mathcal{K}_p$. We have to use SEH-PILoT with specification type HPILOT, specification theory REAL_CLOSED_FIELDS and mode GENERATE_CONSTRAINTS. This time we use the option eliminate to specify that $\{g\}$ is the set of symbols which are eliminated. For comparing the output with and without the use of the SLFQ simplifier, we state two tasks. For the first task we set the option slfq_query to false and for the second one to true in order to simplify the result of the quantifier elimination. The specification is again an input for H-PILoT, where we state the base functions, extension functions, relations, the clauses in $\mathcal{K}$ and the query G.

With the above considerations the input file for SEH-PILoT looks as follows:

```
tasks:
    example_3.10:
        mode: GENERATE_CONSTRAINTS
        options:
            eliminate: [g]
            slfq_query: false
        specification_type: HPILOT
        specification_theory: REAL_CLOSED_FIELDS
        specification: &spec_ex3_10
            file: |
                Extension_functions := {(f,1,1), (h,1,1), (g,1,2)}
                Relations := {(<=,2), (<,2), (>=,2), (>,2)}

                Clauses := (FORALL x). x <= c --> g(x) = f(x);
                           (FORALL x). c < x --> g(x) = h(x);

                Query := c1 <= c2;
                         g(c1) > g(c2);

    example_3.10_slfq:
        mode: GENERATE_CONSTRAINTS
        options:
            eliminate: [g]
            slfq_query: true
        specification_type: HPILOT
        specification_theory: REAL_CLOSED_FIELDS
        specification: *spec_ex3_10
```

We run SEH-PILoT on the input file above and obtain the following output:

```
Metadata:
    Date: '2023-07-14 13:53:31'
    Number of Tasks: 2
    Runtime Sum: 0.53
example_3.10:
    Runtime: 0.1119
    Result: (FORALL c2, c1). NOT(OR(AND(h(c1) - h(c2) > _0, c1 - c2 < _0,
                                        OR(f(c2)- h(c2) = _0, c - c2 < _0),
                                        OR(f(c1) - h(c1) = _0, c - c1 < _0)),
                                    AND(f(c2)- h(c1) < _0, c1 - c2 < _0,
                                        OR(f(c2) - h(c2) = _0, c - c2 >= _0),
                                        OR(f(c1) - h(c1) = _0, c - c1 < _0)),
                                    AND(f(c1) - h(c2) > _0, c1 - c2 < _0,
                                        OR(f(c2) - h(c2) = _0, c - c2 < _0),
                                        OR(f(c1) - h(c1) = _0, c - c1 >= _0)),
                                    AND(f(c1) - f(c2) > _0, c1 - c2 < _0,
                                        OR(f(c2) - h(c2) = _0, c - c2 >= _0),
                                        OR(f(c1)- h(c1) = _0, c - c1 >= _0))))
example_3.10_slfq:
    Runtime: 0.4181
    Result: (FORALL c2, c1). OR(c1 - c2 >= _0,
                                AND(f(c1) - f(c2) <= _0, c - c2 >= _0),
                                AND(h(c2) - f(c1) >= _0, c - c2 < _0,
                                    c - c1 >= _0),
                                AND(h(c1) - h(c2) <= _0, c - c1 < _0))
```

Comparing the results obtained with and without the SLFQ simplifier one can see a significant difference in the length of the formulae. Using SLFQ, the number of atomic formulae in the computed constraint was reduced from 24 to 8. The simpler formula obtained using SLFQ is similar to the one obtained in Example 3.8. The time needed for the computation is again about 4 times higher with SLFQ (0.4181 seconds) than without the simplifier (0.1119 seconds), but in this case it was worth it, because the formula that is not simplified is not easily comprehensible, wheras the simplified formula is. ■

4 Verification of Parametric Systems

In this chapter we analyze possibilities of using symbol elimination and hierarchical reasoning in the verification of parametric systems, i.e. systems in which not every part is completely specified. For systems in general and parametric systems in particular it is important to be able to verify that they work as intended. One possibility to do so is to show that a certain property, for instance a safety condition, is an invariant of the system, i.e. it holds at all times during the runtime of the system. An invariant is called inductive if it holds in the initial states and is preserved under any update of the system. In this chapter we discuss the problem of checking whether a property is an inductive invariant of a system and propose a method for invariant strengthening, i.e. finding an inductive invariant which entails the property to be proved.

Parts of the results in this chapter were already published in [81, 82].

4.1 Parametric Systems and Problems Related to Their Verification

We consider systems S which can be described using system specifications (for an example cf. [43]) consisting of

- a background theory $\mathcal{T}_S$ describing the datatypes used in the specification and their properties, and
- a transition constraint system $T = (\Sigma, \mathsf{Init}, \mathsf{Update})$ specifying
  - the function symbols Σ (including a set V of functions with arity 0, i.e. the variables of the system) whose values can change over time;
  - a formula Init specifying the properties of the initial state;
  - a formula Update with function symbols in $\Sigma \cup \Sigma'$ (where Σ' consists of copies of symbols in Σ, such that if $f \in \Sigma$, then $f' \in \Sigma'$ is the updated function after the transition).

Essentially a transition constraint system describes the initial state of the system and what happens during an update of the system. In state-based approaches one describes all transitions from one state to the next (where states are defined by the values of the functions in Σ). This can be problematic if there are infinitely many states. In transition constraint systems both the initial states and the updates are described using formulae. In this way the formula Init defines the values of the functions in the initial state. The formula Update specifies how the function values change during an update, where for every $f \in \Sigma$ the corresponding symbol f' represents the values of f after the update.

Example 4.1. Consider the system $S = (\mathcal{T}_S, T)$, where $T_S = LI(\mathbb{R})$ is the background theory and $T = \{\Sigma, \mathsf{Init}, \mathsf{Update}\}$ a transition constraint system, where:

$$\begin{aligned}\Sigma &= \{x, y, f\} \text{ with } x \text{ and } y \text{ of arity zero (i.e. variables) and } f \text{ of arity one}\\ \mathsf{Init} &= \{x \approx 0, y \approx 3, \forall z\ f(z) \approx 1\}\\ \mathsf{Update} &= \{y > 0 \rightarrow (y' \approx y - 1 \wedge x' \approx x + 2 \wedge f'(y) \approx f(y) + 1)\}\end{aligned}$$

Since no update rule for $y \leq 0$ is specified, the values of x, y, and f do not change in this case. After one transition of the system we have the following updated values:

$$\begin{aligned}y' &\approx 3 - 1 \approx 2\\ x' &\approx 0 + 2 \approx 2\\ f'(3) &\approx 1 + 1 \approx 2\end{aligned}$$

After three updates of the system a fixpoint is reached with the following values:

$$\begin{aligned}&x \approx 6\\ &y \approx 0\\ &f(3) \approx f(2) \approx f(1) \approx 2\\ &f(z) \approx 1 \text{ for all } z \notin \{1, 2, 3\}\end{aligned}$$

Note that in this example initial values for all variables and all terms $f(z)$ are defined. Therefore the system is fully specified. ■

If there are variables or function symbols of arity ≥ 1 such that not all their initial values are specified, we call these symbols *parameters*. We usually denote the set of parameters with $\Sigma_p \subseteq \Sigma$. A system containing parameters is called a *parametric system*.

Example 4.2. Consider the system $S = (\mathcal{T}_S, T)$, where $T_S = LI(\mathbb{R})$ is the background theory and $T = \{\Sigma, \mathsf{Init}, \mathsf{Update}\}$ a transition constraint system, where:

$$\begin{aligned}\Sigma &= \{x, y, f\} \text{ with } x \text{ and } y \text{ of arity zero (i.e. variables) and } f \text{ of arity one}\\ \mathsf{Init} &= \{y \approx 3, f(3) \approx 1\}\\ \mathsf{Update} &= \{y > 0 \rightarrow (y' \approx y - 1 \wedge x' \approx x + 2 \wedge f'(y) \approx f(y) + 1)\}\end{aligned}$$

In this case we do not have an initial value specified for x, so by the update rule $x' \approx x + 2$ we cannot compute a concrete value for x'. Also for $f(2)$ there is not an initial value specified, so the rule $f'(2) \approx f(2) + 1$ does not yield a concrete value for $f'(2)$. ■

If we cannot verify a parametric system to be safe, it is possible to generate conditions on the parameters such that safety is guaranteed, or to strengthen a property to an inductive invariant which only contains parameters. We describe several types of verification problems in the following.

Inductive Invariant Checking

An inductive invariant of a system is a formula which is true in the initial states of the system and is preserved during any update of the system. Therefore, checking whether a formula Ψ is an inductive invariant of a system $S = (\mathcal{T}_S, T)$, where T=$(\Sigma, \mathsf{Init}, \mathsf{Update})$, can be done in two steps:

(1) check whether $\mathsf{Init} \models_{\mathcal{T}_S} \Psi$;

(2) check whether $\Psi, \mathsf{Update} \models_{\mathcal{T}_S} \Psi'$, where Ψ' results from Ψ by replacing each $f \in \Sigma$ with $f' \in \Sigma'$.

Checking whether a formula Ψ is an invariant can thus be reduced to checking whether $\neg\Psi'$ is satisfiable w.r.t. a theory $\mathcal{T}$. Even if Ψ is a universally quantified formula (and thus $\neg\Psi'$ is a ground formula) the theory $\mathcal{T}$ can be quite complex: it contains the axiomatization $\mathcal{T}_S$ of the datatypes used in the specification of the system, the formalization of the update rules, as well as the formula Ψ itself. Often Init, Ψ and Update can be expressed as sets of clauses and we have chains of theory extensions:

$$\begin{array}{lllllll} \mathcal{T}_0 & \subseteq & \ldots & \subseteq & \mathcal{T}_S & \subseteq & \mathcal{T}_S \cup \mathsf{Init} \\ \mathcal{T}_0 & \subseteq & \ldots & \subseteq & \mathcal{T}_S & \subseteq & \mathcal{T}_S \cup \Psi \subseteq \mathcal{T}_S \cup \Psi \cup \mathsf{Update} \end{array}$$

In many cases these theory extensions have the property that checking satisfiability of ground formulae w.r.t. $\mathcal{T}$ can be reduced (possibly in several steps) to checking satisfiability w.r.t. $\mathcal{T}_0$. This is the case for instance when the theory extensions in the chains above are *local* (for definitions and further properties cf. Section 2.2).

Failure to prove (1) means that Ψ cannot be strengthened to an inductive invariant. However, it is possible to generate additional conditions on the parameters which can be added to the initial states such that Ψ holds.

Failure to prove (2) means that the formula Ψ is not an invariant or Ψ is not inductive w.r.t. T. If Ψ is not an inductive invariant, we can consider the following two orthogonal problems:

(a) **Constraint synthesis**:

Determine additional constraints on the parameters of the system which guarantee that Ψ is an inductive invariant.

(b) **Invariant generation**:

Determine a formula I containing parameters of the system such that $\mathcal{T}_S \models I \rightarrow \Psi$ and I is an inductive invariant.

Problem (a) was studied in [96, 97]. It can be shown that for theory extensions satisfying property ($\mathsf{Comp_f}$) the formulae obtained using Algorithm 2 for symbol elimination (or its improvement described in Chapter 3) are *weakest* constraints on the parameters which guarantee that Ψ is an inductive invariant.

In this thesis we therefore address only problem (b): We show how symbol elimination can be used for giving a complete method for goal-oriented invariant generation, for invariants containing symbols in a specified signature.

Let S be a system, $\mathcal{T}_S$ be the theory and T=($\Sigma_S, \mathsf{Init}, \mathsf{Update}$) the transition constraint system associated with S. We assume that $\Sigma_S = \Sigma_0 \cup \Sigma_p \cup \Sigma$, where Σ_0 is the signature of the base theory $\mathcal{T}_0$, Σ_p is a set of function symbols assumed to be parametric, and Σ is a set of (non-parametric) function symbols disjoint from $\Sigma_0 \cup \Sigma_p$.

Let $\mathsf{LocSafe}$ be a class of universal formulae over the signature Σ_S. Note that the formulae in $\mathsf{LocSafe}$ can be regarded as stes of clauses.

We make the following three assumptions:

(A1) There exists a chain of local theory extensions $\mathcal{T}_0 \subseteq \cdots \subseteq \mathcal{T}_S \cup \mathsf{Init}$ such that in each extension all variables occur below a function in $\Sigma_p \cup \Sigma$.

(A2) For every $\Psi \in \mathsf{LocSafe}$ there exists a chain of local theory extensions $\mathcal{T}_0 \subseteq \cdots \subseteq \mathcal{T}_S \cup \Psi$ such that in each extension all variables occur below a function in $\Sigma_p \cup \Sigma$.

(A3) The update axioms describe the change of the functions in a set $F \subseteq \Sigma$, depending on a finite set $\{\phi_i \mid i \in J\}$ of mutually exclusive and exhaustive conditions over non-primed symbols, i.e. $\mathsf{Update} = \bigcup_{f \in F} \mathsf{Update}_f$, where Update_f has the form

$$\mathsf{Def}_f := \{\forall \overline{x}(\phi_i^f(\overline{x}) \to C_i^f(f'(\overline{x}), \overline{x})) \mid i \in J\}$$

such that

(1) $\phi_i^f(\overline{x})$ contains no symbols from Σ' and $C_i^f(f'(\overline{x}), \overline{x})$ must contain f'.

(2) $\phi_i(\overline{x}) \wedge \phi_j(\overline{x}) \models_{\mathcal{T}_S} \bot$ for $i \neq j$,

(3) $\mathcal{T}_S \models \bigvee_{i \in J} \phi_i$, and

(4) C_i^f are conjunctions of literals and $\mathcal{T}_S \models \forall \overline{x}(\phi_i^f(\overline{x}) \to \exists y(C_i^f(y, \overline{x})))$ for all $i \in J$.

For **(A3)** we can in particular consider definition updates of the form D_f or updates of the form Bound_f as discussed in Example 2.23.

Example 4.3. Consider the system $S = (\mathsf{LI}(\mathbb{R}), (\Sigma, \mathsf{Init}, \mathsf{Update}))$ from Example 4.1. We have $\mathsf{Update} = \{\phi_1 \to C_1\}$, where $\phi_1 = y > 0$ and

$$C_1 = (y' \approx y - 1 \wedge x' \approx x + 2 \wedge f'(y) \approx f(y) + 1).$$

First of all, we can split Update into three update formulae, one for each variable:

$$\begin{aligned}
\mathsf{Update} &= \mathsf{Update}_x \wedge \mathsf{Update}_y \wedge \mathsf{Update}_f, \text{ where} \\
\mathsf{Update}_x &= y > 0 \to x' \approx x + 2 \\
\mathsf{Update}_y &= y > 0 \to y' \approx y - 1 \\
\mathsf{Update}_f &= \forall z(y > 0 \wedge z \approx y \to f'(z) \approx f(z) + 1) \wedge \\
&\qquad \forall z(y > 0 \wedge z \not\approx y \to f'(z) \approx f(z))
\end{aligned}$$

Note that we now express Update_f as a universal formula which specifies not only a change for $f(y)$, but also for terms $f(z)$ with $z \not\approx y$ (in which case the value does not change). Formally, condition (3) does not hold for the three update formulae, as they do not define an exhaustive case distinction. However, it is easy to make a mutually exclusive case distinction with conditions $\phi_1 ....\phi_n$ exhaustive while preserving the mutual exclusiveness, by defining an "otherwise" case with $\phi_{else} = \neg(\phi_1 \vee ... \vee \phi_n)$. In our example we get $\phi_{else} = y \leq 0$. The values do not change in this case. This leads to the following update formula:

$$\begin{aligned}
\mathsf{Update} &= \mathsf{Update}_x \wedge \mathsf{Update}_y \wedge \mathsf{Update}_f, \text{ where} \\
\mathsf{Update}_x &= (y > 0 \to x' \approx x + 2) \wedge (y \leq 0 \to x' \approx x) \\
\mathsf{Update}_y &= (y > 0 \to y' \approx y - 1) \wedge (y \leq 0 \to y' \approx y) \\
\mathsf{Update}_f &= \forall z(y > 0 \wedge z \approx y \to f'(z) \approx f(z) + 1) \wedge \\
&\qquad \forall z(y > 0 \wedge z \not\approx y \to f'(z) \approx f(z)) \wedge \\
&\qquad \forall z(y \leq 0 \to f'(z) \approx f(z))
\end{aligned}$$

For this update formula the conditions (1)-(4) hold. Since for the non-specified case we can always assume that the values do not change, in the specification we sometimes leave those cases out, but when carrying out the proof tasks they are always included in the corresponding update formulae.

Remark 4.4. *If $\Sigma = \{x_1, ..., x_n\}$ and for some $x_i \in \Sigma$ no formula Update_{x_i} is specified in Update, then we can set $\mathsf{Update}_{x_i} = (x_i' \approx x_i)$ (if no update is specified for a symbol x, then the values of x will not change during the update).*

In what follows, for every formula ϕ containing function symbols in Σ we denote by ϕ' the formula obtained from ϕ by replacing every function symbol $f \in \Sigma$ with the corresponding symbol $f' \in \Sigma'$. For instance, if $\phi = y \approx 1 \wedge \forall x\ f(x) > 0$, then $\phi' = y' \approx 1 \wedge \forall x\ f'(x) > 0$.

Theorem 4.5 ([63, 96]). *The following hold under assumptions **(A1) – (A3)**:*

(1) If ground satisfiability w.r.t. $\mathcal{T}_0$ is decidable, then the problem of checking whether a formula $\Psi \in \mathsf{LocSafe}$ is an inductive invariant of S is decidable.

(2) If $\mathcal{T}_0$ allows quantifier elimination and the initial states or the updates contain parameters, the symbol elimination method in Algorithm 2 yields constraints on these parameters that guarantee that Ψ is an inductive invariant.

Proof: The result was proved in [63, 96]. The proof uses the fact that assumptions **(A1)**, **(A2)** and **(A3)** guarantee that there are chains of local theory extensions:

(i) $\mathcal{T}_0 \subseteq \cdots \subseteq \mathcal{T}_S \subseteq \mathcal{T}_S \cup \mathsf{Init}$ and

(ii) $\mathcal{T}_0 \subseteq \cdots \subseteq \mathcal{T}_S \subseteq \mathcal{T}_S \cup \Psi \subseteq \mathcal{T}_S \cup \Psi \cup \mathsf{Update}$

Thus, checking satisfiability of $\mathcal{T}_S \cup \mathsf{Init} \cup \neg\Psi$ and of $\mathcal{T}_S \cup \Psi \cup \mathsf{Update} \cup \neg\Psi'$ can be reduced to a satisfiability test w.r.t. $\mathcal{T}_0$.

If $\mathcal{T}_0$ allows quantifier elimination and the formulae we obtain in $\mathcal{T}_S \cup \mathsf{Init}$ or $\mathcal{T}_S \cup \Psi \cup \mathsf{Update}$ are satisfiable, we can use the symbol elimination method described in Algorithm 2 to obtain a set Γ of $\Sigma_0 \cup \Sigma_p$-constraints such that $\mathcal{T}_S \cup \Gamma \cup \mathsf{Init} \cup \neg\Psi$ or $\mathcal{T}_S \cup \Gamma \cup \Psi \cup \mathsf{Update} \cup \neg\Psi'$ are unsatisfiable, respectively. □

By (1) in Theorem 4.5 the problem of *invariant checking* is decidable and by (2) in Theorem 4.5 the problem of *constraint synthesis* is decidable under assumptions **(A1)**, **(A2)** and **(A3)**. It remains to analyze the problem of *invariant generation*.

4.2 Invariant Strengthening Algorithm

We now study the problem of strengthening a formula Ψ in a goal-oriented way to a universally quantified inductive invariant. We propose an algorithm for invariant strengthening and prove its partial correctness under certain assumptions. We then present some refinements of the algorithm and identify situations in which the assumptions under which partial correctness is guaranteed can be weakened. We conclude this section with an analysis of termination of our method and a short description of an implementation.

The method we propose is described in Algorithm 3. As an input we assume given a transition constraint system $T = (\Sigma, \mathsf{Init}, \mathsf{Update})$. We are searching for inductive invariants

Algorithm 3 Successively strengthening a formula to an inductive invariant

Input:	Transition constraint system $T = (\Sigma, \mathbf{Init}, \mathbf{Update})$ satisfying assumptions **(A1),(A2),(A3)** and $\mathcal{T}_0$ allowing quantifier elimination; signature $\Sigma_p \subseteq \Sigma$; $\Psi \in \mathsf{LocSafe}$, formula over Σ_p.
Output:	Inductive invariant I of T that entails Ψ and contains only function symbols in Σ_p (if such an invariant exists).

1: $I := \Psi$
2: **while** I is not an inductive invariant for T **do:**
  if $\mathsf{Init} \not\models I$ **then:**
    return "no universal inductive invariant entails Ψ"
  if I is not preserved under Update **then:**
    Let Γ be obtained by eliminating all symbols not in Σ_p from $I \wedge \mathsf{Update} \wedge \neg I'$ and negating the result;
    $I := I \wedge \Gamma$
3: **return** "I is an inductive invariant which entails Ψ"

over a certain signature, which is specified by a set of parameters $\Sigma_p \subseteq \Sigma$. The starting point for the algorithm is a formula Ψ, which contains only symbols from $\Sigma_0 \cup \Sigma_p$. We call this formula a candidate invariant, as we do not know beforehand whether it is already an inductive invariant or not, nor whether it can be strengthened to obtain an inductive invariant.

The goal of the algorithm is to strengthen Ψ to an inductive invariant I (which still contains only symbols from $\Sigma_0 \cup \Sigma_p$), i.e. to find a formula I such that I entails Ψ and I is an inductive invariant. The strengthening is done iteratively, by starting with our first candidate invariant $I_0 = \Psi$, which in each step is strengthened further to a new candidate invariant $I_1, I_2, ...$ until an inductive invariant is found or it can be concluded that none exists. In each iteration of the algorithm we have to perform the following steps:

(1) **Invariant Checking:** Check whether the current candidate invariant I_n is an inductive invariant, i.e. check whether it

  a) holds in the initial state, and

  b) is preserved during updates.

  We show that failure to prove a) means that no inductive invariant which entails Ψ exists (which means that the algorithm can terminate here). Failure to prove b) means that I_n is not inductive and needs to be strengthened. If both conditions hold, then the current candidate is an inductive invariant which entails Ψ and the algorithm terminates.

(2) **Invariant Strengthening:** From the current candidate invariant I_n we compute a strengthened formula I_{n+1} which will be our next candidate invariant. We can generate a strengthened formula by applying Algorithm 2 to eliminate all the symbols not in Σ_p from the formula $I_n \cup \mathsf{Update} \cup \neg I'_n$, where I'_n results from I_n by replacing each $f \in \Sigma$ with $f' \in \Sigma'$. This formula expresses that the current candidate invariant holds before the update, but not anymore after the update. By using Algorithm 2 we compute a constraint Γ_n (if the theory extensions involved are local, then Γ_n is the weakest such constraint) which rules out this situation. Our new candidate invariant is then $I_{n+1} = I_n \wedge \Gamma_n$.

We first illustrate how the algorithm can be applied on a simple example containing formulae in $\mathsf{LI}(\mathbb{Z})$. More complex examples follow later.

Example 4.6. We consider Example 12 from [44]. Let $S = (\mathcal{T}_S, T)$, where $\mathcal{T}_S = \mathsf{LI}(\mathbb{Z})$ and $T = (\Sigma_S, \mathsf{Init}, \mathsf{Update})$ is a transition constraint system, where:

- $\Sigma_S = \Sigma_0 \cup \Sigma_p$, where
  - Σ_0 is the signature of linear integer arithmetic and
  - $\Sigma_p = \{x, y\}$, where x and y are 0-ary function symbols
- $\mathsf{Init} = (x \approx y \vee x \approx y + 2)$
- $\mathsf{Update} = (x \leq y + 1 \rightarrow x' \approx x + 2)$

Let $\Psi = (y \leq x \wedge x \leq y + 2)$ be the property that is supposed to be invariant. We apply Algorithm 3, starting with the candidate invariant $I_0 = \Psi = (y \leq x \wedge x \leq y + 2)$.

Invariant checking: We first check whether $\mathsf{Init} \models I_0$. It is easy to see that this holds. We then check whether I_0 is invariant under transitions of T. This is the case if the following formula is unsatisfiable:

$$F_0 = (y \leq x \wedge x \leq y + 2) \wedge \mathsf{Update}_x \wedge \mathsf{Update}_y \wedge (y' > x' \vee x' > y' + 2), \text{ where}$$

$$\mathsf{Update}_x = (x \leq y + 1 \rightarrow x' \approx x + 2) \wedge (x > y + 1 \rightarrow x' \approx x)$$

$$\mathsf{Update}_y = y' \approx y$$

F_0 is satisfiable, so we have to strengthen Ψ.

Invariant strengthening: We eliminate all symbols which are not in Σ_p (i.e. x' and y') from F_0 (we can do this using Redlog with context PASF) and obtain $x \approx y + 1$. We negate it and obtain $\Gamma_0 = x \not\approx y + 1$. Therefore the new candidate invariant is:

$$I_1 = I_0 \wedge \Gamma_0 = (y \leq x \wedge x \leq y + 2) \wedge (x \not\approx y + 1) \equiv x \approx y \vee x \approx y + 2$$

We check whether I_1 is an inductive invariant. I_1 clearly holds in the initial state. We check whether it is invariant under updates by checking satisfiability of the following formula:

$$F_1 = (x \approx y \vee x \approx y + 2) \wedge \mathsf{Update}_x \wedge \mathsf{Update}_y \wedge (x' \not\approx y \wedge x' \not\approx y' + 2)$$

F_1 is unsatisfiable, so $I = x \approx y \vee x \approx y + 2$ is an inductive invariant which entails $\Psi = (y \leq x \wedge x \leq y + 2)$, hence Ψ is an invariant of the system S. ■

Termination of Algorithm 3 is not guaranteed in general, as in the worst case it could happen that the algorithm ends up in an infinite loop of strengthenings. In the following we show such an example.

Example 4.7. We consider Example 2 from [44] with a slight change in the condition of the while-loop. Let $S = (\mathcal{T}_S, T)$, where $\mathcal{T}_S = \mathsf{LI}(\mathbb{Z})$ and $T = (\Sigma_S, \mathsf{Init}, \mathsf{Update})$ is a transition constraint system, where:

- $\Sigma_S = \Sigma_0 \cup \Sigma_p$, where
  - Σ_0 is the signature of linear integer arithmetic and
  - $\Sigma_p = \{x, y, z, N\}$, where x, y, z and N are 0-ary function symbols

- $\mathsf{Init} = (x \approx 0 \wedge y \approx 0 \wedge z \approx 0)$
- $\mathsf{Update} = (x \leq N \rightarrow (x' \approx x + 1 \wedge y' \approx y + 1 \wedge z' \approx z + x' - y'))$

Let $\Psi = (z \leq 0)$ be the property that is supposed to be invariant. We apply Algorithm 3.

We start with the candidate invariant $I_0 = \Psi = (z \leq 0)$.

Iteration 1:

Invariant checking: We first check whether $\mathsf{Init} \models I_0$. It is easy to see that this holds. We then check whether I_1 is invariant under transitions of T. This is the case if the following formula is unsatisfiable:

$F_0 = (z \leq 0) \wedge \mathsf{Update}_x \wedge \mathsf{Update}_y \wedge \mathsf{Update}_z \wedge (z' > 0)$, where

$\mathsf{Update}_x = (x \leq N \rightarrow x' \approx x + 1) \wedge (x > N \rightarrow x' \approx x)$

$\mathsf{Update}_y = (x \leq N \rightarrow y' \approx y + 1) \wedge (x > N \rightarrow y' \approx y)$

$\mathsf{Update}_z = (x \leq N \rightarrow z' \approx z + x' - y') \wedge (x > N \rightarrow z' \approx z)$

This formula is satisfiable, so we have to strengthen Ψ.

Invariant strengthening: We eliminate all symbols which are not in Σ_p (i.e. x', y' and z') from F (we can do this using Redlog with context PASF) and obtain

$$x \leq N \wedge x - y + z > 0 \wedge z \leq 0.$$

We negate it and obtain

$$\Gamma_0 = ((z \leq 0 \wedge x \leq N) \rightarrow z \leq y - x).$$

Therefore the new candidate invariant is:

$$\begin{aligned} I_1 = I_0 \wedge \Gamma_0 =& (z \leq 0) \wedge ((z \leq 0 \wedge x \leq N) \rightarrow z \leq y - x) \\ \equiv& (z \leq 0) \wedge (x \leq N \rightarrow z \leq y - x) \end{aligned}$$

Iteration 2:

Invariant Checking: I_2 clearly holds in the initial state. We check whether it is invariant under updates by checking satisfiability of the following formula:

$$\begin{aligned} F_1 =& ((z \leq 0) \wedge (x \leq N \rightarrow z \leq y - x)) \wedge \mathsf{Update}_x \wedge \mathsf{Update}_y \wedge \mathsf{Update}_z \wedge \\ & ((z' > 0) \vee (x' \leq N \wedge z' > y' - x')) \end{aligned}$$

F_1 is satisfiable, so I_1 has to be strengthened.

Invariant Strengthening: We eliminate x', y' and z' from F_2 and obtain

$$x + 1 \leq N \wedge 2x - 2y + z > 0 \wedge x - y + z \leq 0 \wedge z \leq 0.$$

We negate it and obtain

$$\Gamma_1 = ((z \leq 0 \wedge x + 1 \leq N \wedge z \leq y - x) \rightarrow z \leq 2y - 2x).$$

Therefore the new candidate invariant is:

$$\begin{aligned} I_2 = I_1 \wedge \Gamma_1 &= (z \leq 0) \wedge (x \leq N \rightarrow z \leq y - x) \\ &\quad \wedge ((z \leq 0 \wedge x + 1 \leq N \wedge z \leq y - x) \rightarrow z \leq 2y - 2x) \\ &\equiv (z \leq 0) \wedge (x \leq N \rightarrow z \leq y - x) \\ &\quad \wedge (x + 1 \leq N \rightarrow z \leq 2y - 2x) \end{aligned}$$

Further iterations:

I_2 is still not an inductive invariant. If we iterate the procedure further we would obtain the following:

$$\begin{aligned} I_n = (z \leq 0) &\wedge (x \leq N \rightarrow z \leq y - x) \\ &\wedge (x + 1 \leq N \rightarrow z \leq 2y - 2x) \\ &\wedge (x + 2 \leq N \rightarrow z \leq 3y - 3x) \\ &\vdots \\ &\wedge (x + (n - 1) \leq N \rightarrow z \leq ny - nx) \end{aligned}$$

This process would iterate infinitely. Therefore, the algorithm does not terminate. ■

While Algorithm 3 is not guaranteed to terminate, we can prove that if the algorithm terminates, then its output is correct.

4.2.1 Correctness

In addition to assumptions **(A1), (A2), (A3)** from Section 4.1 we now consider also the following assumptions (where $\mathcal{T}_0$ is the base theory in assumptions **(A1)**–**(A3)**):

(A4) $\mathcal{T}_0$ allows quantifier elimination (then also ground satisfiability in $\mathcal{T}_0$ is decidable).

(A5) All candidate invariants I computed in the while loop in Algorithm 3 are in $\mathsf{LocSafe}$, and all formulae in $\mathsf{LocSafe}$ are sets of clauses which define local theory extensions of $\mathcal{T}_0$ satisfying condition ($\mathsf{Comp_f}$).

In what follows we prove partial correctness of Algorithm 3 under assumptions **(A1)**–**(A5)**, i.e. we show that if it terminates, then its output is correct.

Lemma 4.8. *If Algorithm 3 terminates and returns a formula $I = \Gamma_1 \wedge ... \wedge \Gamma_n \wedge \Psi$, then I is an inductive invariant of T that entails Ψ, where Γ_i contains only function symbols in Σ_p for all $i \in \{1, ..., n\}$.*

Proof: The algorithm terminates and returns a formula I only if the loop condition of the while loop in line 2 of Algorithm 3 is false. This means that I is an inductive invariant. Since Ψ and any Γ computed in the while loop contain only function symbols in Σ_p, the formula I contains only function symbols in Σ_p. □

Lemma 4.9. *Assume Assumptions **(A1)**–**(A5)** hold. Assume that there exists a universal inductive invariant J containing only function symbols in Σ_p that entails Ψ. Then J entails every candidate invariant I generated in the while loop of Algorithm 3.*

Proof: We prove the lemma by induction on the number of iterations in which the candidate invariant I is obtained.

Induction basis:

If $i = 0$, then $I_0 = \Psi$, hence $J \models_{\mathcal{T}_S} \Psi = I_0$.

Induction hypothesis:

Assume that the property holds for the candidate invariant generated in n steps. Let I_{n+1} be generated in step $n+1$. In this case there exist candidate invariants $I_1, \ldots, I_n$ containing only function symbols in Σ_p such that:

(i) $I_0 = \Psi$;

(ii) for all $0 \leq i \leq n$, $\mathsf{Init} \models_{\mathcal{T}_S} I_i$;

(iii) for all $0 \leq i \leq n$, I_i is not an inductive invariant, i.e. $I_i \wedge \mathsf{Update} \wedge \neg I_i'$ is satisfiable and Γ_i is obtained by eliminating the primed functions and all function symbols which are not in Σ_p;

(iv) for all $0 \leq i \leq n$, $I_{i+1} = I_i \wedge \Gamma_i$.

Induction step:

We prove that $J \models_{\mathcal{T}_S} I_{n+1}$, i.e. that $J \models_{\mathcal{T}_S} I_n \wedge \Gamma_n$. By the induction hypothesis it holds that $J \models_{\mathcal{T}_S} I_n$. It is easy to see that then $J \equiv_{\mathcal{T}_S} J \wedge I_n$ (every $\mathcal{T}_S$-model which is a model of J is a model of $J \wedge I_n$; the converse implication is obvious). We know that J is an inductive invariant, i.e. $J \wedge \mathsf{Update} \wedge \neg J'$ is unsatisfiable. This means (if we use the equivalence $J \equiv_{\mathcal{T}_S} J \wedge I_n$) that $(J \wedge I_n) \wedge \mathsf{Update} \wedge (\neg J' \vee \neg I_n')$ is unsatisfiable. Hence, in particular $J \wedge I_n \wedge \mathsf{Update} \wedge \neg I_n'$ is unsatisfiable. By Theorem 3.5 and the way Γ_n is constructed, we know that Γ_n is the weakest universal formula over Σ_p such that $\Gamma_n \wedge I_n \wedge \mathsf{Update} \wedge \neg I_n'$ is unsatisfiable. Together with the fact that J is a universal formula containing only function symbols in Σ_p, we then know that $J \models_{\mathcal{T}_S} \Gamma_n$. Thus, $J \models_{\mathcal{T}_S} I_n \wedge \Gamma_n$, so $J \models_{\mathcal{T}_S} I_{n+1}$. □

Theorem 4.10 (Partial Correctness). *Under Assumptions **(A1)–(A5)**, if Algorithm 3 terminates, then its output is correct.*

Proof: There are two different ways in which the algorithm can terminate. We make a case distinction:

Case 1: Algorithm terminates with output "I is an inductive invariant which entails Ψ"

By Lemma 4.8, if the algorithm returns a formula I, then it is an inductive invariant. Thus, the output of the algorithm is correct.

Case 2: Algorithm terminates with output "no universal inductive invariant entails Ψ"

Assume that there exists a universal inductive invariant J that entails Ψ. By Lemma 4.9, J entails every candidate invariant generated at each iteration, thus entails I. On the one hand, by the termination condition $\mathsf{Init} \not\models I$ there exists a model $\mathcal{A}$ of Init which is not a model of I. On the other hand, every model of Init is a model of J, hence also a model of I. This is a contradiction. Therefore, the assumption that there exists a universal inductive invariant J that entails Ψ was false. Thus, the output of the algorithm is correct. □

4.3 Refinements

In this section we take a look at possible drawbacks of Algorithm 3 and propose several refinements which improve certain aspects of the algorithm, like the complexity of the quantifier elimination, the assumptions being made to guarantee correctness, or the termination.

The invariant generation algorithm may run multiple iterations of strengthening to compute an inductive invariant. With each iteration needed the formula on which quantifier elimination is applied gets larger, which could potentially lead to problems. Therefore, we examine possibilities for applying quantifier elimination on shorter formulae (for instance using the improvement of Algorithm 2 described in Section 3.2.1) and analyze the complexity.

The assumptions **(A1)**–**(A5)** which we stated to guarantee the correctness of the algorithm are quite strong. We analyze situations in which some of the assumptions can be relaxed or avoided.

Last but not least, a drawback of the algorithm is the fact that termination is not guaranteed. We show that the choice of the symbols to be eliminated can sometimes have an influence on the termination of the method and propose a refinement based on trying different sets of symbols to be eliminated. In addition to that we analyze situations in which termination is guaranteed.

4.3.1 Applying Quantifier Elimination on Shorter Formulae

Let $T = (\Sigma, \mathsf{Init}, \mathsf{Update})$ be a transition constraint system. In what follows we assume that $\mathsf{Update} = \bigvee_{f \in F} \mathsf{Update}_f$, where $F \subseteq \Sigma$ (no f' with $f \in F$ is a parameter) such that Update_f satisfies the conditions in Assumption (**A3**). We analyze the computations described in Algorithm 3, in Step 2 of iteration $n+1$.

Consider the case in which $\mathsf{Init} \models I_n$, but I_n is not invariant under updates. By construction, $I_n = \Psi \wedge \Gamma_0 \wedge \cdots \wedge \Gamma_{n-1}$, where Ψ is the safety property we consider and $\Gamma_0, \ldots, \Gamma_{n-1}$ are the additional constraints added in the first n iterations in Step 2 to strengthen the candidate invariants $I_0 = \Psi, I_1, \ldots, I_{n-1}$, respectively.

Lemma 4.11. *If $I_n = I_{n-1} \wedge \Gamma_{n-1}$ is not invariant under updates, then Algorithm 3 computes a formula $\Gamma_n = \bigwedge_{f \in F} \Gamma_n^f$, where Γ_n^f is obtained by applying Algorithm 2 to $\mathcal{K} \wedge I_n \wedge \mathsf{Update}_f \wedge G$, where G is obtained by Skolemization from $\neg \Gamma_n'$.*

Proof: $I_n \wedge \mathsf{Update} \wedge \neg I_n' \equiv \bigvee_{f \in F} (I_n \wedge \mathsf{Update}_f \wedge \neg I_n')$, so it is satisfiable if and only if for some $f \in F$, the formula $\mathcal{K} \wedge I_n \wedge \mathsf{Update}_f \wedge \neg I_n'$ is satisfiable. We have:

$$\begin{aligned} \mathcal{K} \wedge I_n \wedge \mathsf{Update}_f \wedge \neg I_n' &= \mathcal{K} \wedge (I_{n-1} \wedge \Gamma_{n-1}) \wedge \mathsf{Update}_f \wedge (\neg I_{n-1}' \vee \neg \Gamma_{n-1}') \\ &\equiv \mathcal{K} \wedge (I_{n-1} \wedge \Gamma_{n-1}) \wedge \mathsf{Update}_f \wedge \neg \Gamma_{n-1}' \end{aligned}$$

since Γ_{n-1} was introduced such that $\mathcal{K} \wedge (I_{n-1} \wedge \Gamma_{n-1}) \wedge \mathsf{Update}_f \wedge \neg I_{n-1}'$ is unsatisfiable. Then in Algorithm 3 a formula $\Gamma_n = \bigwedge_{f \in F} \Gamma_n^f$ is computed, where Γ_n^f are the (weakest) formulae obtained by Algorithm 2 such that $\mathcal{K} \wedge I_n \wedge \Gamma_n^f \wedge \mathsf{Update}_f \wedge \neg \Gamma_n'$ is unsatisfiable. □

The following Lemma expresses the fact that case distinctions of the form $\bigwedge_{i=1}^{n}(\phi_i \to C_i)$ which are mutually exclusive (i.e. $\phi_i \wedge \phi_j \models_{\mathcal{T}} \bot$ for all $1 \leq i < j \leq n$) and exhaustive (i.e. $\models_{\mathcal{T}} \bigvee_{i=1}^{n} \phi_i$) can be written in the form $\bigvee_{i=1}^{n}(\phi_i \wedge C_i)$.

Lemma 4.12. *Let $\mathcal{T}$ be a theory and for $i \in \{1, ..., n\}$ let ϕ_i be formulae and C_i be clauses. If $\phi_i \wedge \phi_j \models_{\mathcal{T}} \bot$ for all $1 \leq i < j \leq n$ and $\models_{\mathcal{T}} \bigvee_{i=1}^{n} \phi_i$, then the following equivalence holds:*

$$\bigwedge_{i=1}^{n}(\phi_i \to C_i) \equiv \bigvee_{i=1}^{n}(\phi_i \wedge C_i).$$

We now analyze the formulae Γ_n^f generated at iteration n. For simplicity we assume that f is unary; the extension to higher arities is also possible. For the following considerations we need to recall the improvement of Algorithm 2 described in Theorem 3.7. The idea is that if the set of clauses $\mathcal{K}$ can be structured as $\mathcal{K} = \mathcal{K}_p \cup \mathcal{K}_1$, where $\mathcal{K}_p$ contains only parameters (i.e. symbols which are not supposed to be eliminated), then symbol elimination only needs to be applied to $\mathcal{K}_1$. The following theorem describes how this improvement can be used in Step 2 of Algorithm 3.

Theorem 4.13. *Let $\Psi \in \mathsf{LocSafe}$ and $T = (\Sigma, \mathsf{Init}, \mathsf{Update})$ be a transition constraint system, where $\mathsf{Update} = \bigvee_{f \in F} \mathsf{Update}_f$ is of the form discussed above. Assume that the clauses in Ψ and Update_f are flat and linear for all $f \in F$. Let m be the maximal number of variables in a clause in Ψ. Assume that the only non-parametric functions which need to be eliminated are the primed symbols $\{f' \mid f \in F\}$ and that conditions (**A1**)–(**A5**) hold. Consider a variant of Algorithm 3 which uses for symbol elimination Algorithm 2 with the improvement in Theorem 3.7. Then for every step n the following hold:*

(i) The clauses in the candidate invariant I_n obtained at step n of Algorithm 3 are flat.

(ii) The number of universally quantified variables in every clause in I_n is $\leq m$.

Proof: We prove (i) and (ii) by induction over the number of iterations n.

Induction basis:

For $n = 0$ we have $I_0 = \Psi$ and (i) and (ii) clearly hold.

Induction hypothesis:

Assume that (i) and (ii) hold for iteration n.

Induction step:

In the following we prove that (i) and (ii) hold for iteration $n + 1$.

By Theorem 3.7, if the clauses $\mathcal{K} \cup I_n$ contain only function symbols in $\Sigma_0 \cup \Sigma_p$, we need to apply Algorithm 2 to $\mathsf{Update}_f \wedge G$ only, where G is obtained from $\neg\Gamma_n'$ after Skolemization. If Γ_n' is a conjunction of clauses, then G is a disjunction of conjunctions of literals. Each disjunct can be processed separately and we take the conjuction of the obtained constraints. Thus, we can assume without loss of generality that G is a conjunction of literals. By the induction hypothesis the number k of universally quantified variables in Γ_n is $\leq m$, so G contains Skolem constants $\{d_1, ..., d_k, c_1, ..., c_r\}$ with $k + r \leq m$, where $d_1, ..., d_k$ occur below f'.

To prepare the formula for symbol elimination we compute $G_1 = \mathsf{Update}_f[G] \cup G$ (with $\mathsf{est}(\mathrm{G}) = \{f'(d_1), \ldots, f'(d_k)\}$ used in the instantiation), then instantiate also the terms starting with function symbols $g \in \Sigma_p \cup \Sigma$. Hence, we use the set of ground terms

$$T = \mathsf{est}(G) \cup \mathsf{est}(G_1) = \{f'(d_1), \ldots, f'(d_k)\} \cup \{g(c) \mid g \in \Sigma_p \cup \Sigma, g(c) \in \mathsf{est}(G_1)\},$$

which is a set of flat terms, in which we isolated the terms starting with the function symbol f. Let n_f be the number of cases in the definition of f. By Lemma 4.12 we have:

$$\begin{aligned}\mathsf{Update}_f[G] &:= \bigwedge_{j=1}^{k} \left(\bigwedge_{i=1}^{n_f} (\phi_i(d_j) \to C_i(d_j, f'(d_j))) \right) \equiv \bigwedge_{j=1}^{k} \bigvee_{i=1}^{n_f} (\phi_i(d_j) \wedge C_i(d_j, f'(d_j))) \\ &\equiv \bigvee_{i_1,\ldots,i_k \in \{1,\ldots,n_f\}} \left(\bigwedge_{p=1}^{k} \phi_{i_p}(d_p) \wedge \bigwedge_{p=1}^{k} C_{i_p}(d_p, f'(d_p)) \right)\end{aligned}$$

We thus obtained a DNF with $(n_f)^k \leq (n_f)^m$ disjuncts, where n_f and m are constants depending on the description of the transition system. Both n_f and m are typically small, in most cases the definition of f' is done by at most 3 disjoint cases, so $n_f \leq 3$. Algorithm 2 is applied as follows:

Step 1: We introduce a constant $c_{f'd}$ for every term $f'(d) \in \mathsf{est}(G)$, replace $f'(d)$ with $c_{f'd}$, and add the corresponding instances of the congruence axioms. We obtain the following:

$$G_1 \quad = \quad G_0 \wedge \quad (\mathsf{Update}_f[G])_0 \wedge \bigwedge_{f'(d_i), f'(d_j) \in \mathsf{est}(G)} d_i \approx d_j \to c_{f'd_i} \approx c_{f'd_j}$$

We may compute a disjunctive normal form for the instances of congruence axioms or not; depending on this we obtain one of the equivalences (4.1) or (4.2) below:

$$G_1 \quad \equiv G_0 \wedge \bigvee_{\substack{i_1,\ldots,i_k \in \{1,\ldots,n_f\} \\ D \subseteq \mathsf{est}(G)^2}} \left(\bigwedge_{p=1}^{k} \phi_{i_p}(d_p) \wedge \bigwedge_{p=1}^{k} C_{i_p}(d_p, c_{f'd_p}) \wedge K_D(\mathsf{Con}_0) \right) \tag{4.1}$$

$$\equiv G_0 \wedge \bigvee_{i_1,\ldots,i_k \in \{1,\ldots,n_f\}} \left(\bigwedge_{p=1}^{k} \phi_{i_p}(d_p) \wedge \bigwedge_{p=1}^{k} C_{i_p}(d_p, c_{f'd_p}) \wedge \mathsf{Con}_0 \right) \tag{4.2}$$

In equivalence (4.1) we brought the formula to DNF. The formulae $K_D(\mathsf{Con}_0)$ are the conjunctions which appear when bringing the congruence axioms to DNF, based on a subset D of $\mathsf{est}(G)^2$. In equivalence (4.2) we used distributivity and moved the conjunction of all congruence axioms inside the conjunctive formulae in the big disjunction.

Note that

$$\mathsf{Con}_0 = \bigwedge_{f(d_i), f(d_j) \in \mathsf{est}(G)} (d_i \approx d_j \to c_{f'd_i} \approx c_{f'd_j})$$

and every formula

$$K_D(\mathsf{Con}_0) = \bigwedge_{(f(d_i), f(d_j)) \in D} d_i \not\approx d_j \wedge \bigwedge_{(f'(d_1), f'(d_2)) \in \mathsf{est}(G)^2 \setminus D} c_{f'd_1} \approx c_{f'd_2}$$

contain $k^2 \leq m^2$ conjunctions, and the DNF of Con_0 contains $2^{(k^2)} \leq 2^{(m^2)}$ disjuncts, each being of length k.

In a second reduction we replace every term of the form $g(d) \in \mathsf{est}(G_1)$, where $g \in \Sigma_p$, with a new constant c_{gd}. By Theorem 3.7 we do not need to add the corresponding congruence axioms. Using equivalence (4.1) and distributivity we obtain

$$G^0_{1(1)} \equiv \bigvee_{\substack{i_1,\ldots,i_k \in \{1,\ldots,n_f\} \\ D \subseteq \mathsf{est}(G)^2}} \left(\bigwedge_{p=1}^{k} \phi^0_{i_p}(d_p) \wedge \bigwedge_{p=1}^{k} C^0_{i_p}(d_p, c_{f'd_p}) \wedge K_D(\mathsf{Con}_0) \wedge G^0_0(\overline{d}, \overline{c}, \overline{c}_{f'd}, \overline{c}_{gc}) \right)$$

and using equivalence (4.2) and distributivity we obtain

$$G^0_{1(2)} \equiv \bigvee_{i_1,\ldots,i_k \in \{1,\ldots,n_f\}} \left(\bigwedge_{p=1}^{k} \phi^0_{i_p}(d_p) \wedge \bigwedge_{p=1}^{k} C^0_{i_p}(d_p, c_{f'd_p}) \wedge \mathsf{Con}_0 \wedge G^0_0(\overline{d}, \overline{c}, \overline{c}_{f'd}, \overline{c}_{gc}) \right)$$

where ϕ^0_j, C^0_j and G^0_0 are obtained from ϕ_j, C_j and G_0, respectively, after replacing all terms of the form $g(c)$, where $g \in \Sigma_p$, with c_{gc}.

Step 2: f' is not a parameter. All the other function symbols are either parametric or in $\mathcal{T}_0$. We therefore replace the constants $c_{f'd}$ with variables $x_{f'd}$.

Step 3: Note that $\exists x_{f'd_1}, \ldots, x_{f'd_k}\ G^0_{1(1)}(x_{f'd_1}, \ldots, x_{f'd_k})$ is equivalent to

$$G'_{1(1)} \equiv \bigvee_{\substack{i_1,\ldots,i_k \in \{1,\ldots,n_f\} \\ D \subseteq \mathsf{est}(G)^2}} \left(\bigwedge_{p=1}^{k} \phi^0_{i_p}(d_p) \wedge \bigwedge_{(f'(d_i), f'(d_j)) \in D} d_i \not\approx d_j \wedge \right.$$
$$\left. \exists x_{f'd_1}, \ldots, x_{f'd_k} \left(\bigwedge_{p=1}^{k} C^0_{i_p}(d_p, x_{f'd_p}) \wedge \bigwedge_{(f'(d_i), f'(d_j)) \in \mathsf{est}(G)^2 \setminus D} x_{f'd_i} \approx x_{f'd_j} \wedge G^0_0(\overline{d}, \overline{c}, \overline{c}_{gc}, \overline{x}_{f'd}) \right) \right).$$

Using equivalence (4.2) we obtain the following:

$$G'_{1(2)} \equiv \bigvee_{i_1,\ldots,i_k \in \{1,\ldots,n_f\}} \left(\bigwedge_{p=1}^{k} \phi^0_{i_p}(d_p) \wedge \exists x_{f'd_1}, \ldots x_{f'd_n} \left(\bigwedge_{p=1}^{k} C^0_{i_p}(d_p, x_{f'd_p}) \wedge \right. \right.$$
$$\left. \left. \bigwedge_{(f'(d_i), f('d_j)) \in \mathsf{est}(G)} (d_i \approx d_j \rightarrow x_{f'd_i} \approx x_{f'd_j}) \wedge G^0_0(\overline{d}, \overline{c}, \overline{c}_{gc}, \overline{x}_{f'd}) \right) \right)$$

Using equivalence (4.1) we obtain after quantifier elimination a formula of the form

$$G''_{1(1)} \equiv \bigvee_{\substack{i_1,\ldots,i_k \in \{1,\ldots,n_f\} \\ D \subseteq \mathsf{est}(G)^2}} \left(\bigwedge_{p=1}^{k} \phi^0_{i_p}(d_p) \wedge \bigwedge_{(d_i,d_j) \in D} d_i \not\approx d_j \wedge D^1_{i_1,\ldots,i_k,D}(\overline{d}, \overline{c}, \overline{c}_{gc}) \right)$$

where $D^1_{i_1,\ldots,i_k,D}(\overline{d}, \overline{c}, \overline{c}_{gc})$ is obtained after quantifier elimination from

$$\exists x_{f'd_1}, \ldots, x_{f'd_k} \left(\bigwedge_{p=1}^{k} C^0_{i_p}(d_p, x_{f'd_p}) \wedge \bigwedge_{(f'(d_i), f'(d_j)) \in \mathsf{est}(G)^2 \setminus D} x_{f'd_i} \approx x_{f'd_j} \wedge G^0_0(\overline{d}, \overline{c}, \overline{c}_{gc}, \overline{x}_{f'd}) \right).$$

When we use equivalence (4.2) we obtain after eliminating the quantifiers a formula of the form

$$G''_{1(2)} \equiv \bigvee_{i_1,\ldots,i_k \in \{1,\ldots,n_f\}} \left(\bigwedge_{p=1}^{k} \phi^0_{i_p}(d_p) \wedge D^2_{i_1,\ldots,i_k,D}(\overline{d}, \overline{c}, \overline{c}_{gc}) \right)$$

where $D^2_{i_1,\ldots,i_k,D}(\overline{d}, \overline{c}, \overline{c}_{gc})$ is obtained after quantifier elimination form

$$\exists x_{f'd_1}, \ldots, x_{f'd_k} \left(\bigwedge_{p=1}^{k} C^0_{i_p}(d_p, x_{f'd_p}) \wedge \bigwedge_{(f'(d_i), f'(d_j)) \in \mathsf{est}(G)} (d_i \approx d_j \rightarrow x_{f'd_i} \approx x_{f'd_j}) \wedge G^0_0(\overline{d}, \overline{c}, \overline{c}_{gc}, \overline{x}_{f'd}) \right).$$

Step 4: We replace back in the formula obtained this way all constants c_{gd}, where $g \in \Sigma_p$ and $g(c) \in \mathsf{est}(G_1)$, with the terms $g(c)$. Therefore, when starting from formula (4.1) we obtain

$$\Gamma_{2(1)}(\overline{d}, \overline{c}) = \bigvee_{\substack{i_1,\ldots,i_k \in \{1,\ldots,n_f\} \\ D \subseteq \mathsf{est}(G)^2}} \left(\left(\bigwedge_{p=1}^{k} \phi_{i_p}(d_p) \wedge \bigwedge_{(d_i,d_j) \in D} d_i \not\approx d_j \wedge G^g_0(\overline{d}, \overline{c}, \overline{g}(c)) \right) \wedge D^1_{i_1,\ldots,i_k,D}(\overline{d}, \overline{c}, \overline{g}(c)) \right)$$

and when starting from formula (4.2) we obtain

$$\Gamma_{2(2)}(\overline{d}, \overline{c}) = \bigvee_{i_1,\ldots,i_k \in \{1,\ldots,n_f\}} \bigvee_{D \subseteq \mathsf{est}(G)^2} \left(\left(\bigwedge_{p=1}^{k} \phi_{i_p}(d_p) \wedge G^g_0(\overline{d}, \overline{c}, \overline{g}(c)) \right) \wedge D^2_{i_1,\ldots,i_k,D}(\overline{d}, \overline{c}, \overline{g}(c)) \right).$$

Step 5: We negate $\exists \overline{y} \Gamma_{2(1)}(\overline{y})$ and obtain

$$\forall \overline{y} \left[\bigwedge_{i_1,\ldots,i_k \in \{1,\ldots,n_f\}} \bigwedge_{D \subseteq \mathsf{est}(G)^2} \left(\neg \left(\bigwedge_{p=1}^{k} \phi_{i_p}(y_p) \wedge \bigwedge_{(f(d_i), f(d_j)) \in D} y_i \not\approx y_j \wedge G^g_0(\overline{y}, \overline{g}(y)) \right) \wedge \neg D^1_{i_1,\ldots,i_k,D}(\overline{y}, \overline{g}(y)) \right) \right]$$

which is equivalent to

$$\forall \overline{y} \left[\bigwedge_{i_1,\ldots,i_k \in \{1,\ldots,n_f\}} \bigwedge_{D \subseteq \mathsf{est}(G)^2} \left(\left(\bigwedge_{p=1}^{k} \phi_{i_p}(y_p) \wedge \bigwedge_{(f(d_i), f(d_j)) \in D} y_i \not\approx y_j \wedge G^g_0(\overline{y}, \overline{g}(y)) \right) \rightarrow \neg D^1_{i_1,\ldots,i_k,D}(\overline{y}, \overline{g}(y)) \right) \right].$$

We have therefore obtained a formula $\Gamma^f_{n+1}(G) \equiv \neg \exists \overline{y} \Gamma_{2(1)}(\overline{y})$, which is a conjunction of universally quantified clauses. All clauses in $\Gamma^f_{n+1}(G)$ are flat. By construction, the number of universally quantified variables in $\Gamma^f_{n+1}(G)$ is $k + r \leq m$.

The case of $\exists \overline{y} \Gamma_{2(2)}(\overline{y})$ is similar. □

Complexity: We analyze the complexity of the transformations presented above. We show how the size of the formula changes depending on n_f and m (since $k \leq m$ and m is a constant of the system).

- After the instantiation we first have a conjunction of $k * n_f$ implications (where $k * n_f \leq m * n_f$).
- After the transformation to DNF we obtain a disjunction of at most n_f^m conjunctions, each having a length of at most $m * l_f$, where l_f is the maximal length of a rule in Update_f.
- Using formula (4.1), after applying the first reduction in Step 1 we obtain a disjunction of $n_f^m * 2^{(m^2)}$ conjunctions, each of length at most $(m * l_f + m^2 + |G|)$ (note that $m * l_f$ and m^2 are constants of the system).
- Using formula (4.2), after applying the first reduction in Step 1 we obtain a disjunction of n_f formulae, each of length at most $(m * l_f + |G| + |\mathsf{Con}_0|)$ (note that $m * l_f$ is a constant of the system).
- In Step 2 the size of the formulae does not change.
- After quantifier elimination in Step 3 the size of the conjuncts may grow, since for every variable which is eliminated, the size of the formula might grow quadratically.
- The last two steps of the algorithm do not change the size of the formula.

In the following we analyze the length of the formula Γ_n, i.e. the constraint generated by symbol elimination in the n-th iteration of Algorithm 3.

Theorem 4.14. *Let $\Psi \in \mathsf{LocSafe}$ and $T = (\Sigma, \mathsf{Init}, \mathsf{Update})$ be a transition constraint system, where $\mathsf{Update} = \bigvee_{f \in F} \mathsf{Update}_f$, with $F \subseteq \Sigma$ such that Update_f satisfies **(A3)**. The following hold:*

(1) If the constraints C_i are all equalities, then the length of Γ_n is at most $n * k_1 + |\Psi|$, where k_1 is a constant of the system.

(2) If the constraints C_i are linear constraints over $\mathbb{Q}$ or $\mathbb{R}$, then the length of the clauses in Γ_n is of order $O(|\Psi|)^{n*k_2}$, where k_2 is a constant of the system.

Proof: Let m be the maximal number of variables in a clause in Ψ and l_f be the maximal length of a rule in Update_f. Let n_f be the number of cases in the definition of f and mc be the maximal number of case distinctions in the updates Update_f for $f \in F$ (i.e. $n_f \leq \mathsf{mc}$ for all $f \in F$).

We first analyze the number $c(n)$ of clauses in Γ_n. We have $\Gamma_n = \bigwedge_{f \in F} \Gamma_n^f$. From the proof of Theorem 4.13 it can be seen that for every clause C in Γ_n, the number of clauses in Γ_{n+1}^f generated to ensure unsatisfiability of $\mathsf{Update}_f \wedge G$, where $G = \neg C$, is at most $n_f^m * m^2$, where $n_f^m * m^2 \leq \mathsf{mc}^m * m^2$. Thus, for the number $c(n+1)$ of clauses in Γ_{n+1} the following holds:

$$c(n+1) \leq |F| * \mathsf{mc}^m * m^2 * c(n) = k_1 * c(n)$$

This shows that $c(n+1) \leq k_1^n * c(1)$, where $k_1 = |F| * \mathsf{mc}^m * m^2$ is a constant of the system.

We now analyze the length of the clauses in Γ_n. Let q be the maximal length (i.e. the maximal number of literals) of the constraints C_i and $q(n)$ the maximal length of the clauses in Γ_n. Then the negation of every clause in Γ_n contains at most $q(n)$ literals. After instantiation and transformation to disjunctive normal form we obtain a disjunction of conjunctions of the form

$$\bigwedge_{p=1}^{k} \phi_{i_p}(d_p) \wedge \bigwedge_{p=1}^{k} C_{i_p}(d_p, c_{f'd_p}) \wedge \bigwedge_{f'(d_i),f'(d_j))\in D} d_i \not\approx d_j \wedge \bigwedge_{(f'(d_i),f'(d_j))\in \mathsf{est}(G)^2\setminus D} c_{f'd_i} \approx c_{f'd_j} \wedge G_0$$

(where $D \subseteq \mathsf{est}(G)^2$) with at most $(k * l_f + k^2 + |G|) \leq (m * l_f + m^2 + p(n))$ literals. We apply quantifier elimination essentially on $\bigwedge_{p=1}^{k} C_{i_p}(d_p, x_{fd_p}) \wedge G$ (the conjunction of equalities $x_{f'd_i} \approx x_{f'd_j}$ is processed fast). This conjunction contains at most a number $k * p + p(n) \leq m * p + p(n)$ of literals.

If the constraints C_i are equalities (as it is the case with updates of the form D_f from Example 2.21), then the formula obtained by quantifier elimination has equal length or is shorter than $\bigwedge_{p=1}^{k} C_{i_p}(d_p, x_{fd_p}) \wedge G$, so $p(n+1) \leq m * p + p(n)$. Thus, $p(n) \leq k_2 * n + |\Psi|$, where $k_2 = m * p$.

If the constraints are conjunctions of linear inequalities over $\mathbb{Q}$, then using for example the Fourier-Motzkin elimination procedure, after eliminating one variable the size is at most $(m * p + p(n))^2$, and after eliminating m variables it is $(m * p + p(n))^{2^m}$. Thus, $p(n+1) \leq (k_2 + p(n))^{k_3}$, where $k_3 = 2^m$ is a constant depending on the system. Hence, $p(n)$ is in the worst case in $O(p(1)^{k_3^n})$. □

Remark 4.15. *In many of the examples we analyzed the growth of the formulae is not so dramatic for the following reasons:*

(i) In many cases the updates are assignments.

(ii) Even if the updates are specified by giving lower and upper bounds for the new values, the length of the constraints C_i is 1 or 2.

(iii) Many of the disjuncts in the DNF to which quantifier elimination should be applied are unsatisfiable and do not need further consideration.

If there are non-parametric functions that are being updated, then the number of variables in the clauses Γ_n might grow: Any constant $c \in F$ which is not a parameter, but occurs below a parameter in Update or G, is then being converted into a universally quantified variable by Algorithm 2, as the following example shows.

Example 4.16. Consider the program in Figure 4.1 (this is the same program that was illustrated in Chapter 1).

It uses two subprograms $\mathsf{copy}(a, b)$ and $\mathsf{add1}(a)$ with the following meaning:

- $\mathsf{copy}(a, b)$ copies array b into array a;
- $\mathsf{add1}(a)$ adds 1 to every element of array a.

Assume that we are in linear arithmetic over the integers and assume that b is an array which is increasingly sorted. The goal is to prove that the formula $\Psi := d_2 \geq d_1$ is an invariant of the program.

```
d1 = 1;
d2 = 1;
copy(a, b);
i = 0;
while (nondet())
{
   a = add1(a);
   d1 = a[i];
   d2 = a[i+1];
   i = i + 1
}
```

Figure 4.1: A simple program.

We can model the program as a system $S = (\mathcal{T}_S, T)$, where $\mathcal{T}_S = \mathsf{LI}(\mathbb{Z})$ is the background theory and $T = (\Sigma \cup \Sigma_p, \mathsf{Init}, \{\mathsf{Update}_a, \mathsf{Update}_{d_1}, \mathsf{Update}_{d_2}, \mathsf{Update}_i\})$ is a transition constraint system, where:

- $\Sigma = \{b, i\}$, where b is a unary function and i is a variable
- $\Sigma_p = \{a, d_1, d_2\}$, where a is a unary function and d_1, d_2 are variables
- $\mathsf{Init} = \{d_1 \approx 1, d_2 \approx 1, \forall j(a[j] \approx b[j]), i \approx 0, \forall j, k(j \leq k \rightarrow b[j] \leq b[k])\}$
- $\mathsf{Update}_a = \forall j(a'[j] \approx a[j] + 1)$
- $\mathsf{Update}_{d_1} = d_1' \approx a'[i]$
- $\mathsf{Update}_{d_2} = d_2' \approx a'[i+1]$
- $\mathsf{Update}_i = i' \approx i + 1$

The task is to prove that if b is an array with its elements sorted in increasing order, then the formula $\Psi \coloneqq d_1 \leq d_2$ is an invariant of the program.

Invariant checking: Ψ clearly holds in the initial state, as $d_1 \approx 1 \leq 1 \approx d_2$. In order to show that Ψ is an inductive invariant of the while loop, we need to prove that the following formula is unsatisfiable:

$$d_1 \leq d_2 \wedge \forall j(a'[j] \approx a[j] + 1) \wedge d_1' \approx a'[i] \wedge d_2' \approx a'[i+1] \wedge i' \approx i + 1 \wedge d_1' > d_2'$$

We have the chain of local theory extensions

$$\mathsf{LI}(\mathbb{Z}) \quad \subseteq \quad \mathsf{LI}(\mathbb{Z}) \cup \mathsf{UIF}_a \quad \subseteq \quad \mathsf{LI}(\mathbb{Z}) \cup \mathsf{UIF}_a \cup \mathsf{Update}_a = \mathcal{T},$$

where $\mathsf{Update}_a = \forall j(a'[j] \approx a[j]+1)$. Checking unsatisfiability of the formula above can be done using the hierarchical reduction method for local theory extensions. First we compute $\mathsf{Update}_a[G]$ and obtain the set of instances $\{a'[i] \approx a[i] + 1,\ a'[i+1] \approx a[i+1] + 1\}$. After purification and after eliminating the variables d_1', d_2' and i' (by replacing every occurrence of the variables by the right hand side of their respective update rule) we obtain the following:

$$\begin{array}{ll} \mathsf{Def}: & a_1' \approx a'[i] \wedge a_2' \approx a'[i+1] \\ G_0 \wedge (\mathsf{Update}_a)_0: & d_1 \leq d_2 \wedge a_1' > a_2' \ \wedge \ a_1' \approx a[i] + 1 \ \wedge \ a_2' \approx a[i+1] + 1 \end{array}$$

In a second step we can use a similar hierarchical reduction for the extension with the uninterpreted function a. We obtain:

$$\begin{array}{lllll} \mathsf{Def}: & a_1' \approx a'[i] \wedge a_2' \approx a'[i+1] & \wedge & a_1 \approx a[i] \wedge a_2 = a[i+1] \\ G_0 \wedge \mathsf{Update}(a')_0: & d_1 \leq d_2 \wedge a_1 + 1 > a_2 + 1 & \wedge & a_1' \approx a_1 + 1 \wedge a_2' \approx a_2 + 1 \end{array}$$

It can be checked that $G_0 \wedge \mathsf{Update}(a')_0$ is satisfiable, i.e. I_0 is not an inductive invariant and therefore needs to be strengthened.

Invariant strengthening:

The only symbols which need to be eliminated from $G_0 \wedge \mathsf{Update}(a')_0$ are the primed variables a_1' and a_2'. For this we use Algorithm 2. By the improvement in Theorem 3.7 we can ignore $I_0 = d_1 \leq d_2$, as $d_1, d_2 \in \Sigma_p$. After quantifier elimination we obtain the formula $a_1 > a_2$. After replacing the constants with the terms they denote we obtain $\exists i(a[i] > a[i+1])$. The negation of this formula yields the following constraint:

$$\Gamma = \forall j(a[j] \leq a[j+1])$$

Therefore we obtain the following strengthened candidate invariant:

$$I_1 = I_0 \wedge \Gamma = d_1 \leq d_2 \wedge \forall j(a[j] \leq a[j+1])$$

We check whether the candidate invariant I_1 is an inductive invariant. I_1 holds in the initial state, as $d_1 \leq d_2$ holds due to $d_1 \approx 1 \approx d_2$ and $\forall j(a[j] \leq a[j+1])$ holds because array b is increasingly sorted and b is copied into array a, so a is increasingly sorted as well. It can also be checked that I_1 is invariant under updates, i.e. the formula

$$\begin{array}{r} (d_1 \leq d_2 \wedge \forall j(a[j] \leq a[j+1]) \wedge \forall j(a'[j] \approx a[j] + 1) \wedge d_1' \approx a'[i] \wedge d_2' \approx a'[i+1] \\ \wedge\, i' \approx i + 1 \wedge (d_1' > d_2' \vee \exists j(a'[j] > a'[j+1])) \end{array}$$

is unsatisfiable. Thus, $I_1 = d_1 \leq d_2 \wedge \forall j(a[j] \leq a[j+1])$ is an inductive invariant of the system S, i.e. of the program in Figure 4.1. ■

4.3.2 Avoiding Some Conditions

In order to prove correctness of Algorithm 3 we made assumption **(A1)**–**(A5)** from Sections 4.1 and 4.2. We analyze situations in which some of these assumptions can be relaxed.

Assume that only primed variables are to be eliminated, i.e. all function symbols are parameters ($\Sigma_p = \Sigma$). The condition that $\mathcal{T}_0$ allows quantifier elimination in Assumption **(A4)** is not needed if all update axioms are of the form ($\mathsf{D}_{f'}$) in Example 2.23 (i.e. the value of f' is defined using equality), because then the primed variables can be easily eliminated.

Assumption **(A5)** is very strong. Even if we cannot guarantee that Assumption **(A5)** holds, it could theoretically be possible to identify situations in which we can transform candidate invariants which do not define local extensions into equivalent formulae which define local extensions – for instance using the results in [62]. As an example of such a transformation, consider the base theory $\mathcal{T}_0 = \mathsf{LI}(\mathbb{Z})$ extended with a function f satisfying the following property:

$$\mathcal{K} = \forall x(f(x) \leq f(x+1))$$

This extension is not local, as we cannot complete every partial model to a total one: If we have the instances $f(1) \leq f(2)$ and $f(4) \leq f(5)$, then for example $f(1) \approx f(2) \approx 1$ and $f(4) \approx f(5) \approx 0$ satisfies the instances and is a partial model of $\mathcal{K}$, but it cannot be transformed into a total model of $\mathcal{K}$. However, we can write the monotonicity property described by $\mathcal{K}$ equivalently in the following way:

$$\mathcal{K}' = \forall x, y(x \leq y \ \rightarrow f(x) \leq f(y))$$

According to Example 2.21, $\mathcal{T}_0 \subseteq \mathcal{T}_0 \cup \mathcal{K}'$ is a local theory extension satisfying (Comp_f).

If we can guarantee that the candidate invariants I generated in Algorithm 3 are all ground, then Assumption **(A5)** is not necessary. In what follows we describe situations in which Assumption **(A5)** is not needed or is guaranteed to hold, so Lemma 4.9 and Theorem 4.10 hold under Assumptions **(A1)**– **(A4)**.

Ground Invariants

We first present a situation in which all candidate invariants I that are generated with Algorithm 3 are ground formulae, so Assumption **(A5)** is not needed.

Theorem 4.17. *For transition systems in which only variables (0-ary functions in $V \subseteq \Sigma$) are updated and for properties Ψ containing only variables in V, if all conditions imposed on the data structures used in the program are ground, then all candidate invariants I computed in the while loop in Algorithm 3 are ground formulae. In this case, Lemma 4.9 and Theorem 4.10 hold under Assumptions **(A1)**–**(A4)**.*

Proof: If all invariants I_n generated in the while loop of the algorithm are ground formulae, then the result in Theorem 3.5 does not need Assumption **(A5)**. □

The Array Property Fragment

We consider transition constraint systems $T = (\Sigma_S, \mathsf{Init}, \mathsf{Update})$ and properties Ψ, where $\mathcal{T}_S = \mathcal{T}_0 \cup \mathcal{K}$ and $\mathsf{Init}, \Psi, \mathcal{K}$ and Update_f, for $f \in F$, are all in the *array property fragment* (cf. Example 2.24). Then Assumptions (**A1**) and (**A2**) are not needed. We identify conditions under which we can guarantee that at every iteration of Algorithm 3 the candidate invariant I_n is in the array property fragment, so the locality assumption **(A5)** is fulfilled and does not need to be mentioned explicitly.

Many types of systems have descriptions in this fragment. In the following we give an example.

Example 4.18. Consider a controller of a water tank in which we have an inflow and an outflow depending on a moment in time t, denoted by $\mathsf{in}(t)$ and $\mathsf{out}(t)$, respectively. We here assume that the evolution of time goes in discrete measures, i.e. $\mathsf{in}(t)$ describes the inflow in the time unit from t to $t+1$. The values for the inflow and outflow in a time unit from t to $t+1$ can be chosen depending on the moment in time t between two minimum (index m) and two maximum (index M) values that depend on the moment in time, denoted by $\mathsf{in}^i_m(t)$ and $\mathsf{out}^i_M(t)$, where $i \in \{1, 2\}$. Let

- L be the current water level in the tank,
- L_{overflow} be the highest possible water level of the tank, i.e. by exceeding this level the tank will overflow, and
- L_{alarm} be a certain water level which marks a critical height of the water in the tank.

As global constraints for the inflow and outflow we have the formulae

$\mathsf{In}_i = \forall z(0 \leq \mathsf{in}^i_m(z) \leq \mathsf{in}^i_M(z) \leq L_{\mathsf{overflow}} - L_{\mathsf{alarm}} - \varepsilon_i)$ (where $\varepsilon_i > 0$) and

$\mathsf{Out}_i = \forall z(\mathsf{in}^i_M(z) \leq \mathsf{out}^i_m(z) \leq \mathsf{out}^i_M(z))$,

where $i \in \{1,2\}$. The initial states are described by the formula $\mathsf{Init} = (t \approx 0) \wedge (L \approx L_0)$. The updates are described by $\mathsf{Update} = \mathsf{Update}_{\mathsf{in}} \wedge L' \approx L{+}\mathsf{in}'(t) \wedge t' \approx t{+}1$, where

$$\begin{array}{rl}\mathsf{Update}_{\mathsf{in}} = \{ & \forall z(L \leq L_{\mathsf{alarm}} \wedge z \leq t_0 \rightarrow \mathsf{in}^1_m(z) \leq \mathsf{in}'(z) \leq \mathsf{in}^1_M(z)),\\ & \forall z(L \leq L_{\mathsf{alarm}} \wedge z > t_0 \rightarrow \mathsf{in}^2_m(z) \leq \mathsf{in}'(z) \leq \mathsf{in}^2_M(z)),\\ & \forall z(L > L_{\mathsf{alarm}} \wedge z \leq t_0 \rightarrow \mathsf{in}^1_m(z){-}\mathsf{out}^1_M(z) \leq \mathsf{in}'(z) \leq \mathsf{in}^1_M(z){-}\mathsf{out}^1_m(z)),\\ & \forall z(L > L_{\mathsf{alarm}} \wedge z > t_0 \rightarrow \mathsf{in}^2_m(z){-}\mathsf{out}^2_M(z) \leq \mathsf{in}'(z) \leq \mathsf{in}^2_M(z){-}\mathsf{out}^2_m(z))\ \}.\end{array}$$

Assume that $\Sigma_p = \{L_{\mathsf{alarm}}, L_{\mathsf{overflow}}, L, L_0, \mathsf{in}^1_m, \mathsf{in}^1_M, \mathsf{in}^2_m, \mathsf{in}^2_M, \mathsf{out}^1_m, \mathsf{out}^1_M, \mathsf{out}^2_m, \mathsf{out}^2_M\}$ and $\mathcal{K}_p = \{0 \leq L_{\mathsf{alarm}}, L_{\mathsf{alarm}} < L_{\mathsf{overflow}}, L_0 \leq L_{\mathsf{alarm}}\} \cup \{\mathsf{In}_1, \mathsf{Out}_1, \mathsf{In}_2, \mathsf{Out}_2\}$.

Let $\mathcal{T}_0 = \mathsf{LI}(\mathbb{R})$. The theory $\mathcal{T}_S$ is structured as follows: $\mathcal{T}_S = \mathcal{T}_0 \cup \mathcal{K}_p$. All the formulae appearing in $\mathcal{T}_S$, Init and $\mathsf{Update}_{\mathsf{in}}$ are in the array property fragment. In fact, these formulae satisfy also the conditions in Example 2.23, so we have the following local theory extensions:

$$\mathcal{T}_S \subseteq \mathcal{T}_S \cup \mathsf{Init} \qquad \text{and} \qquad \mathcal{T}_S \subseteq \mathcal{T}_S \cup \mathsf{Update}_{\mathsf{in}}.$$

We illustrate the way the refinement of Algorithm 3 described in Section 4.3.1 can be used to strengthen the property $\Psi = L \leq L_{\mathsf{overflow}}$.

Let $I_0 = \Psi$ be the first candidate invariant. It is easy to check that $\mathsf{Init} \models_{\mathcal{T}_S} I_0$. Indeed, $\mathcal{K}_p \wedge \mathsf{Init} \wedge L > L_{\mathsf{overflow}} \models \bot$ because $L_0 \leq L_{\mathsf{alarm}} \wedge L_{\mathsf{alarm}} < L_{\mathsf{overflow}} \wedge L \approx L_0 \wedge L > L_{\mathsf{overflow}}$ is already unsatisfiable.

We now check whether I_0 is invariant under updates, i.e. whether $I_0 \wedge \mathsf{Update} \wedge \neg I_0'$ is unsatisfiable. We can show that

$$\mathcal{K}_p \wedge L \leq L_{\mathsf{overflow}} \ \wedge\ \mathsf{Update}_{\mathsf{in}} \wedge L' \approx L{+}\mathsf{in}'(t) \wedge t' \approx t{+}1 \wedge L' > L_{\mathsf{overflow}}$$

is satisfiable using hierarchical reasoning in the chain of local theory extensions mentioned above, so I_0 is not an inductive invariant. We strengthen I_0 by applying the improvement of the symbol elimination algorithm, according to Theorem 3.7, to the following formula:

$$\mathsf{Update}_{\mathsf{in}} \wedge L' \approx L{+}\mathsf{in}'(t) \wedge t' \approx t{+}1 \wedge L' > L_{\mathsf{overflow}}$$

After instantiating the universally quantified variables z with t, using Lemma 4.12 we can bring $\mathsf{Update}_{\mathsf{in}}$ to DNF. By replacing the ground terms $f(t)$ with c_{ft} and the ground terms to be eliminated, i.e. $\mathsf{in}'(t), L'$ and t', with variables x_{in}, L' and t', respectively, we obtain the following disjunction of formulae:

$$\begin{array}{l}\bigvee_{i=1}^{2}(L{\leq}L_{\mathsf{alarm}} \wedge C_i(t) \wedge c_{\mathsf{in}^i_m(t)}{\leq}x_{in}{\leq}c_{\mathsf{in}^i_M(t)} \wedge L'{\approx}L + x_{in} \wedge t'{\approx}t + 1 \wedge L'{>}L_{\mathsf{overflow}}) \vee\\ \bigvee_{i=1}^{2}(L{>}L_{\mathsf{alarm}} \wedge C_i(t) \wedge c_{\mathsf{in}^i_m(t)} - c_{\mathsf{out}^i_M(t)}{\leq}x_{in}{\leq}c_{\mathsf{in}^i_M(t)} - c_{\mathsf{out}^i_m(t)} \wedge L'{\approx}L + x_{in} \wedge t'{\approx}t + 1\\ \qquad \wedge L'{>}L_{\mathsf{overflow}})\end{array}$$

For uniformity we use the notations $C_1(t) \coloneqq t \leq t_0$ and $C_2(t) \coloneqq t > t_0$.

After eliminating the existentially quantified variables x_{in}, L' and t' and replacing c_{gc} back with $g(c)$ we obtain:

$$\begin{array}{l}\bigvee_{i=1}^{2}(L{\leq}L_{\mathsf{alarm}} \wedge C_i(t) \wedge \mathsf{in}^i_m(t){\leq}\mathsf{in}^i_M(t) \wedge L_{\mathsf{overflow}} - L < \mathsf{in}^i_M(t)) \vee\\ \bigvee_{i=1}^{2}(L{>}L_{\mathsf{alarm}} \wedge C_i(t) \wedge \mathsf{in}^i_m(t){-}\mathsf{out}^i_M(t){\leq}\mathsf{in}^i_M(t){-}\mathsf{out}^i_m(t) \wedge L_{\mathsf{overflow}}{-}L < \mathsf{in}^i_M(t){-}\mathsf{out}^i_m(t))\end{array}$$

As t is not a parameter, we consider it to be existentially quantified. After negation we obtain the formula Γ_0 (equivalent to a formula in the array property fragment):

$\forall t(L{\leq}L_{\mathsf{alarm}} \wedge t \leq t_0 \wedge \mathsf{in}^1_m(t){\leq}\mathsf{in}^1_M(t) \rightarrow \mathsf{in}^1_M(t) \leq L_{\mathsf{overflow}} - L)$
$\forall t(L{\leq}L_{\mathsf{alarm}} \wedge t > t_0 \wedge \mathsf{in}^2_m(t){\leq}\mathsf{in}^2_M(t) \rightarrow \mathsf{in}^2_M(t) \leq L_{\mathsf{overflow}} - L)$
$\forall t(L{>}L_{\mathsf{alarm}} \wedge t \leq t_0 \wedge \mathsf{in}^1_m(t){-}\mathsf{out}^1_M(t){\leq}\mathsf{in}^1_M(t){-}\mathsf{out}^1_m(t) \rightarrow \mathsf{in}^1_M(t){-}\mathsf{out}^1_m(t){\leq}L_{\mathsf{overflow}}{-}L)$
$\forall t(L{>}L_{\mathsf{alarm}} \wedge t > t_0 \wedge \mathsf{in}^2_m(t){-}\mathsf{out}^2_M(t){\leq}\mathsf{in}^2_M(t){-}\mathsf{out}^2_m(t) \rightarrow \mathsf{in}^2_M(t){-}\mathsf{out}^2_m(t){\leq}L_{\mathsf{overflow}}{-}L)$

It can be checked, e.g. using H-PILoT, that $I_1 = I_0 \wedge \Gamma_0$ holds in the initial states and is invariant under updates, i.e. it is an inductive invariant of the system. ■

Lemma 4.19. *Under Assumption **(A3)**, the formulae Update_f are in the array property fragment if and only if the following hold:*

- *$\phi_1, \ldots \phi_{n_f}$ are conjunctions of constraints of the form $x \leq g$ or $x \geq g$, where x is a universally quantified variable and g is a ground term of sort index,*
- *all $\Sigma \cup \Sigma_p$-terms are flat, and*
- *all universally quantified variables occur below a function in $\Sigma \cup \Sigma_p$.*

Proof: Follows from the definition of the array property fragment (cf. Example 2.24). □

Lemma 4.20. *Let G be the negation of a formula in the array property fragment. Then the following are equivalent:*

(1) The formula obtained by applying Algorithm 2 to $\mathsf{Update}_f \wedge G$ is in the array property fragment.

(2) No instances of the congruence axioms need to be used for $\mathsf{est}(G)$.

(3) Either $\mathsf{est}(G)$ contains only one element, or whenever $f'(\overline{d}), f'(\overline{d'}) \in \mathsf{est}(G)$, where $\overline{d} = d_1, \ldots, d_n$ and $\overline{d'} = d'_1, \ldots, d'_n$, we have $\mathcal{T}_0 \cup \mathcal{K} \cup G \models \bigvee_{i=1}^n d_i \not\approx d'_i$.

Proof: Follows from the fact that the formula obtained after applying Algorithm 2,

$$\Big(\bigwedge_{p=1}^{k} \phi_{i_p}(y_p) \wedge \bigwedge_{(d_i,d_j)\in D} y_i \not\approx y_j \wedge G_0^g(\overline{y}, \overline{g}(y)) \Big),$$

can be an index guard only if it does not contain the disequalities $y_i \not\approx y_j$. This is the case if and only if $\mathsf{est}(G)$ contains only one element or if for all $f(d_i), f(d_j) \in \mathsf{est}(G)$ we have $\mathcal{T}_0 \cup G_0 \models d_i \not\approx d_j$. □

Theorem 4.21. *Let $T = (\Sigma_S, \mathsf{Init}, \mathsf{Update})$ be a transition constraint system with theory $\mathcal{T}_S = \mathcal{T}_0 \cup \mathcal{K}$. Assume that $\mathcal{T}_0$ is the disjoint combination of Presburger arithmetic (sort index) and a theory of elements (e.g. linear arithmetic over $\mathbb{Q}$ or $\mathbb{R}$). Assume that all functions in Σ are unary. If the formulae $\mathcal{K}, \mathsf{Init}, \mathsf{Update}$ and Ψ are in the array property fragment and all clauses in Ψ have only one universally quantified variable, then the formulae Γ_n obtained by symbol elimination in Step 2 at every iteration of Algorithm 3 are again in the array property fragment and are conjunctions of clauses having only one quantified variable.*

Proof: Follows from Lemma 4.19 and Lemma 4.20.

4.3.3 Termination

Algorithms of the form of Algorithm 3 do not terminate in general even for simple programs handling only integer or rational variables. However, we can identify situations in which the invariant generation procedure terminates. In the following we describe a situation in which this is the case.

The candidate invariants computed by Algorithm 3 in each step will always be different from each other and cannot be equivalent. If we assume that the generated candidate invariants can only be of a certain form, in which the number of potential candidate invariants is finite, then the algorithm is guaranteed to terminate after a finite number of steps.

Theorem 4.22 (A termination condition). *Let* Var *be a finite set of variables. Assume that the candidate invariants* I *generated at each iteration are conjunctions of flat clauses which contain terms in a given finite family* Ter *of terms. Then Algorithm 3 must terminate with an invariant* I *or after detecting that* $\mathsf{Init} \not\models I$.

Proof: Assume that Algorithm 3 does not terminate. From the finiteness of Var and Ter and the flatness of the clauses it follows that only finitely many different clauses can be generated. Then at some iteration n a candidate invariant $I_n = I_{n-1} \wedge \Gamma_{n-1}$ equivalent to I_{n-1} is obtained. Then, on the one hand the formula $I_{n-1} \wedge \mathsf{Update} \wedge \neg I'_{n-1}$ is satisfiable, as otherwise the algorithm would have terminated at the previous iteration. On the other hand the formula must be unsatisfiable, as it is equivalent to $(I_{n-1} \wedge \Gamma_{n-1}) \wedge \mathsf{Update} \wedge \neg I'_{n-1}$. This is a contradiction. □

A situation in which the termination condition holds is described below. To simplify the notation we assume that the functions in Σ have arity ≤ 1, but similar arguments can be used for n-ary functions as well.

Theorem 4.23. *Let* $\Sigma = \{f_1, \ldots, f_n\}$. *Assume that* $\mathcal{T}_0 = \mathcal{LI}(\mathbb{R})$ *and the following holds:*

- *All clauses used for defining* $\mathcal{T}_S$ *and the property* Ψ *contain only literals of the form*

$$x \rhd t, \qquad u \rhd v, \qquad f_i(x) \rhd s, \qquad f_i(x) \rhd y,$$

  where x, y *are (universally quantified) variables,* s, t, u, v *are constants,* $f_i \in \Sigma$, *and* $\rhd \in \{\leq, <, \geq, >, \approx\}$.

- *All axioms in* Update *are of form* $\forall \overline{x} \left(\phi_i^k(\overline{x}) \rightarrow C_i(\overline{x}, f'_k(\overline{x}))\right)$ *as in assumption (**A2**), where the formulae* $C_i(\overline{x}, y)$ *and* $\phi_i^k(\overline{x})$ *are conjunctions of literals of the form above.*

Then all the candidate invariants I *generated during the execution of Algorithm 3 are equivalent to conjunctions of flat clauses, which all contain terms in a finite set* Ter, *where the terms are formed with variables in a finite set* Var. *Since only finitely many clauses can be formed this way, after a finite number of steps no new formulae can be generated, thus the algorithm terminates.*

Proof: We analyze the way the formulae Γ are built using Algorithm 3. We prove by induction that for every $n \in \mathbb{N}$ the candidate invariant I_n constructed in the n-th iteration is a conjunction of clauses, where each clause contains universally quantified variables in the finite set Var and terms obtained from the terms in Ter.

Induction basis: For $I_0 = \Psi$ the property holds by assumption.

Induction hypothesis: We assume that the property holds for I_n.

Induction step: Consider the case in which $\mathsf{Init} \models I_n$ and I_n is not invariant under all updates, i.e. there exists $f \in \Sigma$ such that the formula $I_n \wedge \mathsf{Update}_f \wedge G$ is satisfiable, where G is obtained from $\neg I'_n$ after Skolemization. From the assumption that the property holds for I_n it follows that G is a conjunction of atoms of the form

$$c_x \rhd c_y, \quad c_x \rhd t, \quad f'(c_x) \rhd c_y, \quad f'(c_x) \rhd s, \quad g(c_y) \rhd c_z, \quad g(c_y) \rhd s, \quad u \rhd v,$$

where $g \in \Sigma$, t, s, u, v are constants, and $c_x, c_y, c_z, ...$ are Skolem constants introduced for variables $x, y, z, ...$ (where x, y, z are among the variables in $\overline{x}$ universally quantified in Γ'_n).

The analysis of the form of the candidate invariants obtained with Algorithm 3 in the proof of Theorem 4.13 shows that after steps 1-2 of Algorithm 2 we obtain the following type of formulae:

$$G'_{1(1)} \equiv \bigvee_{\substack{i_1,\ldots,i_k \in \{1,\ldots,n_f\} \\ D \subseteq \mathsf{est}(G)^2}} \left(\bigwedge_{p=1}^{k} \phi^0_{i_p}(d_p) \wedge \bigwedge_{(d_i,d_j)\in D} d_i \not\approx d_j \wedge \right.$$
$$\left. \exists x_{f'd_1}, \ldots x_{f'd_k} \left(\bigwedge_{p=1}^{k} C^0_{i_p}(d_p, x_{f'd_p}) \wedge \bigwedge_{(f'(d_i),f'(d_j))\in \mathsf{est}(G)^2 \setminus D} x_{f'd_i} \approx x_{f'd_j} \wedge G^0_0(\overline{d}, \overline{c}, \overline{c}_{gc}, \overline{x}_{f'd}) \right) \right)$$

In this case $\overline{c}, \overline{d}$ are the Skolem constants occurring in G, such that (with the notation in the proof of Theorem 4.13) $\mathsf{est}(G) = \{f'(d_1), \ldots, f'(d_k)\}$ and $\mathsf{est}(G_1) = \{g(c_1), \ldots, g(c_r)\}$. Quantifier elimination is applied to formulae of the following form:

$$\exists x_{fd_1}, \ldots x_{fd_k} \left(\bigwedge_{p=1}^{k} C^0_{i_p}(d_p, x_{fd_p}) \wedge \bigwedge_{(d_i,d_j)\in \mathsf{est}(G)^2 \setminus D} x_{fd_i} \approx x_{fd_j} \wedge G^0_0(\overline{d}, \overline{c}, \overline{c}_{gc}, \overline{x}_{fd}) \right)$$

Due to the assumptions on the literals that can occur in Γ_n, G and Update, these formulae contain only literals of the form $s \rhd t$, where $\rhd \in \{\leq, <, \geq, >, \approx\}$ and s and t are

(a) constants in Σ_p,

(b) Skolem constants in $\{d_1, \ldots, d_k, c_1, \ldots, c_r\}$,

(c) constants of the form c_{fd} or c_{gc}, or

(d) variables of the form $x_{f'd}$ which need to be eliminated.

After quantifier elimination (Step 3 of the algorithm) we obtain a disjunction $G''_{1(1)}$ of conjunctions of literals of the form $s \rhd t$, where s and t are as in (a)-(c) above.

After Step 4 of the algorithm we obtain a disjunction of conjunctions of literals of the form $s \rhd t$, where s and t are as in (a)-(b) or terms of the form $f(c)$ and $g(c)$, with $c \in \{d_1, \ldots, d_k, c_1, \ldots, c_n\}$.

In Step 5, after replacing the Skolem constants c_i, d_i again with variables x_i, y_i and negating, we obtain a conjunction of clauses, where each clause contains at most as many universally quantified variables as Γ_n, say $\{x_1, \ldots, x_m\}$, and only terms of the form $s \rhd t$, where s and t are constants in Σ_p, variables x_i, or terms of the form $f(x_i)$ and $g(x_i)$ with $f \in \Sigma$ and $g \in \Sigma_p$. Note that, under the assumptions we made, the number of variables in the newly generated clauses does not grow. For a fixed set of variables $\{x_1, \ldots, x_m\}$ there are only finitely many terms of the form above. Thus, after a finite number of steps no new formulae can be generated and the algorithm terminates. □

Termination vs. Non-termination

The goal of Algorithm 3 is to obtain an inductive invariant containing only parameters specified by a set Σ_p. Therefore, we usually eliminate all symbols except for the ones in Σ_p. However, if we eliminate some of the symbols from Σ_p as well, then we would get an invariant in a slightly more restricted language (assuming the algorithm terminates with an invariant), but it would still only contain symbols from Σ_p.

In our experiments we have discovered that sometimes the choice of the variables to be eliminated can make a difference for the termination of the algorithm. Therefore, as an heuristic, we could try different possibilities of eliminating additional symbols from Σ_p in order to check in which case we have termination. We give an example for this.

Example 4.24. Consider again the system S from Example 4.7, for which we have shown that the algorithm in its standard form does not terminate:

Let $S = (\mathcal{T}_S, T)$, where $\mathcal{T}_S = \mathsf{LI}(\mathbb{Z})$ and $T = (\Sigma_S, \mathsf{Init}, \mathsf{Update})$ is a transition constraint system, where:

- $\Sigma_S = \Sigma_0 \cup \Sigma_p$, where
  - Σ_0 is the signature of linear integer arithmetic and
  - $\Sigma_p = \{x, y, z, N\}$, where x, y, z and N are 0-ary function symbols
- $\mathsf{Init} = (x \approx 0 \wedge y \approx 0 \wedge z \approx 0)$
- $\mathsf{Update} = (x \leq N \rightarrow (x' \approx x + 1 \wedge y' \approx y + 1 \wedge z' = z + x' - y'))$

Let $\Psi = (z \leq 0)$. The goal is to show that Ψ is invariant, e.g. by finding an inductive invariant which entails it.

The first candidate invariant $I_0 = \Psi = (z \leq 0)$ is not an inductive invariant and needs to be strengthened. The standard approach, which was used in Example 4.7, would be to eliminate only the symbols which are not in Σ_p, i.e. the primed variables x',y' and z', from the following formula:

$F_0 = (z \leq 0) \wedge \mathsf{Update}_x \wedge \mathsf{Update}_y \wedge \mathsf{Update}_z \wedge (z' > 0)$, where

$\mathsf{Update}_x = (x \leq N \rightarrow x' \approx x + 1) \wedge (x > N \rightarrow x' \approx x)$

$\mathsf{Update}_y = (x \leq N \rightarrow y' \approx y + 1) \wedge (y > N \rightarrow y' \approx y)$

$\mathsf{Update}_z = (z \leq N \rightarrow z' \approx z + x' - y') \wedge (z > N \rightarrow z' \approx z)$

With this approach we ended up in an infinite sequence of strengthenings. Assume now that we additionally eliminate the variable z. We then obtain $x \leq N \wedge y - x < 0$, which after negation yields the following constraint:

$\Gamma = x \leq N \rightarrow x \leq y$

Indeed, the newly obtained candidate invariant $I_1 = I_0 \wedge \Gamma = (z \leq 0) \wedge (x \leq N \rightarrow x \leq y)$ can be proved to be an inductive invariant of the system S.

4.4 Implementation

Essential to our invariant generation method (Algorithm 3) is on the one hand the hierachical reduction of a formula defining a chain of local theory extensions, and on the other hand the property-directed symbol elimination. As a tool for performing the reduction to the base theory we can use H-PILoT (see Section 2.4), for property-directed symbol elimination we can use Algorithm 2, implemented in SEH-PILoT (see Section 3.2.2). As both of these things can be handled using SEH-PILoT, it was only logical to include the invariant strengthening as a separate mode in SEH-PILoT.

While the symbol elimination algorithm was implemented by Philipp Marohn as part of his Bachelor thesis, he did the implementation of the invariant strengthening algorithm at a later point. The preliminary system description [78] does not include a description of this relatively new mode. An implementation of the optimization using the DNF transformation described in Lemma 4.12 is currently work in progress. In the following we describe how to use SEH-PILoT for invariant checking, constraint synthesis and invariant strengthening in parametric transition systems.

The input file for SEH-PILoT is a YAML-file specifying a mode, a specification type, a specification theory, a specification and possibly some additional options. In order to use SEH-PILoT as a tool for the verification of parametric systems, we have to set the specification type to PTS (short for Parametric Transition Systems). SEH-PILoT supports two specification theories, REAL_CLOSED_FIELDS and PRESBURGER_ARITHMETIC. Specification type PTS can be used together with the modes GENERATE_CONSTRAINTS and INVARIANT_STRENGTHENING, which refer to the problems "constraint synthesis" and "invariant generation", respectively, as described in Section 4.1. In both modes it is obligatory to specify the symbols which are to be eliminated or, alternatively, the parameters (i.e. the symbols which are not eliminated), using the option eliminate or parameter, respectively. In the SPECIFICATION the following components are specified:

- base_functions: specification of the base functions as in H-PILoT
- extension_functions: specification of the extension functions as in H-PILoT
- relations: specification of the relations as in H-PILoT
- axioms: specification of axioms corresponding to the theory $\mathcal{T}_S$
- init: specification of the initial conditions in the transition constraint system
- update: specification of the update rules in the transition constraint system
- safety: specification of the safety condition Ψ, which is the first candidate invariant
- update_vars: gives a list of pairs $x : x'$ mapping variable x to its update variable x'

The invariant generation method consists of two steps that are possibly repeated. It is first checked whether a candidate invariant is already an invariant (invariant checking), and if it is not, then a stronger invariant is generated (invariant strengthening).

For invariant checking H-PILoT is used. It can do the reduction and call a prover. If the answer is "unsatisfiable", we know that our candidate invariant is an inductive invariant. If the answer is "satisfiable", then it is not an inductive invariant and we have to apply the invariant strengthening. Note that the fact that all theory extensions are local needs to

be checked in advance by hand. If locality is not given, then the answer "satisfiable" could also mean that not enough instances were considered.

For invariant strengthening the symbol elimination algorithm in SEH-PILoT is used, which uses Redlog to eliminate all the symbols which are not parameters. The output of this step will be a formula (only containing the parameters) describing the weakest condition for the candidate invariant to hold after the update. This in conjunction with the old candidate invariant is the new candidate invariant for the next step.

The option slfq_formula can be used to apply the SLFQ simplifier to the formula before quantifier elimination, the option slfq_query can be used to apply the SLFQ simplifier to the result of the quantifer elimination. To not run into an infinite loop of strengthenings, SEH-PILoT forces termination by having a maximum number of iterations specified. Currently, by default this limit is set to 5, but the user can set it to any natural number using the option inv_str_max_iter.

As explained in Section 3.2.2, it is possible to specify several tasks in one YAML-file and to use anchor and alias to use the same specification in more than one task in an easy way. Under sehpilot_options and task_options one can set certain global options, which will apply to all tasks specified in the YAML-file. The task option print_steps can be set to true (by default it is false) in order to get a more detailed output, showing e.g. also results of intermediate steps, i.e. intermediate candidate invariants. The SEH-PILoT option keep_files can be set to true (by default it is false) in order to get in addition to the YAML-file containing the output also all the intermediate files used for invariant checking and invariant strengthening in each iteration, i.e. the corresponding input files for H-PILoT and for Redlog.

The output is given as a YAML-file. If within the iteration limit an inductive invariant was found, the formula is given as a result. If within the iteration limit the invariant checking fails because the candidate invariant does not hold in the initial states, a note "No universal inductive invariant over parameters Σ_p entails the safety property" is given in the output. If the iteration limit n is exceeded without an inductive invariant being found yet, a note "Reached iteration limit (n)" is given in the output. If the option print_steps is set to true, then in any case the candidate invariants computed after each iteration are displayed in the output file in addition to the end result. The total time required for the computation and the time required for each separate task will alway be given (in seconds) as part of the output file as well.

Example 4.25. Consider the system $S = (\mathcal{T}_S, T)$ from Example 4.6, where $\mathcal{T}_S = \mathsf{LI}(\mathbb{Z})$ and $T = (\Sigma_S, \mathsf{Init}, \mathsf{Update})$ is a transition constraint system, where:

- $\Sigma_S = \Sigma_0 \cup \Sigma_p$, where
  - Σ_0 is the signature of linear integer arithmetic and
  - $\Sigma_p = \{x, y\}$, where x and y are 0-ary function symbols
- $\mathsf{Init} = (x \approx y \vee x \approx y + 2)$
- $\mathsf{Update} = (x \leq y + 1 \rightarrow x' \approx x + 2) \wedge (x > y + 1 \rightarrow x' \approx x) \wedge (y' \approx y)$

Let $\Psi = (y \leq x \wedge x \leq y + 2)$. The goal is to show that Ψ is an invariant of the system, possibly by strengthening it to an inductive invariant.

We apply the implementation of Algorithm 3 in SEH-PILoT.

We use specification type PTS, specification theory PRESBURGER_ARITHMETIC and mode INVARIANT_STRENGTHENING. In the specification we specify base functions, extension functions and relations similar to H-PILoT. Note that the specification in this mode is not an H-PILoT file as in mode HPILOT, so the syntax is a little different (the H-PILoT input files will be generated by SEH-PILoT afterwards). Under init and update the formulae describing the initial states and the update rules are given, respectively. Under query the safety property is specified. We here use names "xp" and "yp" for the primed variables (i.e. the updated variables). We denote the update variables in SEH-PILoT by matching them with the corresponding variables under update_vars. To see how SEH-PILoT uses H-PILoT and Redlog, we set the SEH-PILoT option keep_files to true, such that the input files for H-PILoT and Redlog are kept for the user (otherwise these files will be deleted after they have been used).

We have the following input file for SEH-PILoT:

```
sehpilot_options:
    keep_files: true

tasks:
    example_4.25:
        mode: INVARIANT_STRENGTHENING
        options:
            parameter: [x, y]
        specification_type: PTS
        specification_theory: PRESBURGER_ARITHMETIC
        specification:
            base_functions: "{(+,2), (-,2), (*,2)}"
            extension_functions: "{}"
            relations: "{(<=,2), (<,2), (>=,2), (>,2)}"
            init: |
                OR(x = y, x = y + _2);
            update: |
                x <= y + _1 --> xp = x + _2;
                x > y + _1 --> xp = x;
                yp = y;
            query: |
                y <= x;
                x <= y + _2;
            update_vars:
                x : xp
                y : yp
```

We get the following output from SEH-PILoT:

```
Metadata:
    Date: '2023-08-01 12:54:15'
    Number of Tasks: 1
    Runtime Sum: 0.6982
example_4.25:
    Runtime: 0.6982
    Result:
        Inductive Invariant: |-
            NOT((x - y) - _1 = _0);
            (x - y) - _2 <= _0;
            x - y >= _0;
```

The result is equivalent to the formula $I_1 = x \approx y \vee x \approx y + 2$ which we obtained in Example 4.6.

Due to the option keep_files we also get the following input files for H-PILoT (loc-files) which were generated by SEH-PILoT and the corresponding input files for Redlog (dat-files) which were generated by H-PILoT:

- VC_Init_1.loc and VC_Init_1.dat
- VC_Init_2.loc and VC_Init_2.dat
- VC_Update_1.loc and VC_Update_1.dat
- VC_Update_2.loc and VC_Update_2.dat
- Strengthening_1.dat

In this example it took only one iteration to compute the inductive invariant, so we have two candidate invariants (the original safety property and the formula obtained after strengthening). In the files this is indicated by the number (1 for the first candidate and 2 for the second). For each candidate invariant it was checked whether they hold in the initial states (files with "Init" in their name) and whether they are invariant under updates (files with "Update" in their name).

The last file in the above list is the one that was used for computing the strengthening. It is essentially the same input as the first Update-file, with the only difference being that for satisfiability checking all the symbols are existentially quantified in the file (and therefore all symbols are eliminated), while for strengthening the parameters are not existentially quantified and therefore not eliminated.

To give an example we show the files for checking whether the first candidate invariant is invariant under updates. The file VC_Update_1.loc contains the following input for H-PILoT:

```
Base_functions := {(+,2), (-,2), (*,2)}
Extension_functions := {}
Relations := {(<=,2), (<,2), (>=,2), (>,2)}

Clauses := % --SAFETY CONDITION--
           y <= x;
           x <= y + _2;
           % --UPDATE--
           x <= y + _1 --> xp = x + _2;
           x > y + _1 --> xp = x;
           yp = y;

Query :=   OR(xp - yp < _0, (xp - yp) - _2 > _0);
```

Using the above file as an input for H-PILoT a corresponding input file VC_Update_1.dat for Redlog is generated.

```
load_package redlog;
rlset PASF;
on rlsimpl;
off nat;
off rlverbose;
on rlnzden;
vars := {x, xp, y, yp};
formula := (
(xp - yp < 0 or (xp - yp) - 2 > 0) and
yp = y and
((x > y + 1) impl (xp = x)) and
((x <= y + 1) impl (xp = x + 2)) and
x <= y + 2 and
y <= x
 );
query :=  (rlqe ex(vars, formula));
end;
```

Note that the file Strengthening_1.dat is identical to the one above, with the only exception being that the set of variables "vars" in this case does not contain the parameters x and y.

Example 4.26. Consider the system $S = (\mathcal{T}_S, T)$ from Example 4.7, where $\mathcal{T}_S = \mathsf{LI}(\mathbb{Z})$ and $T = (\Sigma_S, \mathsf{Init}, \mathsf{Update})$ is a transition constraint system, where:

- $\Sigma_S = \Sigma_0 \cup \Sigma_p$, where
  - Σ_0 is the signature of linear integer arithmetic and
  - $\Sigma_p = \{x, y, z, N\}$, where x, y, z and N are 0-ary function symbols
- $\mathsf{Init} = (x \approx 0 \wedge y \approx 0 \wedge z \approx 0)$
- $\mathsf{Update} = \mathsf{Update}_x \wedge \mathsf{Update}_y \wedge \mathsf{Update}_z$, where
  - $\mathsf{Update}_x = (x \leq N \rightarrow x' \approx x + 1) \wedge (x > N \rightarrow x' \approx x)$
  - $\mathsf{Update}_y = (x \leq N \rightarrow y' \approx y + 1) \wedge (x > N \rightarrow y' \approx y)$
  - $\mathsf{Update}_z = (x \leq N \rightarrow z' \approx z + x' - y') \wedge (x > N \rightarrow z' \approx z)$

Let $\Psi = (z \leq 0)$ be the property to be checked for invariance. We apply the implementation of Algorithm 3 in SEH-PILoT.

We use specification type PTS, specification theory PRESBURGER_ARITHMETIC and mode INVARIANT_STRENGTHENING. We know from Example 4.7 that the procedure does not terminate if we have parameters $\Sigma_p = \{x, y, z, N\}$ and from Example 4.24 that the algorithm terminates if we additionally eliminate z. To test both variants, we state two tasks with the same specification. In the first task we use parameters $\{x, y, z, N\}$, in the second one we use $\{x, y, N\}$. To force termination of SEH-PILoT also on the non-terminating task, we use the option inv_str_max_iter to set the maximal number of iterations to 3. We use the global SEH-PILoT option keep_files to get all the intermediate files generated by SEH-PILoT during the execution of the algorithm and the global task option print_steps to have the candidate invariants computed in each iteration written in the output file.

We have the following input file for SEH-PILoT:

```
sehpilot_options:
    keep_files: true

task_options:
    print_steps: true

tasks:
    example_4.26:
        mode: INVARIANT_STRENGTHENING
        options:
            inv_str_max_iter: 3
            parameter: [x, y, z, n]
        specification_type: PTS
        specification_theory: PRESBURGER_ARITHMETIC
        specification: &spec_ex4_28
            base_functions: "{(+,2), (-,2), (*,2)}"
            extension_functions: "{}"
            relations: "{(<=,2), (<,2), (>=,2), (>,2)}"
            init: |
                x = _0;
                y = _0;
                z = _0;
            update: |
                x <= n --> xp = x + _1;
                x <= n --> yp = y + _1;
                x <= n --> zp = (z + xp) - yp;
                x > n --> xp = x;
                x > n --> yp = y;
                x > n --> zp = z;
            query: |
                z <= _0;
            update_vars:
                x : xp
                y : yp
                z : zp

    example_4.26_elim_z:
        mode: INVARIANT_STRENGTHENING
        options:
            inv_str_max_iter: 3
            parameter: [x, y, n]
        specification_type: PTS
        specification_theory: PRESBURGER_ARITHMETIC
        specification: *spec_ex4_28
```

As expected, the maximum number of iterations, in this case 3, is exceeded in the first task. Due to the print_steps option we can see in the output file all the candidate invariants that were computed in the three iterations of the algorithm, which show nicely the pattern that is repeated infinitely and causes non-termination. 15 intermediate files were generated in total (and kept because of the keep_files option):

- 3 loc-files and 3 dat-files for checking the initial states,
- 3 loc-files and 3 dat-files for checking the updates, and
- 3 dat-files for the strengthenings.

The second task terminates after only one step with the output of an inductive invariant that is equivalent to the inductive invariant $I_1 = (z \leq 0) \wedge (x \leq N \rightarrow x \leq y)$ that we obtained in Example 4.24. Nine intermediate files were generated, two loc-files and two dat-files each for checking the initial states and the updates and one dat-file for the strengthening. The following output file was generated by SEH-PILOT:

```
Metadata:
    Date: '2023-08-01 15:55:30'
    Number of Tasks: 2
    Runtime Sum: 1.894
example_4.26:
    Runtime: 1.2004
    Result:
        Inductive Invariant: NA
        Note: Reached iteration limit (3)
    Steps:
        1. Step:
            New Candidate: |-
                z <= _0;
                OR(n - x < _0,
                   (x - y) + z <= _0);
        2. Step:
            New Candidate: |-
                z <= _0;
                OR(n - x <= _0,
                   ((_2 * x) - (_2 * y)) + z <= _0,
                   (x - y) + z > _0);
                OR(n - x < _0,
                   (x - y) + z <= _0);
        3. Step:
            New Candidate: |-
                z <= _0;
                OR((n - x) - _1 = _0,
                   n - x <= _0,
                   ((_3 * x) - (_3 * y)) + z <= _0,
                   ((_2 * x) - (_2 * y)) + z > _0,
                   (x - y) + z > _0);
                OR(n - x <= _0,
                   ((_2 * x) - (_2 * y)) + z <= _0,
                   (x - y) + z > _0);
                OR(n - x < _0,
                   (x - y) + z <= _0);
example_4.26_elim_z:
    Runtime: 0.6936
    Result:
        Inductive Invariant: |-
            z <= _0;
            OR(n - x < _0, (x - y) - _1 <= _0);
            OR(n - x < _0, x - y <= _0);
    Steps:
        1. Step:
            New Candidate: |-
                z <= _0;
                OR(n - x < _0, (x - y) - _1 <= _0);
                OR(n - x < _0, x - y <= _0);
```

4.5 Conclusion

We considered parametric systems described by their initial states and transition rules for the upates of the system. We proposed a method for property-directed invariant generation in such systems and analyzed its properties.

We start from a given universal formula describing a property of the data of the system. We can also consider properties on individual elements of an array, such as $a[i] \leq b[j]+c[k]$ for fixed indices i, j, k, and "global properties", for instance sortedness or a property such as $\forall i, j, k(a[i] \leq b[j] + c[k])$. These are properties which describe relationships which refer to the values of the variables or of the functions (e.g. arrays) at a given, fixed iteration in the execution of a loop. The invariants we generate have a similar form.

Our results extend the results in [21] and [38], as we consider more complex theories. In contrast to approaches to invariant generation that use templates, such as e.g. [16], we do not start with concrete templates. We specify beforehand which class of formulae the invariants should belong to, by restricting the signature for the computed invariants.

There are similarities between our method and the one in [80], but our approach is different and does not restrict to theories with the finite model property (although a form of finiteness ensures termination). In [69, 80], if a counterexample to the inductiveness of a candidate invariant I is found, a formula is constructed using the finite model property assumption and added to I to avoid finding the same counterexample again in the next iteration. In our work we do not try to avoid a single counterexample to the inductiveness of the current candidate invariant, but instead we use the symbol elimination method in Algorithm 2 to compute the weakest formula Γ with the property that under the additional conditions Γ the formula I is invariant, which rules out a whole class of counterexamples at once. The procedure has to be iterated since Γ itself might not be invariant under transitions. We think that our method should help to accelerate the procedure compared to the diagram-based approach.

The algorithm proposed in [53] for the theories of arrays uses a non-deterministic function ChooseCover that returns a cover of a formula (as an approximation of the reachable states). It is proved that if the theory of elements is locally finite, then for every universal formula Ψ a universal inductive invariant I strengthening Ψ exists if and only if there exists a suitable ChooseCover function for which the algorithm returns an inductive invariant strengthening Ψ. In contrast to the algorithm proposed in [53], our algorithm is deterministic.

The methods used in [73, 72, 60] and also in [56] often introduce a new argument to constants and function symbols. If n is the arity of a function f, then a version of f with arity $n+1$ is used, where $f(x_1, \ldots, x_n, i)$ denotes the value of $f(x_1, \ldots, x_n)$ at iteration i. A major difference between our approach and the methods for invariant generation used in [73, 72, 60] and [56] is that we do not use additional indices to refer to the values of variables at iteration steps and do not quantify over the iteration steps. However, an extension with quantification over iteration steps and possibilities of giving explicit solutions to at least simple types of recurrences seems to be feasible.

We analyzed correctness and termination of our invariant strengthening algorithm. For proving partial correctness we had to make several assumptions related to the locality of the extensions. These assumptions hold for instance for systems in which all properties are expressed in the array property fragment. While our method is not guaranteed to terminate in general, we identified situations in which termination is guaranteed. To prove

termination we show that the length of the quantifier prefix in the candidate invariants generated in every iteration does not grow. Termination is then guaranteed if only finitely many atomic formulae formed with a fixed number of variables can be generated using quantifier elimination when applying the algorithm.

We analyzed the applicability of our methods on several examples. In our tests we used the implementation described in Section 4.4, which is part of the program SEH-PILoT. Using the mode PTS for parametric transition systems one can specify the initial states and update rules of the system as well as the property that is supposed to be invariant. Then one can choose between satisfiability checking, constraint generation and invariant strengthening and also between real closed fields and Presburger arithmetic as base theories. The tests that we made for invariant strengthening showed that usually, if the algorithm terminates, it does so in very few iterations. In fact, all the terminating examples terminated after only one iteration.

4.5.1 Future Work

As future work we would like to investigate how to use similar ideas for the goal-oriented generation of inductive properties for recursively defined functions. The tasks are likely to be very similar, as we would first try to prove a property by structural induction and, if it cannot be proved, we would either generate constraints on parameters or strengthen the property until it becomes an inductive property.

In addition to that, we plan to identify additional situations in which our invariant generation method is correct, terminates and has low complexity. This can be done by considering either other theories or more general first-order properties.

Another interesting question to investigate further is how locality can be detected easily, since locality of the involved theory extensions is a key aspect of our approach and an assumption that was needed to prove correctness.

We here restricted to universally quantified invariants and theories related to the array property fragment, but an extension to a framework using the notion of "extended locality", in which both the axioms in $\mathcal{K}$ and the formula G to refute are allowed to contain arbitrary formulae over the base theory (cf. [63, 64]), seems unproblematic.

Performing instantiation and quantified generalization on demand as done in [57] might be useful for theories for which it is not known whether complete instantiations exist. Incorporating such ideas in our own approach could be useful even if completeness cannot be guaranteed in this case.

5 Finding Explanations in $\mathcal{EL}^+$

When combining two ontologies or when extending an existing ontology with additional axioms we may get new consequences arising from this combination. Some of these consequences may be desirable, but also unwanted consequences could arise, which would require repairing the ontology. For this it is important to locate the reason for the occurrence of those faulty consequences. In this section we present a method for finding explanations for subsumptions in $\mathcal{EL}$ and $\mathcal{EL}^+$ ontologies that is based on a translation to the theory of semilattices and a form of interpolation which we call P-interpolation (for some set P of predicates). For an illustration of the problem we are addressing see Section 1.1.

We first describe the description logics $\mathcal{EL}$ and $\mathcal{EL}^+$ with their algebraic semantics, i.e. classes $SL_{\mathcal{EL}}$ and $SL_{\mathcal{EL}^+}$ of semilattices with monotone operators, and present existing locality results for the theories of both classes. Afterwards we define the notion of P-interpolation, which was first studied in [91], and show that the classes $SL_{\mathcal{EL}}$ and $SL_{\mathcal{EL}^+}$ both have this property for the predicate $P = \{\leq\}$. We then propose an algorithm for obtaining high-level explanations for subsumptions in $\mathcal{EL}$ and $\mathcal{EL}^+$ based on the computation of $\leq$-interpolants. We demonstrate the applicability of this algorithm on various examples.

Parts of the results in this chapter were already published in [83]. The results have recently been extended in [86].

5.1 The Description Logics $\mathcal{EL}$ and $\mathcal{EL}^+$

The definitions of the basic notions of description logics given here are based on the Description Logic Handbook by Franz Baader et al. [6]. The central notions in description logics are concepts and roles. In any description logic a set N_C of *concept names* and a set N_R of *roles* is assumed to be given (we also call these atomic concepts and atomic roles). A *concept description* is either an atomic concept, an atomic role, or a complex concept that is built from them using *concept constructors*. The available constructors determine the expressive power of a description logic.

The semantics of description logics is defined in terms of interpretations $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\Delta^{\mathcal{I}}$ is a non-empty set, and the function $\cdot^{\mathcal{I}}$ maps each concept name $C \in N_C$ to a set $C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ and each role name $r \in N_R$ to a binary relation $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. Figure 5.1 shows the constructor names used in the description logic $\mathcal{ALC}$ and their semantics. The extension of $\cdot^{\mathcal{I}}$ to concept descriptions is inductively defined using the semantics of the constructors.

Restricting the type of allowed concept constructors from Figure 5.1 leads to less expressive but more tractable description logics. If we only allow top, intersection and existential restriction as concept constructors, we obtain the description logic $\mathcal{EL}$. By extending $\mathcal{EL}$ with role inclusion axioms we get the description logic $\mathcal{EL}^+$.

Constructor name	Syntax	Semantics
bottom	$\bot$	$\emptyset$
top	$\top$	$\Delta^{\mathcal{I}}$
negation	$\neg C$	$\Delta^{\mathcal{I}} \backslash C^{\mathcal{I}}$
intersection	$C_1 \sqcap C_2$	$C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$
union	$C_1 \sqcup C_2$	$C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$
existential restriction	$\exists r.C$	$\{x \mid \exists y((x,y) \in r^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}})\}$
universal restriction	$\forall r.C$	$\{x \mid \forall y((x,y) \in r^{\mathcal{I}} \longrightarrow y \in C^{\mathcal{I}})\}$

Figure 5.1: $\mathcal{ALC}$ constructors and their semantics.

Relationships between concepts and roles are described using TBoxes or, more generally, using CBoxes.

Definition 5.1 (TBox, Model, TBox Subsumption). *A TBox (or terminology) is a finite set consisting of*

- primitive concept definitions *of the form $C \equiv D$, where C is a concept name and D a concept description, and*
- general concept inclusions *(GCIs) of the form $C \sqsubseteq D$, where C and D are concept descriptions.*

We define the following:

- *An interpretation $\mathcal{I}$ is a* model of a TBox $\mathcal{T}$ *if it satisfies*
  - *all concept definitions in $\mathcal{T}$, i.e. $C^{\mathcal{I}}{=}D^{\mathcal{I}}$ for all definitions $C{\equiv}D \in \mathcal{T}$, and*
  - *all general concept inclusions in $\mathcal{T}$, i.e. $C^{\mathcal{I}}{\subseteq}D^{\mathcal{I}}$ for every $C{\sqsubseteq}D \in \mathcal{T}$.*
- *Let $\mathcal{T}$ be a TBox and let C_1, C_2 be two concept descriptions.* C_1 is subsumed by C_2 w.r.t. $\mathcal{T}$ *($C_1 \sqsubseteq_{\mathcal{T}} C_2$) if and only if $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\mathcal{T}$.*

Since definitions can always be expressed as double inclusions, in what follows we will refer to TBoxes consisting of general concept inclusions only.

Definition 5.2 (CBox, Model, CBox Subsumption). *A CBox consists of a TBox $\mathcal{T}$ and a set RI of role inclusions of the form $r_1 \circ \cdots \circ r_n \sqsubseteq s$. Since terminologies can be expressed as sets of general concept inclusions, we will view CBoxes as unions $GCI{\cup}RI$ of a set GCI of general concept inclusions and a set RI of role inclusions of the form $r_1 \circ \cdots \circ r_n \sqsubseteq s$, with $n{\geq}1$. We define the following:*

- *An interpretation $\mathcal{I}$ is a model of the CBox $\mathcal{C} = GCI \cup RI$ if it is a model of GCI and satisfies all role inclusions in $\mathcal{C}$, i.e. $r_1^{\mathcal{I}} \circ \cdots \circ r_n^{\mathcal{I}} \subseteq s^{\mathcal{I}}$ for all $r_1 \circ \cdots \circ r_n \subseteq s \in RI$.*
- *If $\mathcal{C}$ is a CBox and C_1, C_2 are concept descriptions, then $C_1 \sqsubseteq_{\mathcal{C}} C_2$ if and only if $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\mathcal{C}$.*

For our applications we restrict to the less expressive but tractable description logic $\mathcal{EL}$ [4] and its extension $\mathcal{EL}^+$ [9, 8], which are used for example in terminological reasoning in medicine, like the clinical database SNOMED [103, 102]. In $\mathcal{EL}$ and $\mathcal{EL}^+$ CBox subsumption checking can be done in polynomial time and we can use an encoding of this problem as a uniform word problem in a suitable class of semilattices.

In [9] it was shown that subsumption w.r.t. CBoxes in $\mathcal{EL}^+$ can be reduced in linear time to subsumption w.r.t. *normalized* CBoxes, in which all GCIs have one of the forms

$$\begin{aligned} C &\sqsubseteq D, \\ C_1 \sqcap C_2 &\sqsubseteq D, \\ C &\sqsubseteq \exists r.D, \\ \exists r.C &\sqsubseteq D, \end{aligned}$$

where C, C_1, C_2, D are concept names, and all role inclusions are of the form $r \sqsubseteq s$ or $r_1 \circ r_2 \sqsubseteq s$, where r, s, r_1, r_2 are role names. Therefore, in what follows we consider w.l.o.g. that CBoxes only contain role inclusions of the form $r \sqsubseteq s$ and $r_1 \circ r_2 \sqsubseteq s$.

Algebraic Semantics for $\mathcal{EL}$ and $\mathcal{EL}^+$

In [92] the link between TBox subsumption in $\mathcal{EL}$ and uniform word problems in the corresponding classes of semilattices with monotone functions was studied. In [94, 95] it was shown that these results naturally extend to the description logic $\mathcal{EL}^+$.

The relation $\leq$ is a (non-strict) *partial order* on a set S if it satisfies the following conditions:

(1) Reflexivity: $\forall x \in S : x \leq x$

(2) Antisymmetry: $\forall x, y \in S : x \leq y \wedge y \leq x \rightarrow x = y$

(3) Transitivity: $\forall x, y, z \in S : (x \leq y \wedge y \leq z) \rightarrow x \leq z$

S together with $\leq$ is then called a *partially ordered set* (poset).

A partially ordered set S is a *meet-semilattice* ($\wedge$-semilattice) if for any two elements x and y a greatest lower bound exists (called the *meet* of x and y and denoted with $x \wedge y$). The class SLat of (meet-)semilattices can alternatively be axiomatized by the following axioms:

- Associativity: $\forall x, y, z : x \wedge (y \wedge z) \approx (x \wedge y) \wedge z$
- Commutativity: $\forall x, y : x \wedge y \approx y \wedge x$
- Idempotency: $\forall x : x \wedge x \approx x$

Therefore, SLat is an equational class.

Let $\mathsf{SLO}(\Sigma) = \mathsf{SLat} \cup \bigcup_{f \in \Sigma} \mathsf{Mon}(f)$ be the theory of $\wedge$-semilattices $(S, \wedge, \{f_S\}_{f \in \Sigma})$ with unary operators in Σ, such that, for every $f \in \Sigma$, $f_S : S \rightarrow S$ is a monotone function, i.e. f satisfies the condition

$$\mathsf{Mon}(\Sigma) = \bigwedge_{f \in \Sigma} \forall x, y \big(x \leq y \rightarrow f(x) \leq f(y)\big).$$

When defining the semantics of $\mathcal{EL}$ or $\mathcal{EL}^+$ with role names N_R we use a class of $\wedge$-semilattices with monotone operators of the form $(S, \wedge, \{f_{\exists r}\}_{r \in N_R})$. Every concept description C can be represented as a term $\overline{C}$ by the following inductively defined encoding:

- Every concept name $C \in N_C$ is regarded as a variable $\overline{C} = C$.
- $\overline{C_1 \sqcap C_2} = \overline{C}_1 \wedge \overline{C}_2$ and $\overline{\exists r.C} = f_{\exists r}(\overline{C})$.

If RI is a set of role inclusions of the form $r \sqsubseteq s$ and $r_1 \circ r_2 \sqsubseteq s$, let RI_a be the set of all axioms of the form

$$\begin{array}{ll} \forall x \quad (f_{\exists r}(x) \leq f_{\exists s}(x)) & \text{for all } r \sqsubseteq s \in RI \text{ and} \\ \forall x \quad (f_{\exists r_1}(f_{\exists r_2}(x)) \leq f_{\exists s}(x)) & \text{for all } r_1 \circ r_2 \sqsubseteq s \in RI. \end{array}$$

We will denote by $\mathsf{SLO}(\Sigma, RI_a) = \mathsf{SLat} \cup \bigcup_{f \in \Sigma} \mathsf{Mon}(f) \cup RI_a$ the theory of $\wedge$-semilattices with monotone operators in a set $\Sigma = \{f_{\exists r} \mid r \in N_R\}$ in which all axioms in RI_a hold.

Theorem 5.3 ([95]). *If the only concept constructors are intersection and existential restriction, then for all concept descriptions D_1, D_2 and every $\mathcal{EL}^+$ CBox $\mathcal{C}$=GCI$\cup$RI, where RI consists of role inclusions of the form $r \sqsubseteq s$ and $r_1 \circ r_2 \sqsubseteq s$, with concept names $N_C = \{C_1, \ldots, C_n\}$ and set of roles N_R the following are equivalent:*

(1) $D_1 \sqsubseteq_{\mathcal{C}} D_2$.

(2) $\mathsf{SLO}(\Sigma, RI_a) \models \forall C_1 \ldots C_n \left(\left(\bigwedge_{C \sqsubseteq D \in GCI} \overline{C} \leq \overline{D} \right) \rightarrow \overline{D_1} \leq \overline{D_2} \right)$.

Locality Properties for $\mathcal{EL}$ and $\mathcal{EL}^+$

In [95, 99] it was shown that the uniform word problem for the class of algebras $\mathsf{SLO}(\Sigma)$ is decidable in PTIME. For this, it was proved that $\mathsf{SLO}(\Sigma)$ can be seen as a Ψ-local extension of the theory SLat of semilattices.

Theorem 5.4 ([101, 95]). *Let G be a set of ground clauses. The following are equivalent:*

(1) $\mathsf{SLat} \cup \mathsf{Mon}(\Sigma) \cup G \models \bot$.

(2) $\mathsf{SLat} \cup \mathsf{Mon}(\Sigma)[G] \cup G$ *has no partial model A such that its $\{\wedge\}$-reduct is a semilattice, and all Σ-subterms of G are defined.*

Here we denote by $\mathsf{Mon}(\Sigma)[G]$ the set of all instances of axioms of $\mathsf{Mon}(\Sigma)$ containing only (ground) subterms occurring in G.

Let $\mathsf{Mon}(\Sigma)[G]_0 \cup G_0 \cup \mathsf{Def}$ be obtained from $\mathsf{Mon}(\Sigma)[G] \cup G$ by replacing (in a bottom-up manner) every term $t = f(c)$ starting with functions in Σ with a fresh constant c_t, and adding $t \approx c_t$ to the set Def.

The following are equivalent (and equivalent to (1) and (2) above):

(3) $\mathsf{Mon}(\Sigma)[G]_0 \cup G_0 \cup \mathsf{Def}$ *has no partial model A such that its $\{\wedge\}$-reduct is a semilattice, and all Σ-subterms of G are defined.*

(4) $\mathsf{Mon}(\Sigma)[G]_0 \cup G_0$ *is unsatisfiable in SLat.*

(In the presence of $\mathsf{Mon}(\Sigma)$ the instances $\mathsf{Con}[G]_0$ of the congruence axioms for functions in Σ, $\mathsf{Con}[G]_0 = \{g \approx g' \rightarrow c_{f(g)} \approx c_{f(g')} \mid f(g) \approx c_{f(g)}, f(g') \approx c_{f(g')} \in \mathsf{Def}\}$, are not necessary.)

The equivalences described in Theorem 5.4 allow us to hierarchically reduce, in polynomial time, proof tasks in $\mathsf{SLat} \cup \mathsf{Mon}(\Sigma)$ to proof tasks in SLat, which can then be solved in polynomial time.

Example 5.5. We illustrate the method on an example first considered in [4]. This example was also discussed in [95, 97]. Consider the $\mathcal{EL}$ TBox $\mathcal{T}$ consisting of the following definitions:

$$\begin{array}{lcl} A_1 & = & P_1 \sqcap A_2 \sqcap \exists r_1.\exists r_2.A_3 \\ A_2 & = & P_2 \sqcap A_3 \sqcap \exists r_2.\exists r_1.A_1 \\ A_3 & = & P_3 \sqcap A_2 \sqcap \exists r_1.(P_1 \sqcap P_2) \end{array}$$

We want to prove that $P_3 \sqcap A_2 \sqcap \exists r_1.(A_1 \sqcap A_2) \sqsubseteq_{\mathcal{T}} A_3$.

We translate this subsumption problem to the following satisfiability problem:

$$\mathsf{SLat} \cup \mathsf{Mon}(f_1, f_2) \cup G \models \bot$$

where G is the following set of ground clauses obtained from the TBox $\mathcal{T}$ and the negation of the subsumption to be proved:

$$\begin{array}{rl} G = & \{\, a_1 \approx (p_1 \wedge a_2 \wedge f_1(f_2(a_3))), \\ & \;\; a_2 \approx (p_2 \wedge a_3 \wedge f_2(f_1(a_1))), \\ & \;\; a_3 \approx (p_3 \wedge a_2 \wedge f_1(p_1 \wedge p_2)), \\ & \;\; \neg(p_3 \wedge a_2 \wedge f_1(a_1 \wedge a_2) \leq a_3)\}. \end{array}$$

We proceed as follows: We flatten and purify the set G of ground clauses by introducing new names for the terms starting with the function symbols f_1 or f_2 (we here use f_1 and f_2 as abbreviations for $f_{\exists r_1}$ and $f_{\exists r_2}$, respectively). Let Def be the corresponding set of definitions. We then take into account only those instances of the monotonicity and congruence axioms for f_1 and f_2 which correspond to the instances in Def, and purify them as well by replacing the terms themselves with the constants which denote them. We obtain the following separated set of formulae:

Def	$G_0 \quad \cup$	$(\mathsf{Mon}(f_1, f_2)[G])_0 \cup \mathsf{Con}[G]_0$
$f_2(a_3) \approx c_1$	$(a_1 \approx p_1 \wedge a_2 \wedge c_2)$	$a_1 R c_1 \rightarrow c_3 R c_2, \quad R \in \{\leq, \geq, \approx\}$
$f_1(c_1) \approx c_2$	$(a_2 \approx p_2 \wedge a_3 \wedge c_4)$	$a_3 R c_3 \rightarrow c_1 R c_4, \quad R \in \{\leq, \geq, \approx\}$
$f_1(a_1) \approx c_3$	$(a_3 \approx p_3 \wedge a_2 \wedge d_1)$	$a_1 R e_1 \rightarrow c_3 R d_1, \quad R \in \{\leq, \geq, \approx\}$
$f_2(c_3) \approx c_4$	$(p_3 \wedge a_2 \wedge d_2 \not\leq a_3)$	$a_1 R e_2 \rightarrow c_3 R d_2, \quad R \in \{\leq, \geq, \approx\}$
$f_1(e_1) \approx d_1$	$p_1 \wedge p_2 \approx e_1$	$c_1 R e_1 \rightarrow c_2 R d_1, \quad R \in \{\leq, \geq, \approx\}$
$f_1(e_2) \approx d_2$	$a_1 \wedge a_2 \approx e_2$	$c_1 R e_2 \rightarrow c_2 R d_2, \quad R \in \{\leq, \geq, \approx\}$
		$e_1 R e_2 \rightarrow d_1 R d_2, \quad R \in \{\leq, \geq, \approx\}$

Then, by Theorem 5.4, it follows that the subsumption $P_3 \sqcap A_2 \sqcap \exists r_1.(A_1 \sqcap A_2) \sqsubseteq_{\mathcal{T}} A_3$ holds if and only if $G_0 \cup (\mathsf{Mon}(f_1, f_2)[G])_0 \cup \mathsf{Con}[G]_0$ is unsatisfiable w.r.t. the theory of semilattices. Unsatisfiability can be checked by hand as follows: Note that $a_1 \wedge a_2 \leq p_1 \wedge p_2$, i.e. $e_2 \leq e_1$. Then using monotonicity instance $e_2 \leq e_1 \rightarrow d_2 \leq d_1$ it follows that $d_2 \leq d_1$, so $p_3 \wedge a_2 \wedge d_2 \leq p_3 \wedge a_2 \wedge d_1 \approx a_3$, which is a contradiction to $p_3 \wedge a_2 \wedge d_2 \not\leq a_3$. ■

In [95, 99] it was proved that similar results hold for the class $SLO(\Sigma, RI_\alpha)$ of semilattices with monotone operators in a set Σ satisfying a family RI_a of axioms of the form

$$\forall x \qquad g(x) \leq h(x), \qquad (5.1)$$

$$\forall x \quad f(g(x)) \leq h(x), \qquad (5.2)$$

as well as their flattened version RI_a^{flat}, in which (5.2) is replaced by

$$\forall x, y \quad y \leq g(x) \rightarrow f(y) \leq h(x). \qquad (5.3)$$

In this case we need more instances, which can be described using a suitable closure operator.

We define the following closure operator Ψ_{RI} on a ground set G:

$$\begin{aligned}
\Psi_{RI}(G) &= \bigcup_{i\geq 0} \Psi^i_{RI}(G), \text{ with}\\
\Psi^0_{RI}(G) &= \mathsf{est}(G), \text{ and}\\
\Psi^{i+1}_{RI}(G) &= \{h(c) \mid \forall x(g(x) \leq h(x)) \in RI_a^{\mathsf{flat}} \text{ and } g(c) \in \Psi^i_{RI}(G)\} \cup\\
&\quad \{g(c) \mid \forall x(g(x) \leq h(x)) \in RI_a^{\mathsf{flat}} \text{ and } h(c) \in \Psi^i_{RI}(G)\} \cup\\
&\quad \{h(c) \mid \forall x, y(y \leq g(x) \rightarrow f(y) \leq h(x)) \in RI_a^{\mathsf{flat}} \text{ and } g(c) \in \Psi^i_{RI}(G)\} \cup\\
&\quad \{g(c) \mid \forall x, y(y \leq g(x) \rightarrow f(y) \leq h(x)) \in RI_a^{\mathsf{flat}} \text{ and } h(c) \in \Psi^i_{RI}(G)\}.
\end{aligned}$$

The following theorem states that the extension of the theory of semilattices with monotone operators and role inclusions is Ψ_{RI}-local.

Theorem 5.6. *$\mathsf{SLat} \cup \mathsf{Mon}(\Sigma) \cup RI_a^{\mathsf{flat}}$ is a Ψ_{RI}-local theory extension of SLat.*

Proof: By Theorem 2.19 we know that $(\mathsf{Emb}_{\mathsf{f}}^{\Psi})$ implies $(\mathsf{Loc}_{\mathsf{f}}^{\Psi})$ for flat and linear clauses. In RI_a^{flat} we have clauses of type 5.1 and 5.3, which both are not linear. However, the closure operator Ψ_{RI} satisifes for clauses of type 5.1 and 5.3 the condition that for all constants c, if $g(c) \in \Psi_{RI}(G)$, then $h(c) \in \Psi_{RI}(G)$, and vice versa. Therefore, by Remark 2.20, we can conclude that $(\mathsf{Emb}_{\mathsf{f}}^{\Psi})$ implies $(\mathsf{Loc}_{\mathsf{f}}^{\Psi})$ for the closure operator Ψ_{RI}. In Lemma 4.5 from [99] it was shown that every semilattice $P = (S, \wedge, \{f\}_{f\in\Sigma})$ with partially defined monotone operators satisfying the axioms in RI_a^{flat}, and with the property that if a variable occurs in two terms $g(x)$ and $h(x)$ in a clause in RI_a^{flat}, then for every $s \in S$, $g(s)$ is defined if and only if $h(s)$ is defined, weakly embeds into a semilattice with totally defined operators satisfying RI_a^{flat}. Therefore, $\mathsf{SLat} \cup \mathsf{Mon}(\Sigma) \cup RI_a^{\mathsf{flat}}$ satisfies property $(\mathsf{Emb}_{\mathsf{f}}^{\Psi})$, thus it satisfies property $(\mathsf{Loc}_{\mathsf{f}}^{\Psi})$ (for $\Psi = \Psi_{RI}$). □

Example 5.7. We illustrate the ideas on an example presented in [9] (here slightly simplified). This example was also considered in [95, 97], but there a different locality notion was used. We present here the way Ψ-locality can be used in this context.

Consider the CBox $\mathcal{C}$ consisting of the following general concept inclusions and role inclusions:

$$\begin{aligned}
\mathsf{Endocard} &\sqsubseteq \mathsf{Tissue} \sqcap \exists \mathsf{cont\text{-}in}.\mathsf{HeartWall} \sqcap \exists \mathsf{cont\text{-}in}.\mathsf{HeartValve}\\
\mathsf{HeartWall} &\sqsubseteq \exists \mathsf{part\text{-}of}.\mathsf{Heart}\\
\mathsf{HeartValve} &\sqsubseteq \exists \mathsf{part\text{-}of}.\mathsf{Heart}\\
\mathsf{Endocarditis} &\sqsubseteq \mathsf{Inflammation} \sqcap \exists \mathsf{has\text{-}loc}.\mathsf{Endocard}\\
\mathsf{Inflammation} &\sqsubseteq \mathsf{Disease}\\
\mathsf{HeartDisease} &= \mathsf{Disease} \sqcap \exists \mathsf{has\text{-}loc}.\mathsf{Heart}
\end{aligned}$$

$$\begin{aligned}
\mathsf{part\text{-}of} \circ \mathsf{part\text{-}of} &\sqsubseteq \mathsf{part\text{-}of}\\
\mathsf{part\text{-}of} &\sqsubseteq \mathsf{cont\text{-}in}\\
\mathsf{has\text{-}loc} \circ \mathsf{cont\text{-}in} &\sqsubseteq \mathsf{has\text{-}loc}
\end{aligned}$$

We want to check whether $\mathsf{Endocarditis} \sqsubseteq_{\mathcal{C}} \mathsf{Heartdisease}$.

We first introduce abbreviations for the other concepts to make the formulae shorter:

$$\begin{aligned} f_{\mathsf{ci}} &= f_{\exists\mathsf{cont\text{-}in}} & e &= \mathsf{Endocard} \\ f_{\mathsf{po}} &= f_{\exists\mathsf{part\text{-}of}} & h &= \mathsf{Heart} \\ f_{\mathsf{hl}} &= f_{\exists\mathsf{has\text{-}loc}} & t &= \mathsf{Tissue} \\ h_w &= \mathsf{HeartWall} & i &= \mathsf{Inflammation} \\ h_v &= \mathsf{HeartValve} & d &= \mathsf{Disease} \end{aligned}$$

From the three role inclusions we get the set $RI_a^{\mathsf{flat}} = \{R_1, R_2, R_3\}$ of (flattened) role inclusion axioms, where

$$\begin{aligned} R_1 &= \forall x, y \;\; y \leq f_{\mathsf{po}}(x) \rightarrow f_{\mathsf{po}}(y) \leq f_{\mathsf{po}}(x), \\ R_2 &= \forall x \;\; f_{\mathsf{po}}(x) \leq f_{\mathsf{ci}}(x), \\ R_3 &= \forall x, y \;\; y \leq f_{\mathsf{ci}}(x) \rightarrow f_{\mathsf{hl}}(y) \leq f_{\mathsf{hl}}(x). \end{aligned}$$

$\mathsf{Endocarditis} \sqsubseteq_{\mathcal{C}} \mathsf{HeartDisease}$ holds if and only if

$$\begin{aligned} &\mathsf{SLat} \;\cup\; \mathsf{Mon}(f_{\mathsf{ci}}, f_{\mathsf{hl}}, f_{\mathsf{po}}) \;\cup\; \{\forall x, y \;\; y \leq f_{\mathsf{po}}(x) \rightarrow f_{\mathsf{po}}(y) \leq f_{\mathsf{po}}(x), \\ &\qquad\qquad\qquad\qquad\quad \forall x \;\; f_{\mathsf{po}}(x) \leq f_{\mathsf{ci}}(x), \\ &\qquad\qquad\qquad\qquad\quad \forall x, y \;\; y \leq f_{\mathsf{ci}}(x) \rightarrow f_{\mathsf{hl}}(y) \leq f_{\mathsf{hl}}(x)\} \\ &\quad\cup\; \{e \leq (t \wedge f_{\mathsf{ci}}(h_w) \wedge f_{\mathsf{ci}}(h_v)), \;\; h_w \leq f_{\mathsf{po}}(h), \;\; h_v \leq f_{\mathsf{po}}(h), \\ &\qquad \mathsf{Endocarditis} \leq (i \wedge f_{\mathsf{hl}}(e)), \;\; i \leq d, \;\; \mathsf{HeartDisease} \approx (d \wedge f_{\mathsf{hl}}(h)), \\ &\qquad \mathsf{Endocarditis} \not\leq \mathsf{HeartDisease}\} \;\; \models \;\; \bot \,. \end{aligned}$$

We use the closure operator Ψ_{RI} to compute the set of ground terms needed for instantiation of the axioms in RI_a^{flat}.

$$\Psi_{RI}^0 = \mathsf{est}(\mathcal{K}, G) = \{\; f_{\mathsf{ci}}(h_w), f_{\mathsf{ci}}(h_v), f_{\mathsf{po}}(h), f_{\mathsf{hl}}(e), f_{\mathsf{hl}}(h) \;\}$$

From $R_2 \in RI_a^{\mathsf{flat}}$ and $f_{\mathsf{po}}(h) \in \Psi_{RI}^0$ it follows that $f_{\mathsf{ci}}(h) \in \Psi_{RI}^1$.

From $R_2 \in RI_a^{\mathsf{flat}}$ and $f_{\mathsf{ci}}(h_w), f_{\mathsf{ci}}(h_v) \in \Psi_{RI}^0$ it follows that $f_{\mathsf{po}}(h_w), f_{\mathsf{po}}(h_v) \in \Psi_{RI}^1$.

From $R_3 \in RI_a^{\mathsf{flat}}$ and $f_{\mathsf{ci}}(h_w), f_{\mathsf{ci}}(h_v) \in \Psi_{RI}^0$ it follows that $f_{\mathsf{hl}}(h_w), f_{\mathsf{hl}}(h_v) \in \Psi_{RI}^1$.

From $R_3 \in RI_a^{\mathsf{flat}}$ and $f_{\mathsf{hl}}(e), f_{\mathsf{hl}}(h) \in \Psi_{RI}^0$ it follows that $f_{\mathsf{ci}}(e), f_{\mathsf{ci}}(h) \in \Psi_{RI}^1$.

$$\begin{aligned} \Psi_{RI}^1 = \{\; & f_{\mathsf{ci}}(h_w), f_{\mathsf{ci}}(h_v), f_{\mathsf{ci}}(h), f_{\mathsf{ci}}(e), \\ & f_{\mathsf{po}}(h_w), f_{\mathsf{po}}(h_v), f_{\mathsf{po}}(h), \\ & f_{\mathsf{hl}}(h_w), f_{\mathsf{hl}}(h_v), f_{\mathsf{hl}}(h), f_{\mathsf{hl}}(e) \;\}. \end{aligned}$$

From $R_2 \in RI_a^{\mathsf{flat}}$ and $f_{\mathsf{ci}}(e) \in \Psi_{RI}^1$ it follows that $f_{\mathsf{po}}(e) \in \Psi_{RI}^2$. No other new terms are added in this step.

$$\begin{aligned} \Psi_{RI}^2 = \{\; & f_{\mathsf{ci}}(h_w), f_{\mathsf{ci}}(h_v), f_{\mathsf{ci}}(h), f_{\mathsf{ci}}(e), \\ & f_{\mathsf{po}}(h_w), f_{\mathsf{po}}(h_v), f_{\mathsf{po}}(h), f_{\mathsf{po}}(e), \\ & f_{\mathsf{hl}}(h_w), f_{\mathsf{hl}}(h_v), f_{\mathsf{hl}}(h), f_{\mathsf{hl}}(e) \;\}. \end{aligned}$$

In the next step no new terms are added. From $\Psi_{RI}^3 = \Psi_{RI}^2$ it follows that $\Psi_{RI}(G) = \Psi_{RI}^2$. After computing $(RI_a \cup \mathsf{Mon}(f_{\mathsf{ci}}, f_{\mathsf{hl}}, f_{\mathsf{po}}) \cup \mathsf{Con})[\Psi_{RI}(G)]$ we obtain:

G	$(RI_a \cup \mathsf{Mon} \cup \mathsf{Con})[\Psi_{RI}(G)]$	
$e \leq (t \wedge f_{\mathsf{ci}}(h_w) \wedge f_{\mathsf{ci}}(h_v))$	$y \leq f_{\mathsf{po}}(x) \rightarrow f_{\mathsf{po}}(y) \leq f_{\mathsf{po}}(x)$	for $x, y \in \{h_v, h_w, h, e\}$
$h_w \leq f_{\mathsf{po}}(h)$	$f_{\mathsf{po}}(x) \leq f_{\mathsf{ci}}(x)$	for $x \in \{h_v, h_w, h, e\}$
$h_v \leq f_{\mathsf{po}}(h)$	$y \leq f_{\mathsf{ci}}(x) \rightarrow f_{\mathsf{hl}}(y) \leq f_{\mathsf{hl}}(x)$	for $x, y \in \{h_v, h_w, h, e\}$
$\mathsf{Endocarditis} \leq (i \wedge f_{\mathsf{hl}}(e))$		
$i \leq d$	$xRy \rightarrow f_{\mathsf{ci}}(x) R f_{\mathsf{ci}}(y)$	for $x, y \in \{h_w, h_v, h, e\}$
$\mathsf{HeartDisease} \approx (d \wedge f_{\mathsf{hl}}(h))$	$xRy \rightarrow f_{\mathsf{hl}}(x) R f_{\mathsf{hl}}(y)$	for $x, y \in \{h_v, h_w, h, e\}$
$\mathsf{Endocarditis} \not\leq \mathsf{HeartDisease}$	$R \in \{\leq, \geq\}$	$x \neq y$

We can simplify the problem even further by replacing the ground terms in $\Psi_{RI}(G)$ with new constants and taking into account the corresponding definitions $c_t \approx t$. Let the set of clauses obtained this way be $(RI_a \cup \mathsf{Mon} \cup \mathsf{Con})[\Psi_{RI}(G)]_0$.

G_0	$(RI_a \cup \mathsf{Mon} \cup \mathsf{Con})[\Psi_{RI}(G)]_0$	
$e \leq (t \wedge c_{f_{\mathsf{ci}}(h_w)} \wedge c_{f_{\mathsf{ci}}(h_v)})$	$y \leq c_{f_{\mathsf{po}}(x)} \rightarrow c_{f_{\mathsf{po}}(y)} \leq c_{f_{\mathsf{po}}(x)}$	for $x, y \in \{h_v, h_w, h, e\}$
$h_w \leq c_{f_{\mathsf{po}}(h)}$	$c_{f_{\mathsf{po}}(x)} \leq c_{f_{\mathsf{ci}}(x)}$	for $x \in \{h_v, h_w, h, e\}$
$h_v \leq c_{f_{\mathsf{po}}(h)}$	$y \leq c_{f_{\mathsf{ci}}(x)} \rightarrow c_{f_{\mathsf{hl}}(y)} \leq c_{f_{\mathsf{hl}}(x)}$	for $x, y \in \{h_v, h_w, h, e\}$
$\mathsf{Endocarditis} \leq (i \wedge c_{f_{\mathsf{hl}}(e)})$		
$i \leq d$	$xRy \rightarrow c_{f_{\mathsf{ci}}(x)} R c_{f_{\mathsf{ci}}(y)}$	for $x, y \in \{h_w, h_v, h\}$
$\mathsf{HeartDisease} \approx (d \wedge c_{f_{\mathsf{hl}}(h)})$	$xRy \rightarrow c_{f_{\mathsf{hl}}(x)} R c_{f_{\mathsf{hl}}(y)}$	for $x, y \in \{h_v, h_w, h, e\}$
$\mathsf{Endocarditis} \not\leq \mathsf{HeartDisease}$	$R \in \{\leq, \geq\}$	$x \neq y$

With the notation in the previous table, $\mathsf{Endocarditis} \sqsubseteq_{\mathcal{C}} \mathsf{HeartDisease}$ if and only if it holds that $G_0 \cup (RI_a \cup \mathsf{Mon} \cup \mathsf{Con})[\Psi(G)]_0 \models_{\mathsf{SLat}} \bot$. Unsatisfiability can be proved as follows:

(1) From $\mathsf{Endocarditis} \leq (i \wedge c_{f_{\mathsf{hl}}(e)})$ it follows that $\mathsf{Endocarditis} \leq i$.

(2) From $\mathsf{Endocarditis} \leq i$ and $i \leq d$ it follows that $\mathsf{Endocarditis} \leq d$.

(3) From $\mathsf{Endocarditis} \leq (i \wedge c_{f_{\mathsf{hl}}(e)})$ it follows that $\mathsf{Endocarditis} \leq c_{f_{\mathsf{hl}}(e)}$.

(4) From $e \leq (t \wedge c_{f_{\mathsf{ci}}(h_w)} \wedge c_{f_{\mathsf{ci}}(h_v)})$ it follows that $e \leq c_{f_{\mathsf{ci}}(h_v)}$.

(5) From $e \leq c_{f_{\mathsf{ci}}(h_v)}$ and an instance of the role inclusion axiom R_3 it follows that $c_{f_{\mathsf{hl}}(e)} \leq c_{f_{\mathsf{hl}}(h_v)}$.

(6) From an instance of the role inclusion axiom R_2 it follows that $c_{f_{\mathsf{po}}(h)} \leq c_{f_{\mathsf{ci}}(h)}$.

(7) From $h_v \leq c_{f_{\mathsf{po}}(h)}$ and $c_{f_{\mathsf{po}}(h)} \leq c_{f_{\mathsf{ci}}(h)}$ it follows that $h_v \leq c_{f_{\mathsf{ci}}(h)}$.

(8) From $h_v \leq c_{f_{\mathsf{ci}}(h)}$ and an instance of the role inclusion axiom R_2 it follows that $c_{f_{\mathsf{hl}}(h_v)} \leq c_{f_{\mathsf{hl}}(h)}$.

(9) From $\mathsf{Endocarditis} \leq c_{f_{\mathsf{hl}}(e)}$, $c_{f_{\mathsf{hl}}(e)} \leq c_{f_{\mathsf{hl}}(h_v)}$ and $c_{f_{\mathsf{hl}}(h_v)} \leq c_{f_{\mathsf{hl}}(h)}$ it follows that $\mathsf{Endocarditis} \leq c_{f_{\mathsf{hl}}(h)}$.

(10) From $\mathsf{Endocarditis} \leq d$, $\mathsf{Endocarditis} \leq c_{f_{\mathsf{hl}}(h)}$ and $\mathsf{HeartDisease} \approx (d \wedge c_{f_{\mathsf{hl}}(h)})$ it follows that $\mathsf{Endocarditis} \leq \mathsf{HeartDisease}$.

Thus, we have obtained a contradiction to $\mathsf{Endocarditis} \not\leq \mathsf{HeartDisease}$. ■

5.2 P-Interpolation Property

Let Pred be a set of predicates and let $P \subseteq$ Pred. We look at a form of interpolation property which we call P-interpolating [91]. If P consists of one predicate only, e.g. $P = \{\leq\}$, we may omit the set notation and refer to the property simply as $\leq$-interpolation instead of $\{\leq\}$-interpolation. In the following we give a definition for P-interpolation and prove that the theory of semilattices has $\leq$-interpolation. For this, we use a form of symbol elimination based on positive unit resolution. We afterwards show that the $\leq$-interpolation property extends to $\mathsf{SLO}(\Sigma, RI_\alpha)$.

Definition 5.8 ((Strong) P-interpolation). *A theory $\mathcal{T}_0$ is* P-interpolating *with respect to $P \subseteq$ Pred if for all conjunctions A and B of ground literals, all binary predicates $R \in P$ and all terms a and b such that a contains only constants occurring in A and b contains only constants occurring in B (or vice versa), the following holds:*

If $A \wedge B \models_{\mathcal{T}_0} aRb$, then there exists a term t containing only constants common to A and B with $A \wedge B \models_{\mathcal{T}_0} aRt \wedge tRb$.

$\mathcal{T}_0$ is strongly P-interpolating *if there exists such a term t with $A \models_{\mathcal{T}_0} aRt$ and $B \models_{\mathcal{T}_0} tRb$.*[1]

Proving P-interpolation is sometimes easier for theories which are convex.

Definition 5.9 (Convexity). *A theory $\mathcal{T}_0$ is* convex *with respect to the set Pred of all predicates (which may include also equality $\approx$) if for all conjunctions Γ of ground atoms, relations $R_1, \ldots, R_m \in$ Pred and ground tuples $\bar{t}_1, \ldots, \bar{t}_m$ of corresponding arity, the following holds:*

If $\Gamma \models_{\mathcal{T}_0} \bigvee_{i=1}^{m} R_i(\bar{t}_i)$, then there exists $j \in \{1, \ldots, m\}$ such that $\Gamma \models_{\mathcal{T}_0} R_j(\bar{t}_j)$.

We prove that the theory of semilattices with monotone operators is $\leq$-interpolating. For this we use the fact that the theory of semilattices is $\leq$-convex.

Lemma 5.10. *The theory of semilattices is convex w.r.t. $P = \{\approx, \leq\}$.*

Proof: The convexity of the theory of semilattices w.r.t. $\approx$ follows from the fact that this is an equational class; convexity w.r.t. $\leq$ follows from the fact that $x \leq y$ if and only if $(x \wedge y) \approx x$. □

Theorem 5.11. *If 0 is not included in the signature, then the theory SLat of semilattices is $\leq$-interpolating.*

Proof: This is a constructive proof based on the fact that $\mathsf{SLat} = ISP(S_2)$, where S_2 is the 2-element semilattice. We prove that the theory of semilattices (without 0 in its signature) is $\leq$-interpolating, i.e. we show that if A and B are two conjunctions of literals and $A \wedge B \models_{\mathsf{SLat}} a \leq b$, where a is a term containing only constants which occur in A and b a term containing only constants occurring in B, then there exists a term containing only common constants in A and B such that $A \wedge B \models_{\mathsf{SLat}} a \leq t$ and $A \wedge B \models_{\mathsf{SLat}} t \leq b$. We can assume without loss of generality that A and B consist only of atoms: Indeed, assume that $A \wedge B = A_1 \wedge \cdots \wedge A_n \wedge \neg A'_1 \wedge \cdots \wedge \neg A'_m$, where $A_1, \ldots, A_n, A'_1, \ldots, A'_m$ are atoms, then the following are equivalent:

[1] This definition is equivalent to the definition, sometimes used in the literature, in which a and b are required to be constants.

- $A \wedge B \models_{\mathsf{SLat}} a \leq b$
- $\models_{\mathsf{SLat}} A \wedge B \rightarrow a \leq b$
- $\models_{\mathsf{SLat}} \neg A_1 \vee \cdots \vee \neg A_n \vee A'_1 \vee \cdots \vee A'_m \vee a \leq b$
- $\models_{\mathsf{SLat}} (A_1 \wedge \cdots \wedge A_n) \rightarrow A'_1 \vee \cdots \vee A'_m \vee a \leq b$
- $A_1 \wedge \cdots \wedge A_n \models_{\mathsf{SLat}} A'_1 \vee \cdots \vee A'_m \vee a \leq b$

Since the theory of semilattices is convex w.r.t. $\leq$ and $\approx$, it follows that if $A \wedge B \models_{\mathsf{SLat}} a \leq b$, then either

(a) $A_1 \wedge \cdots \wedge A_n \models_{\mathsf{SLat}} A'_j$ for some $j \in \{1, \ldots, m\}$, or

(b) $A_1 \wedge \cdots \wedge A_n \models_{\mathsf{SLat}} a \leq b$.

It is easy to see that in case (a), $A \wedge B \models \bot$. Then the conclusion of the theorem follows immediately. We therefore continue the proof for the case when A and B consist only of atoms.

As $\mathsf{SLat} = ISP(S_2)$, in SLat the same Horn sentences are true as in the 2-element semilattice S_2. Thus, $A \wedge B \models_{\mathsf{SLat}} a \leq b$ if and only if $A \wedge B \models_{S_2} a \leq b$, so we can reduce such a test to entailment in propositional logic.

It follows that $A \wedge B \models_{\mathsf{SLat}} a \leq b$ if and only if the conjunction $N_A \wedge N_B \wedge P_a \wedge \neg P_b$ of literals in propositional logic is unsatisfiable, where:

$$N_A : \begin{cases} P_{e_1 \wedge e_2} \leftrightarrow P_{e_1} \wedge P_{e_2} \\ P_{e_1} \leftrightarrow P_{e_2} \quad e_1 \approx e_2 \in A \\ P_{e_1} \rightarrow P_{e_2} \quad e_1 \leq e_2 \in A \\ \text{for all } e_1, e_2 \text{ subterms in } A \end{cases}$$

$$N_B : \begin{cases} P_{g_1 \wedge g_2} \leftrightarrow P_{g_1} \wedge P_{g_2} \\ P_{g_1} \leftrightarrow P_{g_2} \quad g_1 \approx g_2 \in B \\ P_{g_1} \rightarrow P_{g_2} \quad g_1 \leq g_2 \in B \\ \text{for all } g_1, g_2 \text{ subterms in } B \end{cases}$$

We obtain an unsatisfiable set of clauses $(N_A \wedge P_a) \wedge (N_B \wedge \neg P_b) \models \bot$, where N_A and N_B are sets of Horn clauses. We can saturate $N_A \wedge N_B \wedge P_a$ under positive unit resolution, i.e. a form of resolution in which an inference is only allowed if one of the clauses is a positive unit clause. We show that if $A \wedge B \models_{\mathsf{SLat}} a \leq b$ holds, then for the term

$$t \coloneqq \bigwedge \{c \mid A \wedge B \models_{\mathsf{Slat}} a \leq c, \text{ where } c \text{ is a common subterm of } A \text{ and } B\}$$

the following hold:

(i) $A \wedge B \models_{\mathsf{SLat}} a \leq t$, and

(ii) $A \wedge B \models_{\mathsf{SLat}} t \leq b$.

Every $c \in T = \{c \mid A \wedge B \models_{\mathsf{SLat}} a \leq c, \text{ where } c \text{ is a common subterm of } A \text{ and } B\}$ corresponds to the positive unit clause P_c (where P_c is a propositional variable common to N_A and N_B) which can be derived from $N_A \wedge N_B$ using positive unit resolution. Let $T = \{c_1, ..., c_k\}$, i.e. $t = c_1 \wedge ... \wedge c_k$.

It is clearly the case that $A \wedge B \models_{\mathsf{SLat}} a \leq t$, because $N_A \wedge N_B \wedge P_a \wedge \neg P_t \wedge (P_t \leftrightarrow \bigwedge_{c \in T} P_c)$ is unsatisfiable. Thus, (i) holds.

For proving (ii), we have to show that $N_A \wedge N_B \wedge P_{c_1} \wedge ... \wedge P_{c_k} \models P_b$. It is known that $N_A \wedge N_B \wedge P_a \wedge \neg P_b \models \bot$. We analyze which clauses are derived from $N_A \wedge N_B \wedge P_a$ by resolution.

Note first that, since N_A, N_B and P_a are Horn clauses, we do not have to consider all the resolvents obtained by resolution, but only those obtained from inferences with positive unit clauses (this follows from a result in [59], stating that for an unsatisfiable set of Horn clauses there always exists a resolution proof using only inferences in which one clause is a positive unit clause). We therefore analyze how $\bot$ can be derived by positive unit resolution.

Assume that N_A contains the following unit clauses:

- $P_{a_1}, ..., P_{a_{n_1}}$, where $a_1, ..., a_{n_1}$ are terms only occurring in A, and
- $P_{c_1}, ..., P_{c_{m_1}}$, where and $c_1, ..., c_{m_1}$ are terms common to A and B.

Assume that N_B contains the following unit clauses:

- $P_{b_1}, ..., P_{b_{r_1}}$, where $b_1, ..., b_{r_1}$ are terms only occurring in B, and
- $P_{c_{m_1+1}}, ..., P_{c_{m_2}}$, where $m_1 \leq m_2$ and $c_{m_1+1}, ..., c_{m_2}$ are terms common to A and B.

In a first step we consider only inferences of unit clauses with clauses in N_A. Since clauses in N_A cannot contain any of the atoms $P_{b_1}, ..., P_{b_{r_1}}$, at first only inferences between $P_a, P_{a_1}, ..., P_{a_{n_1}}, P_{c_1}, ..., P_{c_{m_2}}$ and clauses in N_A are possible, and later also inferences between newly derived unit clauses and clauses in N_A. By saturating under positive unit resolution $P_a \wedge N_A \wedge P_{c_{m_1+1}} \wedge ... \wedge P_{c_{m_2}}$ we obtain the following unit clauses:

- $P_{a_{n_1+1}}, ..., P_{a_{n_2}}$, where $n_1 \leq n_2$ and $a_{n_1+1}, ..., a_{n_2}$ are terms only occurring in A, and
- $P_{c_{m_2+1}}, ..., P_{c_{m_3}}$, where $m_2 \leq m_3$ and $c_{m_2+1}, ..., c_{m_3}$ are terms common to A and B.

In a next step we consider inferences of unit clauses with clauses in N_B. Since clauses in N_B cannot contain any of the atoms $P_a, P_{a_1}, ..., P_{a_{n_2}}$, at first only inferences between $P_{b_1}, ..., P_{b_{r_1}}, P_{c_1}, ..., P_{c_{m_3}}$ and clauses in N_B are possible, and later also inferences between newly derived unit clauses with clauses in N_B. By saturating under positive unit resolution $N_B \wedge P_{c_1} \wedge ... \wedge P_{c_{m_1}} \wedge P_{c_{m_2+1}} \wedge ... \wedge P_{c_{m_3}}$ we obtain the following unit clauses:

- $P_{b_{r_1+1}}, ..., P_{b_{r_2}}$, where $r_1 \leq r_2$ and $b_{r_1+1}, ..., b_{r_2}$ are terms only occurring in B, and
- $P_{c_{m_3}+1}, ..., P_{c_{m_4}}$, where $m_3 \leq m_4$ and $c_{m_3+1}, ..., c_{m_4}$ are terms common to A and B.

It can be proved that for all $i \in \{1, ..., r_2\}$ there exists a clause $(\bigwedge_{l \in J_1} P_{c_l} \wedge \bigwedge_{j \in J_2} P_{b_j}) \rightarrow P_{b_i}$, where $J_1 \subseteq \{1, ..., m_4\}$ and $J_2 \subseteq \{1, ..., i-1\}$, such that all P_{c_l} and all P_{b_j} are derived as unit clauses before P_{b_i}. It therefore holds that $N_A \wedge N_B \models \bigwedge_{s=1}^{m_4} P_{c_s} \rightarrow P_{b_t}$ for all $t \in \{1, ..., r_2\}$.

This procedure may continue, such that new "common" unit clauses, i.e. unit clauses which are atoms common to N_A and N_B, are again obtained from inferences with clauses in N_A, then again from inferences with clauses in N_B, and so on. At some point, since $N_A \wedge P_a \wedge N_B \models P_b$ holds, P_b has to be derived from an inference of a unit clause with a clause in N_B. Then, since we do not need P_a in the presence of the common unit clauses to derive P_b, we have $N_A \wedge N_B \wedge P_{c_1} \wedge ... \wedge P_{c_k} \models P_b$, where $c_1, ..., c_k$ are all the common unit clauses obtained by positive unit resolution. Thus, (ii) is proved. $\square$

Based on the ideas illustrated in the proof of Theorem 5.11 we propose Algorithm 4 for $\leq$-interpolation in SLat.

Algorithm 4 $\leq$-interpolation in SLat

Input:	Conjunctions of literals A and B; terms a and b such that $A \wedge B \models_{\mathsf{SLat}} a \leq b$
Output:	Intermediate term t containing only common constants of A and B such that $A \wedge B \models_{\mathsf{SLat}} a \leq t$ and $A \wedge B \models_{\mathsf{SLat}} t \leq b$

Step 1 Let S_A and S_B be the set of all constants appearing in A and B, respectively, and let $S_{AB} = S_A \cap S_B$, $S_A^* = S_A \setminus S_{AB}$ and $S_B^* = S_B \setminus S_{AB}$.

Step 2 Let P_a be a propositional variable corresponding to a and N_{AB} be the set of Horn clauses obtained from $A \wedge B$ using the following transformations:

- Translate every constant c to a corresponding propositional variable P_c.
- Interpret every meet operator $\wedge$ as a logical conjunction.
- Replace every symbol $\leq$ by a logical implication sign $\rightarrow$.

Step 3 Let N_{sat} be the set of clauses obtained after saturating $N_{AB} \wedge P_a$ under positive unit resolution.

Step 4 Let $t \coloneqq \bigwedge\{c \mid P_c \text{ is a positive unit clause in } N_{sat} \text{ with } c \in S_{AB}\}$.

We show in detail how to compute an intermediate term in the theory of semilattices using Algorithm 4 on the following example.

Example 5.12. Let A and B be the following two sets of formulae:

$$A = \{a_1 \leq c_1, \quad c_2 \leq a_2,\ a_2 \leq c_3 \quad\}$$
$$B = \{\quad c_1 \leq b_1,\ b_1 \leq c_2, \quad c_3 \leq b_2\}$$

It is easy to see that $A \wedge B \models a_1 \leq b_2$. We can compute an intermediate term, i.e. a term t such that $A \wedge B \models a_1 \leq t$ and $A \wedge B \models t \leq b_2$, using Algorithm 4.

Step 1: We have the following sets of constants:

$$\begin{aligned} S_A &= \{a_1, a_2, c_1, c_2, c_3\} \\ S_B &= \{b_1, b_2, c_1, c_2, c_3\} \\ S_{AB} &= \{c_1, c_2, c_3\} \\ S_A^* &= \{a_1, a_2\} \\ S_B^* &= \{b_1, b_2\} \end{aligned}$$

Step 2: Using the translation described in the algorithm we get the following set $N_{AB} \wedge P_{a_1}$ of Horn clauses:

$$\begin{aligned} N_{AB} \wedge P_{a_1} = \{&(P_{a_1} \rightarrow P_{c_1}), \\ &(P_{c_2} \rightarrow P_{a_2}), \\ &(P_{a_2} \rightarrow P_{c_3}), \\ &(P_{c_1} \rightarrow P_{b_1}), \\ &(P_{b_1} \rightarrow P_{c_2}), \\ &(P_{c_3} \rightarrow P_{b_2}), \\ &P_{a_1}\} \end{aligned}$$

Step 3: We saturate $N_{AB} \wedge P_{a_1}$ under positive unit resolution:

- Resolution of P_{a_1} and $(P_{a_1} \to P_{c_1})$ yields P_{c_1}.
- Resolution of P_{c_1} and $(P_{c_1} \to P_{b_1})$ yields P_{b_1}.
- Resolution of P_{b_1} and $(P_{b_1} \to P_{c_2})$ yields P_{c_2}.
- Resolution of P_{c_2} and $(P_{c_2} \to P_{a_2})$ yields P_{a_2}.
- Resolution of P_{a_2} and $(P_{a_2} \to P_{c_3})$ yields P_{c_3}.
- Resolution of P_{c_3} and $(P_{c_3} \to P_{b_2})$ yields P_{b_2}.

Therefore, our saturated set of clauses looks as follows:

$$\begin{aligned} N_{sat} = \{&(P_{a_1} \to P_{c_1}),\\ &(P_{c_2} \to P_{a_2}),\\ &(P_{a_2} \to P_{c_3}),\\ &(P_{c_1} \to P_{b_1}),\\ &(P_{b_1} \to P_{c_2}),\\ &(P_{c_3} \to P_{b_2}),\\ &P_{a_1}, P_{c_1}, P_{c_2}, P_{c_3}, P_{b_1}, P_{b_2}\} \end{aligned}$$

Step 4: The unit clauses in N_{sat} corresponding to a constant in S_{AB} are P_{c_1}, P_{c_2} and P_{c_3}. For $t = c_1 \wedge c_2 \wedge c_3$ the following holds:

$$\begin{aligned} A \wedge B &\models \quad a_1 \leq c_1 \wedge c_2 \wedge c_3\\ A \wedge B &\models \qquad c_1 \wedge c_2 \wedge c_3 \leq b_2 \end{aligned}$$

Therefore, $t = c_1 \wedge c_2 \wedge c_3$ is an intermediate term for $a_1 \leq b_2$. ■

For the theory $\mathsf{SLO}(\Sigma, RI_a)$ of semilattices with monotone operators satisfying axioms RI_a it is a bit more complicated to show that it is $\leq$-interpolating. Remember that in the presence of monotone operators we have monotonicity and (flattened) role inclusion axioms of the following form:

$$\begin{aligned} &\forall x, y && x \leq y \to f(x) \leq f(y)\\ &\forall x, y && y \leq g(x) \to f(y) \leq h(x) \end{aligned}$$

These axioms need to be instantiated (using the closure operator Ψ_{RI}). It can happen that x is replaced by a ground term that occurs only in A and y is replaced by a ground term that occurs only in B. We call such an instance a mixed instance. Since we need a clear distinction between the A-part and the B-part, these mixed instances need to be separated, which in fact is always possible.

Theorem 5.13. *Let $\mathcal{T}_0$ be a theory with signature $\Pi_0 = (\Sigma_0, \mathsf{Pred})$. Assume that $\leq \in \mathsf{Pred}$ is such that*

- *$\leq$ is a transitive relation in all models of $\mathcal{T}_0$,*
- *$\mathcal{T}_0$ is convex with respect to $\leq$, and*
- *$\mathcal{T}_0$ is $\leq$-interpolating.*

Let A_0 and B_0 be conjunctions of ground literals in the signature Π_0^C (the extension of Π_0 with a set C of constants) such that $A_0 \wedge B_0 \wedge \mathcal{H} \models_{\mathcal{T}_0} a \leq b$, where a contains only symbols occurring in A_0, b contains only symbols occurring in B_0, and $\mathcal{H}$ is a set of Horn clauses of the form $c_1 \leq d_1 \rightarrow c \leq d$ in the signature Π_0^C which are instances of axioms of the following type:

$$\begin{cases} x \leq g(y) & \rightarrow & f(x) \leq h(y) \\ x \leq y & \rightarrow & f(x) \leq f(y) \end{cases}$$

Then the following hold:

(1) There exists a set T of Π_0^C-terms containing only constants common to A_0 and B_0 and a term $t \in T$ such that

$$A_0 \wedge B_0 \wedge (\mathcal{H} \backslash \mathcal{H}_{mix}) \wedge \mathcal{H}_{sep} \models_{\mathcal{T}_0} a \leq t \wedge t \leq b,$$

where

$$\begin{aligned} \mathcal{H}_{mix} \;=\; & \{a_1 \leq b_1 \rightarrow a_2 \leq b_2 \in \mathcal{H} \mid a_1, a_2 \text{ constants in } A, b_1, b_2 \text{ constants in } B\} \cup \\ & \{b_1 \leq a_1 \rightarrow b_2 \leq a_2 \in \mathcal{H} \mid b_1, b_2 \text{ constants in } B, a_1, a_2 \text{ constants in } A\} \end{aligned}$$

$$\begin{aligned} \mathcal{H}_{sep} \;=\; & \{(a_1 \leq t_1 \rightarrow a_2 \leq c_{f(t_1)}) \wedge (t_1 \leq b_1 \rightarrow c_{f(t_1)} \leq b_2) \mid a_1 \leq b_1 \rightarrow a_2 \leq b_2 \in \mathcal{H}_{mix}, \\ & b_1 \approx g(e_1), b_2 \approx h(e_1) \in \mathsf{Def}_B \text{ and } a_2 \approx f(a_1) \in \mathsf{Def}_A \text{ or vice versa}, \\ & \text{and } t_1, f(t_1) \in T\} = \mathcal{H}_{sep}^A \wedge \mathcal{H}_{sep}^B \end{aligned}$$

where $c_{f(t_1)}$ are new constants in Σ_c (considered to be common) introduced for the corresponding terms $f(t_1)$.

(2) $A_0 \wedge B_0 \wedge (\mathcal{H} \backslash \mathcal{H}_{mix}) \wedge \mathcal{H}_{sep} \wedge \neg(a \leq t \wedge t \leq b)$ is logically equivalent w.r.t. $\mathcal{T}_0$ with the following separated conjunction of literals:

$$\begin{aligned} \overline{A}_0 \wedge \overline{B}_0 \wedge \neg(a \leq t \wedge t \leq b) = A_0 \;\wedge\; B_0 \;\wedge\; & \bigwedge \{c \leq d \mid \Gamma \rightarrow c \leq d \in \mathcal{H} \backslash \mathcal{H}_{mix}\} \wedge \\ & \bigwedge \{c \leq c_{f(t)} \wedge c_{f(t)} \leq d \mid (\Gamma \rightarrow c \leq c_{f(t)}) \wedge (\Gamma \rightarrow c_{f(t)} \leq d) \in \mathcal{H}_{sep}\} \wedge \\ & \neg(a \leq t \wedge t \leq b) \end{aligned}$$

Proof: We prove (1) and (2) by induction on the number of clauses in $\mathcal{H}$.

If $\mathcal{H} = \emptyset$, then the initial problem is already separated into an A-part and a B-part. We have $A_0 \wedge B_0 \models_{\mathcal{T}_0} a \leq b$ and since we assumed that $\mathcal{T}_0$ is $\leq$-interpolating, there exists a term t containing only constants common to A_0 and B_0 such that $A_0 \wedge B_0 \models_{\mathcal{T}_0} a \leq t \wedge t \leq b$ (we can choose $T = \{t\}$).

Assume that $\mathcal{H}$ contains at least one clause, and that for every $\mathcal{H}_1$ with fewer clauses and every conjunction of literals A_0', B_0' with $A_0' \wedge B_0' \wedge \mathcal{H}_1 \models_{\mathcal{T}_0} a \leq b$, (1) and (2) hold.

Let $\mathcal{D}$ be the set of all atoms $c \leq d$ occurring in premises of clauses in $\mathcal{H}$. As every model of $A_0 \wedge B_0 \wedge \bigwedge_{(c \leq d) \in \mathcal{D}} \neg(c \leq d) \wedge \neg(a \leq b)$ is also a model of $\mathcal{H} \wedge A_0 \wedge B_0 \wedge \neg(a \leq b)$ and $\mathcal{H} \wedge A_0 \wedge B_0 \wedge \neg(a \leq b) \models_{\mathcal{T}_0} \bot$, we have $A_0 \wedge B_0 \wedge \bigwedge_{(c \leq d) \in \mathcal{D}} \neg(c \leq d) \wedge \neg(a \leq b) \models_{\mathcal{T}_0} \bot$. Let $(A_0 \wedge B_0)^+$ be the conjunction of all positive literals in $A_0 \wedge B_0$, and $(A_0 \wedge B_0)^-$ be the set of all negative literals in $A_0 \wedge B_0$. Then

$$(A_0 \wedge B_0)^+ \models_{\mathcal{T}_0} \bigvee_{(c \leq d) \in \mathcal{D}} (c \leq d) \vee \bigvee_{\neg L \in (A_0 \wedge B_0)^-} L \vee (a \leq b).$$

$\mathcal{T}_0$ is convex with respect to $\leq$ and $(A_0 \wedge B_0)^+$ is a conjunction of positive literals. Therefore, either

(i) $(A_0 \wedge B_0)^+ \models L$ for some $L \in (A_0 \wedge B_0)^-$ (then $A_0 \wedge B_0$ is unsatisfiable and hence entails any atom $c_i \leq d_i$), or

(ii) $(A_0 \wedge B_0)^+ \models a \leq b$, or

(iii) there exists $(c_1 \leq d_1) \in \mathcal{D}$ such that $A_0 \wedge B_0 \models_{\mathcal{T}_0} c_1 \leq d_1$.

Case 1: $A_0 \wedge B_0$ is unsatisfiable. In this case (1) and (2) hold for $T = \{t\}$, where t is an arbitrary term over the common symbols of A_0 and B_0.

Case 2: $A_0 \wedge B_0$ is satisfiable and $(A_0 \wedge B_0)^+ \models a \leq b$. Then we can use the fact that $\mathcal{T}_0$ is $\leq$-interpolating and we are done.

Case 3: $A_0 \wedge B_0$ is satisfiable and there exists $(c_1 \leq d_1) \in \mathcal{D}$ such that $A_0 \wedge B_0 \models_{\mathcal{T}_0} c_1 \leq d_1$. Then $A_0 \wedge B_0$ is logically equivalent in $\mathcal{T}_0$ with $A_0 \wedge B_0 \wedge c_1 \leq d_1$.

Let $C = c_1 \leq d_1 \rightarrow c \leq d \in \mathcal{H}$ such that $A_0 \wedge B_0 \models c_1 \leq d_1$.

Case 3a: Assume that C contains only constants occurring in A or only constants occurring in B. Then $A_0 \wedge B_0 \wedge \mathcal{H}$ is equivalent w.r.t. $\mathcal{T}_0$ with $A_0 \wedge B_0 \wedge (\mathcal{H} \backslash \{C\}) \wedge c \leq d$. By the induction hypothesis for $A_0' \wedge B_0' = A_0 \wedge B_0 \wedge c \leq d$ and $\mathcal{H}_1 = \mathcal{H} \backslash \{C\}$, we know that there exists T' and $t \in T'$ such that $A_0' \wedge B_0' \wedge (\mathcal{H}_1 \backslash \mathcal{H}_{1\mathsf{mix}}) \wedge \mathcal{H}_{1\mathsf{sep}} \models a \leq t \wedge t \leq b$, and (2) holds too.

Then, for $T = T'$, $A_0' \wedge B_0' \wedge (\mathcal{H}_1 \backslash \mathcal{H}_{1\mathsf{mix}}) \wedge \mathcal{H}_{1\mathsf{sep}} \wedge \neg(a \leq t \wedge t \leq b)$ is logically equivalent to $A_0 \wedge B_0 \wedge (\mathcal{H} \backslash \mathcal{H}_{\mathsf{mix}}) \wedge \mathcal{H}_{\mathsf{sep}} \wedge \neg(a \leq t \wedge t \leq b)$, so $A_0 \wedge B_0 \wedge (\mathcal{H} \backslash \mathcal{H}_{\mathsf{mix}}) \wedge \mathcal{H}_{\mathsf{sep}} \models (a \leq t \wedge t \leq b)$, i.e. (1) holds.

In order to prove (2), note that, by definition, $\mathcal{H}_{1\mathsf{mix}} = \mathcal{H}_{\mathsf{mix}}$ and $\mathcal{H}_{1\mathsf{sep}} = \mathcal{H}_{\mathsf{sep}}$. On the one hand, by the induction hypothesis, $A_0' \wedge B_0' \wedge (\mathcal{H}_1 \backslash \mathcal{H}_{1\mathsf{mix}}) \cup \mathcal{H}_{1\mathsf{sep}} \wedge \neg(a \leq t \wedge t \leq b)$ is logically equivalent to a corresponding conjunction $\overline{A}_0' \wedge \overline{B}_0' \wedge \neg(a \leq t \wedge t \leq b)$ containing as conjuncts all literals in A_0' and B_0' and all conclusions of rules in $\mathcal{H}_1 \backslash \mathcal{H}_{1\mathsf{mix}}$ and $\mathcal{H}_{1\mathsf{sep}}$. On the other hand, $A_0' \wedge B_0' \wedge \neg(a \leq t \wedge t \leq b)$ is logically equivalent to $A_0 \wedge B_0 \wedge (c \leq d) \wedge \neg(a \leq t \wedge t \leq b)$, where $(c \leq d)$ is the conclusion of the rule $C \in \mathcal{H} \backslash \mathcal{H}_{\mathsf{mix}}$. This proves (2).

Case 3b: Assume now that $C = c_1 \leq d_1 \rightarrow c \leq d$ is mixed, for instance that c_1, c are constants in A and d_1, d are constants in B.

(a) Assume C is obtained from an instance of a clause of the form $x \leq g(y) \rightarrow f(x) \leq h(y)$. This means that there exist $c \approx f(c_1) \in \mathsf{Def}_A$ and $d_1 \approx g(e), d \approx h(e) \in \mathsf{Def}_B$. We know that $A_0 \wedge B_0 \models_{\mathcal{T}_0} c_1 \leq d_1$ and that $\mathcal{T}_0$ is $\leq$-interpolating. Thus, there exists a term t_1 containing only constants common to A_0 and B_0 such that

$$A_0 \wedge B_0 \models_{\mathcal{T}_0} c_1 \leq t_1 \wedge t_1 \leq d_1.$$

Let $c_{f(t_1)}$ be a new constant, denoting the term $f(t_1)$, and let

$$C_A = c_1 \leq t_1 \rightarrow c \leq c_{f(t_1)} \quad \text{and} \quad C_B = t_1 \leq d_1 \rightarrow c_{f(t_1)} \leq d.$$

Thus, C_A corresponds to the instance $c_1 \leq t_1 \rightarrow f(c_1) \leq f(t_1)$ of the monotonicity axiom for f, whereas C_B corresponds to the rule $t_1 \leq g(e) \rightarrow f(t_1) \leq h(e)$.

As $A_0 \wedge B_0 \models c_1 \leq t_1 \wedge t_1 \leq d_1$ and as $\leq$ is transitive, the following holds:

$$\begin{aligned} A_0 \wedge B_0 \wedge C_A \wedge C_B &\equiv_{\mathcal{T}_0} A_0 \wedge B_0 \wedge (c_1 \leq t_1 \wedge C_A) \wedge (t_1 \leq d_1 \wedge C_B) \\ &\models_{\mathcal{T}_0} A_0 \wedge B_0 \wedge c \leq c_{f(t_1)} \wedge c_{f(t_1)} \leq d \\ &\models_{\mathcal{T}_0} A_0 \wedge B_0 \wedge c \leq d, \end{aligned}$$

Thus, $A_0 \wedge B_0 \wedge C_A \wedge C_B \wedge (\mathcal{H} \backslash \{C\}) \models_{\mathcal{T}_0} A_0 \wedge B_0 \wedge c \leq d \wedge (\mathcal{H} \backslash \{C\})$. Since $A_0 \wedge B_0 \models_{\mathcal{T}_0} c_1 \leq d_1$ it follows that $A_0 \wedge B_0 \wedge \mathcal{H}$ is logically equivalent with $A_0 \wedge B_0 \wedge c \leq d \wedge (\mathcal{H} \backslash \{C\})$, so we have $A_0 \wedge B_0 \wedge C_A \wedge C_B \wedge (\mathcal{H} \backslash \{C\}) \wedge \neg(a \leq b) \models_{\mathcal{T}_0} \bot$.

By the induction hypothesis for $A_0 \wedge B_0 \wedge c \leq c_{f(t_1)} \wedge c_{f(t_1)} \leq d$ and $\mathcal{H}_1 = \mathcal{H} \backslash \{C\}$ we know that there exists a set T' of terms such that

$$A_0 \wedge B_0 \wedge c \leq c_{f(t_1)} \wedge c_{f(t_1)} \leq d \wedge (\mathcal{H}_1 \backslash \mathcal{H}_{1\mathsf{mix}}) \wedge \mathcal{H}_{1\mathsf{sep}} \wedge \neg(a \leq t \wedge t \leq b) \models \bot,$$

and also (2) holds. Then (1) holds for $T = T' \cup \{f(t_1), t_1\}$.

(b) Assume C corresponds to an instance of a monotonicity axiom $x \leq y \rightarrow f(x) \leq f(y)$. This means that there exist $c \approx f(c_1) \in \mathsf{Def}_A$ and $d \approx f(d_1) \in \mathsf{Def}_B$. We know that $A_0 \wedge B_0 \models_{\mathcal{T}_0} c_1 \leq d_1$ and that $\mathcal{T}_0$ is $\leq$-interpolating. Thus, there exists a term t_1 containing only constants common to A_0 and B_0 such that

$$A_0 \wedge B_0 \models_{\mathcal{T}_0} c_1 \leq t_1 \wedge t_1 \leq d_1.$$

Let $c_{f(t_1)}$ be a new constant, denoting the term $f(t_1)$, and let

$$C_A = c_1 \leq t_1 \rightarrow c \leq c_{f(t_1)} \quad \text{and} \quad C_B = t_1 \leq d_1 \rightarrow c_{f(t_1)} \leq d.$$

Thus, C_A corresponds to the instance $c_1 \leq t_1 \rightarrow f(c_1) \leq f(t_1)$ of the monotonicity axiom for f, whereas C_B corresponds to the instance $t_1 \leq d_1 \rightarrow f(t_1) \leq f(d_1)$ of the monotonicity axiom for f. The proof can then continue as the proof of case (a); also in this case we can choose $T = T' \cup \{f(t_1), t_1\}$.

(2) can be proved similarly using the induction hypothesis. □

We show two examples for the separation of mixed instances, one containing a mixed instance of a role inclusion axiom (Example 5.14) and one containing a mixed instance of a monotonicity axiom (Example 5.15).

Example 5.14. Let A_0, B_0 and $\mathcal{H}$ be the following sets of flattened and purified formulae (where the clause in $\mathcal{H}$ corresponds to an instance of a role inclusion axiom of the form $x \leq g(y) \rightarrow f(x) \leq h(y)$):

$$\begin{aligned} A_0 &= \{\ a_1 \leq c_{f(a_2)},\ a_2 \leq d\ \} \\ B_0 &= \{\ d \leq c_{g(b_1)},\ c_{h(b_1)} \leq b_2\ \} \\ \mathcal{H} &= \{\ a_2 \leq c_{g(b_1)} \rightarrow c_{f(a_2)} \leq c_{h(b_1)}\ \} = \mathcal{H}_{\mathsf{mix}} \end{aligned}$$

It holds that $A_0 \wedge B_0 \wedge \mathcal{H} \models a_1 \leq b_2$. The clause in $\mathcal{H}$ is a mixed instance, since a_2 appears only in A_0 and b_1 only in B_0, so $\mathcal{H}_{\mathsf{mix}} = \mathcal{H}$. To separate the mixed instance we compute

an intermediate term t for its premise $a_2 \leq c_{g(b_1)}$ using Algorithm 4. As a result we get $t = d$. From this we get the following set of separated instances:

$$\begin{aligned}\mathcal{H}_{\mathsf{sep}} = \{\ a_2 \leq d &\rightarrow\ c_{f(a_2)} \leq c_{f(d)},\\ d \leq c_{g(b_1)} &\rightarrow\ c_{f(d)} \leq c_{h(b_1)}\ \}\end{aligned}$$

Note that the first one is an instance of the monotonicity axiom for f and contains only symbols appearing in A_0, whereas the second one is an instance of the role inclusion axiom and contains only symbols appearing in B_0. ■

Example 5.15. Let A_0, B_0 and $\mathcal{H}$ be the following sets of flattened and purified formulae (where both of the clauses in $\mathcal{H}$ correspond to an instance of a monotonicity axiom):

$$\begin{aligned}A_0 &= \{\ a_1 \leq c_{f(a_2)},\ a_2 \leq c_{f(d_1)},\ a_2 \leq d_2\ \}\\ B_0 &= \{\ d_1 \leq b_1,\ (c_{f(b_1)} \wedge d_2) \leq b_2\ \}\\ \mathcal{H} &= \{\ a_2 \leq b_2 \rightarrow c_{f(a_2)} \leq c_{f(b_2)},\ d_1 \leq b_1 \rightarrow c_{f(d_1)} \leq c_{f(b_1)}\ \}\end{aligned}$$

It holds that $A_0 \wedge B_0 \wedge \mathcal{H} \models a_1 \leq c_{f(b_2)}$. The first clause in $\mathcal{H}$ is a mixed instance, since a_2 appears only in A_0 and b_2 only in B_0. The second clause in $\mathcal{H}$ is not mixed, since both d_1 and b_1 appear in B_0. Thus, $\mathcal{H}_{\mathsf{mix}} = \{a_2 \leq b_2 \rightarrow c_{f(a_2)} \leq c_{f(b_2)}\}$. We use Algorithm 4 to compute an intermediate term t for the premise $a_2 \leq b_2$ and obtain $t = (c_{f(d_1)} \wedge d_2)$. We then have the following set of separated instances:

$$\begin{aligned}\mathcal{H}_{\mathsf{sep}} = \{\ a_2 \leq (c_{f(d_1)} \wedge d_2) &\rightarrow\ c_{f(a_2)} \leq c_{f(c_{f(d_1)} \wedge d_2)},\\ (c_{f(d_1)} \wedge d_2) \leq b_2 &\rightarrow\ c_{f(c_{f(d_1)} \wedge d_2)} \leq c_{f(b_2)}\ \}\end{aligned}$$

Both clauses correspond to instances of the monotonicity axiom of f. The first one contains only symbols from A_0 and the second one only symbols from B_0. ■

Theorem 5.16. *The theory $\mathsf{SLO}(\Sigma, RI_a)$ of semilattices with monotone operators satisfying axioms RI_a is $\leq$-interpolating.*

Proof: The operators of $\mathsf{SLO}(\Sigma, RI_a)$ satisfy the monotonicity condition Mon; the axioms in RI_a are in a class that was studied in [93]. Let A and B be two conjunctions of literals (corresponding to two TBoxes), let RI be a set of role axioms and let Mon be the family of all monotonicity axioms for the functions $\{f_{\exists r} \mid r \in N_R\}$. Assume that $A \wedge B \models_{\mathsf{SLO}(\Sigma, RI_a)} a \leq b$, where a is a term containing only constants and Σ-functions occurring in A and b is a term containing only constants and Σ-functions occurring in B. By Theorem 5.6, $A \wedge B \models_{\mathsf{SLO}(\Sigma, RI_a)} a \leq b$ if and only if (with the notation used in Theorem 5.4) $A_0 \wedge B_0 \wedge \mathsf{Mon}[A \wedge B]_0 \wedge RI_a[A \wedge B]_0 \wedge \mathsf{Con}_0 \wedge \neg(a \leq b)_0 \models_{\mathsf{SLat}} \bot$. In the presence of monotonicity, Con is not needed. The set $\mathcal{H} = \mathsf{Mon}[A \wedge B]_0 \wedge RI_a[A \wedge B]_0 \wedge \neg(a \leq b)_0$ contains mixed clauses. Using the result from Theorem 5.13 we can "separate" all clauses in $\mathcal{H}$ and then compute an intermediate term for $a \leq b$ using Algorithm 4, i.e. we have $A_0 \wedge \mathcal{H}^A_{\mathsf{sep}} \wedge B_0 \wedge \mathcal{H}^B_{\mathsf{sep}} \wedge \neg(a_0 \leq t_0 \wedge t_0 \leq b_0) \models_{\mathsf{SLat}} \bot$, where t_0 contains only constants common to A_0 and B_0. After replacing back the new constants with the terms they represent, we obtain $A \wedge B \models_{\mathsf{SLO}(\Sigma, RI_a)} (a \leq t \wedge t \leq b)$, where t contains only symbols which are common to A and B.[2] □

[2] As in [93], for function symbols f, g, if f occurs in A and g occurs in B, but they occur together in one of the axioms in RI, they are considered to be shared.

5.3 $\leq$-Interpolation for High-Level Explanations

In this section we explain in detail our method for finding high-level explanations for subsumptions in combinations or extensions of $\mathcal{EL}^+$ ontologies. The method we propose combines hierarchical reasoning, $\leq$-interpolation and resolution and can be enhanced by unsatisfiable core computation. We first present the algorithm and then apply it on the example from Section 1.1. We then briefly present an implementation of the algorithm and test it on two more examples.

First we formally state the problem we are addressing. Let the following be given:

- Let $\mathcal{T}_A$ and $\mathcal{T}_B$ be two $\mathcal{EL}^+$ TBoxes and RI be a set of role inclusions.
- Let N_C be the set of all concept names occurring in $\mathcal{T}_A \cup \mathcal{T}_B$.
- Let N_C^A be the set of concept names occurring in $\mathcal{T}_A$.
- Let N_C^B be the set of concept names occurring in $\mathcal{T}_B$.
- Let $N_C^{AB} = (N_C^A \cap N_C^B)$ be the common concept names.
- Let X be a concept description over N_C^A and Y a concept description over N_C^B such that they do not contain only shared symbols and such that the following hold:

$$\begin{aligned} \mathcal{T}_A \cup \mathcal{T}_B \cup RI &\models X \sqsubseteq Y \\ \mathcal{T}_A \cup RI &\not\models X \sqsubseteq Y \\ \mathcal{T}_B \cup RI &\not\models X \sqsubseteq Y \end{aligned}$$

The goal is to find a concept description C containing only concepts in N_C^{AB} (and possibly also only roles common to $\mathcal{T}_A$ and $\mathcal{T}_B$) such that

$$\begin{aligned} &\mathcal{T}_A \cup \mathcal{T}_B \cup RI \models X \sqsubseteq C \qquad \text{and} \\ &\mathcal{T}_A \cup \mathcal{T}_B \cup RI \models \quad C \sqsubseteq Y. \end{aligned}$$

The concept description C can then be seen as a "high-level explanation" for $X \sqsubseteq Y$. Using Theorem 5.3 and Theorem 5.16 we can always compute such a concept description. For this we propose Algorithm 5.

Algorithm 5 Algorithm for finding explanations in $\mathcal{EL}^+$

Input:	$\mathcal{EL}^+$ TBoxes $\mathcal{T}_A$ and $\mathcal{T}_B$; set of role inclusions RI; concept descriptions X over N_C^A and Y over N_C^B such that $\mathcal{T}_A \cup \mathcal{T}_B \cup RI \models X \sqsubseteq Y$
Output:	Concept description C over N_C^{AB} such that $\mathcal{T}_A \cup \mathcal{T}_B \cup RI \models X \sqsubseteq C$ and $\mathcal{T}_A \cup \mathcal{T}_B \cup RI \models C \sqsubseteq Y$

Step 1: 1. Translate $\mathcal{T}_A \cup \mathcal{T}_B \cup RI$ to the theory of semilattices with monotone operators.
2. Compute a minimal set of axioms min such that $min \cup RI \models_{\mathsf{SLO}(\Sigma, RI_a)} X \leq Y$.

Step 2: 1. Flatten, purify and instantiate (using Ψ_{RI}) $min \cup RI$.
2. Compute a minimal set of axioms min' such that $min' \models_{\mathsf{SLO}(\Sigma, RI_a)} X \leq Y$.

Step 3: Separate all mixed instances of role and monotonicity axioms.

Step 4: 1. Compute an intermediate term C for $X \leq Y$ using Algorithm 4.
2. Translate C to the language of $\mathcal{EL}^+$.

The purpose of Steps 1.2 and 2.2 in Algorithm 5 is to make the algorithm more efficient, especially for large ontologies. Usually, if we have very large TBoxes, only some of their axioms are necessary for obtaining a certain consequence. Therefore it is sufficient to apply Step 2 only on the relevant axioms. Similarly, it is sufficient to apply Step 3 only on the instances relevant to the problem. For determining which axioms/instances are relevant we can compute a minimal axiom set, for example by using unsatisfiable core computation, which is done in Steps 1.2 and 2.2 of the algorithm. Note that these two steps are completely optional, they can also be left out or, alternatively, only a reduced set of axioms, which is not necessarily minimal, could be computed.

Example 5.17. Consider the $\mathcal{EL}$ ontology $\mathcal{O}_{Amp}$ in Figure 5.2, which was already presented in Section 1.1 and is based on an example from [12].

A_1 :	AmpOfFinger	$\sqsubseteq$	Amputation
A_2 :	AmpOfFinger	$\sqsubseteq$	$\exists$site.FingerStructure
A_3 :	Amputation $\sqcap \exists$site.FingerStructure	$\sqsubseteq$	AmpOfFinger
A_4 :	InjToFinger	$\sqsubseteq$	Injury
A_5 :	InjToFinger	$\sqsubseteq$	$\exists$site.FingerStructure
A_6 :	Injury $\sqcap \exists$site.FingerStructure	$\sqsubseteq$	InjToFinger
A_7 :	FingerEntity	$\sqsubseteq$	FingerStructure
A_8 :	FingerPart	$\sqsubseteq$	FingerStructure
A_9 :	FingerPart	$\sqsubseteq$	$\exists$part.FingerEntity
A_{10} :	FingerStructure	$\sqsubseteq$	HandPart
A_{11} :	HandEntity	$\sqsubseteq$	HandStructure
A_{12} :	HandPart	$\sqsubseteq$	HandStructure
A_{13} :	HandPart	$\sqsubseteq$	$\exists$part.HandEntity
B_1 :	AmpOfHand	$\sqsubseteq$	Amputation
B_2 :	AmpOfHand	$\sqsubseteq$	$\exists$site.HandStructure
B_3 :	Amputation $\sqcap \exists$site.HandStructure	$\sqsubseteq$	AmpOfHand
B_4 :	InjToHand	$\sqsubseteq$	Injury
B_5 :	InjToHand	$\sqsubseteq$	$\exists$site.HandStructure
B_6 :	Injury $\sqcap \exists$site.HandStructure	$\sqsubseteq$	InjToHand

Figure 5.2: Ontology $\mathcal{O}_{Amp}$.

The ontology consists of two $\mathcal{EL}$ TBoxes $\mathcal{T}_A$ and $\mathcal{T}_B$. As we are in $\mathcal{EL}$, we do not have any role inclusions. TBox $\mathcal{T}_A$ describes concepts for amputation of a finger and injury to a finger. It also defines relations between finger and hand using concepts for structure, entity and part of a finger or hand. TBox $\mathcal{T}_B$ extends $\mathcal{T}_A$ by a description of concepts for amputation and injury of a hand. As a consequence of this extension the following two subsumptions are true:

$$\mathsf{InjToFinger} \sqsubseteq \mathsf{InjToHand} \quad \text{and} \quad \mathsf{AmpOfFinger} \sqsubseteq \mathsf{AmpOfHand}$$

While the first subsumption makes sense, the second one is not a subsumption that is supposed to hold, as an amputation of a finger is not an amputation of the whole hand.

We are interested in repairing the ontology such that $\mathsf{AmpOfFinger} \sqsubseteq \mathsf{AmpOfHand}$ does not hold anymore. For this we need to understand what the cause of this faulty consequence could be. Note that the following requirements hold:

$$\mathcal{T}_A \cup \mathcal{T}_B \cup RI \vDash \mathsf{AmpOfFinger} \sqsubseteq \mathsf{AmpOfHand}$$
$$\mathcal{T}_A \cup RI \nvDash \mathsf{AmpOfFinger} \sqsubseteq \mathsf{AmpOfHand}$$
$$\mathcal{T}_B \cup RI \nvDash \mathsf{AmpOfFinger} \sqsubseteq \mathsf{AmpOfHand}$$

Therefore, we can use Algorithm 5 to find an explanation over the common signature of the two TBoxes, i.e. a concept C such that the following hold:

$$\mathcal{T}_A \cup \mathcal{T}_B \cup RI \models \mathsf{AmpOfFinger} \sqsubseteq C$$
$$\mathcal{T}_A \cup \mathcal{T}_B \cup RI \models C \sqsubseteq \mathsf{AmpOfHand}$$

In the following we show how to apply the four steps of the algorithm. We have the following sets of symbols (we indicate also the abbreviations used in what follows):

$$\begin{aligned} N_C^A &= \{\mathsf{AmpOfFinger\ (AF)}, \mathsf{Amputation\ (A)}, \mathsf{FingerStructure\ (FS)}, \mathsf{InjToFinger\ (IF)}, \\ &\quad \mathsf{Injury\ (I)}, \mathsf{FingerEntity\ (FE)}, \mathsf{FingerPart\ (FP)}, \mathsf{HandPart\ (HP)}, \\ &\quad \mathsf{HandEntity\ (HE)}, \mathsf{HandStructure\ (HS)}\} \\ N_C^B &= \{\mathsf{AmpOfHand\ (AH)}, \mathsf{Amputation\ (A)}, \mathsf{HandStructure\ (HS)}, \mathsf{InjToHand\ (IH)}, \\ &\quad \mathsf{Injury\ (I)}\} \\ N_C^{AB} &= \{\mathsf{Amputation\ (A)}, \mathsf{Injury\ (I)}, \mathsf{HandStructure\ (HS)}\} \end{aligned}$$

Step 1.1: We translate the ontology to the theory of semilattices with monotone operators. For this we replace $\sqsubseteq$ by $\leq$ and $\sqcap$ by $\wedge$. We now state the monotonicity axioms for each role, i.e. site and part, explicitly. Figure 5.3 shows the ontology after the translation to the theory of semilattices. Note that from here on we use the abbreviations for concept names indicated in the sets N_C^A, N_C^B and N_C^{AB} above.

A_1 :	AF	$\leq$	A	B_1 :	AH	$\leq$	A
A_2 :	AF	$\leq$	site(FS)	B_2 :	AH	$\leq$	site(HS)
A_3 :	A $\wedge$ site(FS)	$\leq$	AF	B_3 :	A $\wedge$ site(HS)	$\leq$	AH
A_4 :	IF	$\leq$	I	B_4 :	IH	$\leq$	I
A_5 :	IF	$\leq$	site(FS)	B_5 :	IH	$\leq$	site(HS)
A_6 :	I $\wedge$ site(FS)	$\leq$	IF	B_6 :	I $\wedge$ site(HS)	$\leq$	IH
A_7 :	FE	$\leq$	FS				
A_8 :	FP	$\leq$	FS				
A_9 :	FP	$\leq$	part(FE)				
A_{10} :	FS	$\leq$	HP	M_1 :	$\forall$X,Y: X $\leq$ Y	$\rightarrow$	site(X) $\leq$ site(Y)
A_{11} :	HE	$\leq$	HS	M_2 :	$\forall$X,Y: X $\leq$ Y	$\rightarrow$	part(X) $\leq$ part(Y)
A_{12} :	HP	$\leq$	HS				
A_{13} :	HP	$\leq$	part(HE)				

Figure 5.3: $\mathcal{O}_{Amp}$ after Step 1.1.

Step 1.2: Based on the axioms from Figure 5.3 we compute a minimal unsatisfiable core of the set $\{A_1, ..., A_{13}\} \cup \{B_1, ..., B_6\} \cup \{M_1, M_2\} \cup \{\neg(\mathsf{AF} \leq \mathsf{AH})\}$ and obtain the minimal axiom set

$$min = \{A_1, A_2, A_{10}, A_{12}, B_3, M_1\}$$

from which $\mathsf{AF} \leq \mathsf{AH}$ can be derived. This means that for the following instantiation step we only have to consider monotonicity axiom M_1, but not M_2. Figure 5.4 shows the reduced ontology.

A_1 :	AF	$\leq$	A	B_3 :	A $\wedge$ site(HS)	$\leq$	AH
A_2 :	AF	$\leq$	site(FS)				
A_{10} :	FS	$\leq$	HP	M_1 :	$\forall$X,Y: X $\leq$ Y	$\rightarrow$	site(X) $\leq$ site(Y)
A_{12} :	HP	$\leq$	HS				

Figure 5.4: $\mathcal{O}_{Amp}$ after Step 1.2.

Step 2.1: Let $\mathcal{T}_0 = \mathsf{SLat}$ and $T_1 = \mathsf{SLat} \cup M_1$ be the extension of T_0 with the monotonicity axiom M_1. We know from Theorem 5.4 that this is a local theory extension, so we can use hierarchical reasoning. We have the following set of ground terms:

$$T = \{\ \mathsf{site(FS)},\ \mathsf{site(HS)}\ \}$$

From T we only get two instances I_1 and I_2 of the monotonicity axiom M_1:

$$I_1 : \quad \mathsf{FS} \leq \mathsf{HS} \quad \rightarrow \quad \mathsf{site(FS)} \leq \mathsf{site(HS)}$$
$$I_2 : \quad \mathsf{HS} \leq \mathsf{FS} \quad \rightarrow \quad \mathsf{site(HS)} \leq \mathsf{site(FS)}$$

We purify all formulae by introducing new constants for the terms starting with a function symbol, i.e. role names. We save the definitions in the following set:

$$\mathsf{Def} = \{\mathsf{site_{FS}} = \mathsf{site(FS)}, \mathsf{site_{HS}} = \mathsf{site(HS)}\}$$

We then have the set $A_0 \cup B_0 \cup I_0$, where A_0, B_0 and I_0 are the purified versions of $A = \{A_1, A_2, A_{10}, A_{12}\}$, $B = \{B_3\}$ and $I = \{I_1, I_2\}$, respectively. We obtain the set of formulae shown in Figure 5.5, containing these purified axioms.

A_1 :	AF	$\leq$	A	B_3 :	A $\wedge$ $\mathsf{site_{HS}}$	$\leq$	AH
A_2 :	AF	$\leq$	$\mathsf{site_{FS}}$				
A_{10} :	FS	$\leq$	HP	I_1 :	FS $\leq$ HS	$\rightarrow$	$\mathsf{site_{FS}} \leq \mathsf{site_{HS}}$
A_{12} :	HP	$\leq$	HS	I_2 :	HS $\leq$ FS	$\rightarrow$	$\mathsf{site_{HS}} \leq \mathsf{site_{FS}}$

Figure 5.5: $\mathcal{O}_{Amp}$ after Step 2.1.

Step 2.2: To reduce the number of instances we compute a minimal unsatisfiable core of $A_0 \cup B_0 \cup I_0 \cup \{\neg(\mathsf{AF} \leq \mathsf{AH})\}$ and obtain the set of axioms

$$min' = \{A_1, A_2, A_{10}, A_{12}, B_3, I_1\}$$

from which $\mathsf{AF} \leq \mathsf{AH}$ can be derived. We therefore know that from the two instances of M_1 only I_1 is needed. In Figure 5.6 the new set of formulae is shown.

A_1 :	AF	$\leq$	A	B_3 :	$\mathsf{A} \wedge \mathsf{site_{HS}}$	$\leq$	AH
A_2 :	AF	$\leq$	$\mathsf{site_{FS}}$				
A_{10} :	FS	$\leq$	HP	I_1 :	$\mathsf{FS} \leq \mathsf{HS}$	$\rightarrow$	$\mathsf{site_{FS}} \leq \mathsf{site_{HS}}$
A_{12} :	HP	$\leq$	HS				

Figure 5.6: $\mathcal{O}_{Amp}$ after Step 2.2.

Step 3: The premise of I_1 contains only symbols occuring in N_C^A, so it is not a mixed instance. Thus, there is nothing to do in this step and the set of axioms remains the same.

Step 4.1: We know that I_1 is needed to derive the unwanted subsumption, so its premise must be true (in fact, from A_{10} and A_{12} it follows that $\mathsf{FS} \leq \mathsf{HP} \leq \mathsf{HS}$). Note that w.r.t. SLat the formula $A_0 \wedge I_1$ is equivalent to the following formula:

$$\overline{A}_0 = (\mathsf{AF} \leq \mathsf{A}) \wedge (\mathsf{AF} \leq \mathsf{site_{FS}}) \wedge (\mathsf{FS} \leq \mathsf{HP}) \wedge (\mathsf{HP} \leq \mathsf{HS}) \wedge (\mathsf{site_{FS}} \leq \mathsf{site_{HS}})$$

As a whole, we have the formula $\overline{A}_0 \wedge B_0$ with the following sets of symbols:

- $S_A^* = \{\mathsf{AF}, \mathsf{site_{FS}}, \mathsf{FS}, \mathsf{HP}, \mathsf{HS}\}$, the set of symbols occurring only in $\overline{A}_0$
- $S_B^* = \{\mathsf{AH}\}$, the set of symbols occurring only in B_0
- $S_{AB} = \{\mathsf{A}, \mathsf{site_{HS}}\}$, the set of symbols occurring in $\overline{A}_0$ and B_0

Using on $\overline{A}_0 \wedge B_0$ the translation to propositional logic described in the proof of Theorem 5.11 and using the same names for the SLat-terms and the corresponding propositional variables we obtain a set of Horn clauses N_{AB}:

$$N_{AB} = \{\ (\neg\mathsf{AF} \vee \mathsf{A}),\ (\neg\mathsf{AF} \vee \mathsf{site_{FS}}),\ (\neg\mathsf{FS} \vee \mathsf{HP}),\ (\neg\mathsf{HP} \vee \mathsf{HS}),\ (\neg\mathsf{site_{FS}} \vee \mathsf{site_{HS}}),\ (\neg\mathsf{A} \vee \neg\mathsf{site_{HS}} \vee \mathsf{AH})\ \}$$

To obtain an explanation for $\mathcal{T}_A \cup \mathcal{T}_B \cup RI \models \mathsf{AmpOfFinger} \sqsubseteq \mathsf{AmpOfHand}$ we saturate the set $N_{AB} \cup \{\mathsf{AF}\}$ under positive unit resolution as described in the proof of Theorem 5.11, restricting to inferences in which one clause is a positive unit clause:

- Resolution of AF and $\neg\mathsf{AF} \vee \mathsf{A}$ yields A.
- Resolution of AF and $\neg\mathsf{AF} \vee \mathsf{site_{FS}}$ yields $\mathsf{site_{FS}}$.
- Resolution of $\mathsf{site_{FS}}$ and $\neg\mathsf{site_{FS}} \vee \mathsf{site_{HS}}$ yields $\mathsf{site_{HS}}$.
- Resolution of A and $\neg\mathsf{A} \vee \neg\mathsf{site_{HS}} \vee \mathsf{AH}$ yields $\neg\mathsf{site_{HS}} \vee \mathsf{AH}$.
- Resolution of $\mathsf{site_{HS}}$ and $\neg\mathsf{A} \vee \neg\mathsf{site_{HS}} \vee \mathsf{AH}$ yields $\neg\mathsf{A} \vee \mathsf{AH}$.
- Resolution of A and $\neg\mathsf{A} \vee \mathsf{AH}$ yields AH.

Two of the obtained resolvents are unit clauses containing only common symbols: A and $\mathsf{site_{HS}}$. Thus, we obtain the intermediate term $\mathsf{A} \wedge \mathsf{site_{HS}}$ for $\mathsf{AF} \leq \mathsf{AH}$.

Step 4.2: Translating $\mathsf{A} \wedge \mathsf{site_{HS}}$ back to description logic yields the following formula:

$$J = \mathsf{Amputation} \sqcap \exists \mathsf{site}.\mathsf{HandStructure}$$

Indeed, the following properties hold:

$$\mathcal{T}_A \cup \mathcal{T}_B \cup RI \models \mathsf{AmpOfFinger} \sqsubseteq J$$
$$\mathcal{T}_A \cup \mathcal{T}_B \cup RI \models J \sqsubseteq \mathsf{AmpOfHand}$$

So J is the intermediate term that is computed by Algorithm 5.

With J we now have a high-level explantion for the unwanted subsumption wich can give us a hint on how to repair the ontology such that the unwanted consequence is not entailed by it anymore. Since we consider $\mathcal{T}_A$ to be consistent, we are looking for axioms in $\mathcal{T}_B$ that could cause this problem. The explanation contains the concepts Amputation and HandStructure, so we could try to change some of the axioms in $\mathcal{T}_B$ containing one of these concepts in our original ontology. Actually, by the unsat core computation in Step 2 of the algorithm we have already pinpointed B_3 as a cause of error. Indeed, replacing $\exists$site.HandStructure by $\exists$site.HandEntity in Axioms B_2 and B_3 will fix the ontology. ■

5.4 Implementation and Tests

An implementation of Algorithm 5 was done by Sebastian Thunert for his Master thesis "Automatization of Computing High-Level Explanations for Subsumptions in $\mathcal{EL}$ and $\mathcal{EL}^+$" [108]. The program is written in Python and uses Z3 and SPASS as external provers. The steps of the program follow those of the algorithm, including the optional optimization steps for computating reduced sets of axioms and instances.

In what follows we give a short description of the input and output as well as the four steps of the implemented program.

Input: The input must be given by the user in the form of a Python Script file and contains the axioms of the TBoxes $\mathcal{T}_A$ and $\mathcal{T}_B$, the role inclusions RI, and the subsumption for which an explanation is demanded.

Step 1: In this step a translation to either SPASS or Z3 is performed. It also aims at reducing the number of axioms by computing a subset of the axioms which entails the subsumption. For this the user can choose between a precise translation to SPASS or a translation to Z3 which is not always precise and may therefore give a wrong reduced set of axioms (i.e. one that does not entail the subsumption). While with using Z3 always a minimal set of axioms which entails the subsumption is computed, SPASS will only compute a subset of the axioms that is not necessarily minimal.

Step 2: In this step the flattening, instantiation and purification of the axioms is done. For the reduced problem a precise translation to Z3 is possible. Z3 is used to reduce the number of axioms and instances to a minimal set that is necessary for deriving the subsumption.

Step 3: The program separates the mixed instances by computing intermediate terms for their premises using Algorithm 4. For applying resolution SPASS is used.

Step 4: As a last step an intermediate term for the subsumption is computed using Algorithm 4. From the result of the resolution, again applied using SPASS, the program generates the intermediate term in the language of $\mathcal{EL}^+$.

Output: A text file containing the generated explanation for the subsumption is created.

In [108] the implementation has been tested on Example 5.17 from the previous chapter, as well as two examples which follow (Example 5.18 and Example 5.19). In contrast to the ontology in Example 5.17, which did not contain role inclusions, the ontologies in the examples that follow are in $\mathcal{EL}^+$.

Example 5.18. Consider the ontology $\mathcal{O}_{Oxi}$ in Figure 5.7, which is based on an example in [93] with some slight changes.

A_1 :	CatOxidation	$\sqsubseteq$	$\exists$catalyzes.Oxidation
A_2 :	CatOxidation	$\sqsubseteq$	Substance
A_3 :	Oxidation	$\sqsubseteq$	Reaction
B_1 :	Reaction	$\sqsubseteq$	Process
B_2 :	Reaction	$\sqsubseteq$	$\exists$produces.Substance
B_3 :	Enzyme	$\sqsubseteq$	Substance
B_4 :	Enzyme	$\sqsubseteq$	$\exists$enhances.Process
B_5 :	Substance $\sqcap$ $\exists$enhances.Process	$\sqsubseteq$	Enzyme
R_1 :	catalyzes	$\sqsubseteq$	enhances

Figure 5.7: Ontology $\mathcal{O}_{Oxi}$.

We have a TBox $\mathcal{T}_A$ specifying the concepts CatOxidation and Oxidation, and a TBox $\mathcal{T}_B$ describing the concepts Reaction and Enzyme. The ontology also contains a role inclusion R_1, so we are in $\mathcal{EL}^+$ now.

We have the following sets of symbols (we also indicate the abbreviations used in what follows):

$$\begin{aligned} N_C^A &= \{\text{CatOxidation (CO), Oxidation (O), Substance (S), Reaction (R)}\} \\ N_C^B &= \{\text{Reaction (R), Process (P), Substance (S), Enzyme (E)}\} \\ N_C^{AB} &= \{\text{Substance (S), Reaction (R)}\} \end{aligned}$$

In the ontology $\mathcal{O}_{Oxi}$ it holds that CatOxidation $\sqsubseteq$ Enzyme, where CatOxidation $\in N_C^A$ and Enzyme $\in N_C^B$. More precisely, the following holds:

$$\begin{aligned} \mathcal{T}_A \cup \mathcal{T}_B \cup RI &\models \text{CatOxidation} \sqsubseteq \text{Enzyme} \\ \mathcal{T}_A \cup RI &\not\models \text{CatOxidation} \sqsubseteq \text{Enzyme} \\ \mathcal{T}_B \cup RI &\not\models \text{CatOxidation} \sqsubseteq \text{Enzyme} \end{aligned}$$

We use Algorithm 5 to compute an intermediate term containing only shared symbols for the subsumption CatOxidation $\sqsubseteq$ Enzyme.

Step 1.1: We translate the ontology to the theory of semilattices with monotone operators and state monotonicity axioms for each role explicitly. We use the abbreviations for concept names indicated in the sets N_C^A, N_C^B and N_C^{AB}. We also use abbreviations for role names, i.e. c for catalyzes, p for produces and e for enhances. Figure 5.8 shows the ontology after the translation.

A_1 :		CO	$\leq$	$\mathsf{c(O)}$	B_1 :	R	$\leq$	P
A_2 :		CO	$\leq$	S	B_2 :	R	$\leq$	$\mathsf{p(S)}$
A_3 :		O	$\leq$	R	B_3 :	E	$\leq$	S
					B_4 :	E	$\leq$	$\mathsf{e(P)}$
M_1 :	$\forall \mathsf{X,Y}$:	$\mathsf{X} \leq \mathsf{Y}$	$\rightarrow$	$\mathsf{c(X)} \leq \mathsf{c(Y)}$	B_5 :	$\mathsf{S} \wedge \mathsf{e(P)}$	$\leq$	E
M_2 :	$\forall \mathsf{X,Y}$:	$\mathsf{X} \leq \mathsf{Y}$	$\rightarrow$	$\mathsf{p(X)} \leq \mathsf{p(Y)}$				
M_3 :	$\forall \mathsf{X,Y}$:	$\mathsf{X} \leq \mathsf{Y}$	$\rightarrow$	$\mathsf{e(X)} \leq \mathsf{e(Y)}$	R_1 :	$\forall \mathsf{X}$: $\mathsf{c(X)}$	$\leq$	$\mathsf{e(X)}$

Figure 5.8: $\mathcal{O}_{Oxi}$ after translation to SLat with monotone operators.

Step 1.2: Based on the axioms from Figure 5.8 we compute a minimal unsatisfiable core of the set $\{A_1, ..., A_3\} \cup \{B_1, ..., B_5\} \cup \{M_1, ..., M_3\} \cup \{R_1\} \cup \{\neg(\mathsf{CO} \leq \mathsf{E})\}$ and obtain the minimal axiom set

$$min = \{A_1, A_2, A_3, B_1, B_5, R_1, M_3\}$$

from which $\mathsf{CO} \leq \mathsf{E}$ can be derived.

Step 2.1: Let $\mathcal{T}_0 = \mathsf{SLat}$ and $\mathcal{T}_1 = \mathsf{SLat} \cup R_1$ be the extension of $\mathcal{T}_0$ with axiom R_1. Since $\mathcal{T}_0 \subseteq \mathcal{T}_1$ is a local theory extension, we can use hierarchical reasoning.

We have the set of ground terms $T = \{\mathsf{c(O)},\ \mathsf{e(P)}\}$.

We use the closure operator $\Psi_{RI}(T)$ described in Section 5.1 to extend our set of ground terms. This means that for every term $\mathsf{c(X)}$ in T we have to add the term $\mathsf{e(X)}$ and vice versa. This leads to the following extended set T' of ground terms:

$$T' = \Psi_{RI}(T) = \{\ \mathsf{c(O)},\ \mathsf{c(P)},\ \mathsf{e(O)},\ \mathsf{e(P)}\ \}$$

From T' we get two instances of the role inclusion axiom R_1 and two instances of the monotonicity axiom M_3:

$$\begin{aligned} I_1 &: \quad \mathsf{c(O)} \leq \mathsf{e(O)} \\ I_2 &: \quad \mathsf{c(P)} \leq \mathsf{e(P)} \\ I_3 &: \quad \mathsf{O} \leq \mathsf{P} \rightarrow \mathsf{e(O)} \leq \mathsf{e(P)} \\ I_4 &: \quad \mathsf{P} \leq \mathsf{O} \rightarrow \mathsf{e(P)} \leq \mathsf{e(O)} \end{aligned}$$

We purify all formulae by introducing new constants for the terms starting with a function symbol (i.e. role names). We save the definitions in the following set:

$$\mathsf{Def} = \{\ \mathsf{c_O} = \mathsf{c(O)},\ \mathsf{c_P} = \mathsf{c(P)},\ \mathsf{e_O} = \mathsf{e(O)},\ \mathsf{e_P} = \mathsf{e(P)}\ \}$$

We then have the set $A_0 \cup B_0 \cup I_0$, where A_0, B_0 and I_0 are the purified versions of $A = \{A_1, A_2, A_3\}$, $B = \{B_1, B_5\}$ and $I = \{I_1, I_2, I_3, I_4\}$, respectively.

Step 2.2: To reduce the number of instances we compute a minimal unsatisfiable core of $A_0 \cup B_0 \cup I_0 \cup \{\neg(\mathsf{CO} \leq \mathsf{E})\}$ and obtain the set of axioms

$$min' = \{A_1, A_2, A_3, B_1, B_5, I_1, I_3\}$$

from which $\mathsf{CO} \leq \mathsf{E}$ can be derived. We then have $\mathcal{H} = \{I_1, I_3\}$. Since $\mathsf{O} \in N_C^A \backslash N_C^B$ and $\mathsf{P} \in N_C^B \backslash N_C^A$, we have $\mathcal{H}_{\mathsf{mix}} = \{I_3\}$.

Step 3: We separate the mixed instance $\mathsf{O} \leq \mathsf{P} \rightarrow \mathsf{e_O} \leq \mathsf{e_P}$ by using Algorithm 4 to find an intermediate term t in the common signature such that $\mathsf{O} \leq t$ and $t \leq \mathsf{P}$. We obtain $t = \mathsf{R}$. Therefore, we get $\mathcal{H}_{\mathsf{sep}} = \{I_3^A, I_3^B\}$, where:

$$\begin{aligned} I_3^A: \quad & \mathsf{O} \leq \mathsf{R} \quad \rightarrow \quad \mathsf{e_O} \leq \mathsf{e_R} \\ I_3^B: \quad & \mathsf{R} \leq \mathsf{P} \quad \rightarrow \quad \mathsf{e_R} \leq \mathsf{e_P} \end{aligned}$$

Note that both I_3^A and I_3^B are instances of the monotonicity axiom for the $\mathsf{enhances}$ role.

Step 4.1: Since for every instance that is necessary to derive the consequence it must be true that $\mathcal{T}_A \cup \mathcal{T}_B$ entails its premise, it is sufficient to consider only the corresponding conclusions. Note that w.r.t. SLat the formula $A_0 \wedge I_1 \wedge I_3^A$ is equivalent to $\overline{A}_0$ and $B_0 \wedge I_3^B$ is equivalent to $\overline{B}_0$, where:

$$\begin{aligned} \overline{A}_0 &= \mathsf{CO} \leq \mathsf{c_O} \wedge \mathsf{CO} \leq \mathsf{S} \wedge \mathsf{O} \leq \mathsf{R} \wedge \mathsf{c_O} \leq \mathsf{e_O} \wedge \mathsf{e_O} \leq \mathsf{e_R} \\ \overline{B}_0 &= \mathsf{R} \leq \mathsf{P} \wedge (\mathsf{S} \wedge \mathsf{e_P}) \leq \mathsf{E} \wedge \mathsf{e_R} \leq \mathsf{e_P} \end{aligned}$$

As a whole, we have the formula $\overline{A}_0 \wedge \overline{B}_0$ with the following sets of symbols:

- $S_A^* = \{\mathsf{CO}, \mathsf{c_O}, \mathsf{O}, \mathsf{e_O}\}$, the set of symbols occurring only in $\overline{A}_0$
- $S_B^* = \{\mathsf{P}, \mathsf{E}, \mathsf{e_P}\}$, the set of symbols occurring only in $\overline{B}_0$
- $S_{AB} = \{\mathsf{S}, \mathsf{R}, \mathsf{e_R}\}$, the set of symbols occurring in $\overline{A}_0$ and $\overline{B}_0$

Using on $\overline{A}_0 \wedge \overline{B}_0$ the translation to propositional logic described in the proof of Theorem 5.11 and using the same names for the SLat-terms and the corresponding variables we obtain the following set N_{AB} of Horn clauses:

$$\begin{aligned} N_{AB} = \ \{\ & (\neg\mathsf{CO} \vee \mathsf{c_O}),\ (\neg\mathsf{CO} \vee \mathsf{S}),\ (\neg\mathsf{O} \vee \mathsf{R}),\ (\neg\mathsf{c_O} \vee \mathsf{e_O}),\ (\neg\mathsf{e_O} \vee \mathsf{e_R}), \\ & (\neg\mathsf{R} \vee \mathsf{P}),\ (\neg\mathsf{S} \vee \neg\mathsf{e_P} \vee \mathsf{E}),\ (\neg\mathsf{e_R} \vee \mathsf{e_P})\ \} \end{aligned}$$

We saturate the set $N_{AB} \cup \{\mathsf{CO}\}$ under positive unit resolution as described in the proof of Theorem 5.11, restricting to inferences in which one clause is a positive unit clause:

- Resolution of CO and $\neg\mathsf{CO} \vee \mathsf{S}$ yields S.
- Resolution of CO and $\neg\mathsf{CO} \vee \mathsf{c_O}$ yields $\mathsf{c_O}$.
- Resolution of $\mathsf{c_O}$ and $\neg\mathsf{c_O} \vee \mathsf{e_O}$ yields $\mathsf{e_O}$.
- Resolution of $\mathsf{e_O}$ and $\neg\mathsf{e_O} \vee \mathsf{e_R}$ yields $\mathsf{e_R}$.
- Resolution of $\mathsf{e_R}$ and $\neg\mathsf{e_R} \vee \mathsf{e_P}$ yields $\mathsf{e_P}$.
- Resolution of S and $\neg\mathsf{S} \vee \neg\mathsf{e_P} \vee \mathsf{E}$ yields $\neg\mathsf{e_P} \vee \mathsf{E}$.
- Resolution of $\mathsf{e_P}$ and $\neg\mathsf{S} \vee \neg\mathsf{e_P} \vee \mathsf{E}$ yields $\neg\mathsf{S} \vee \mathsf{E}$.
- Resolution of S and $\neg\mathsf{S} \vee \mathsf{E}$ yields E.

Two of the obtained resolvents contain only common symbols, namely S and $\mathsf{e_R}$, so $\mathsf{S} \wedge \mathsf{e_R}$ is an intermediate term for $\mathsf{CO} \leq \mathsf{E}$.

Step 4.2: Translating $\mathsf{S} \wedge \mathsf{e_R}$ back to description logic yields the following formula:

$$J = \mathsf{Substance} \ \sqcap \ \exists \mathsf{enhances}.\mathsf{Reaction}$$

Indeed, the following properties hold:

$$\mathcal{T}_A \cup \mathcal{T}_B \cup RI \models \mathsf{CatOxidation} \sqsubseteq J$$
$$\mathcal{T}_A \cup \mathcal{T}_B \cup RI \models J \sqsubseteq \mathsf{Enzyme}$$

Thus, J is a high-level explanation for the subsumption $\mathsf{CatOxidation} \sqsubseteq \mathsf{Enzyme}$. ■

Example 5.19. Consider the ontology $\mathcal{O}_{Med}$ in Figure 5.9, based on an example from [106], which we modified in some points. We changed it in a way that it only contains general concept inclusions and conjunction only appears on the left hand side of an axiom. Furthermore we left out some axioms and concepts, but also added new concepts (LeftVentricle, RightVentricle, Ventricle) and changed some axioms accordingly. We devided the ontology into three parts: A TBox $\mathcal{T}_A$, a TBox $\mathcal{T}_B$ and a set of role axioms RI.

A_1 :	Endocardium	$\sqsubseteq$	Tissue
A_2 :	Endocardium	$\sqsubseteq$	$\exists$part-of.HeartWall
A_3 :	HeartWall	$\sqsubseteq$	BodyWall
A_4 :	HeartWall	$\sqsubseteq$	$\exists$part-of.LeftVentricle
A_5 :	HeartWall	$\sqsubseteq$	$\exists$part-of.RightVentricle
A_6 :	LeftVentricle	$\sqsubseteq$	Ventricle
A_7 :	RightVentricle	$\sqsubseteq$	Ventricle
A_8 :	Endocarditis	$\sqsubseteq$	Inflammation
A_9 :	Endocarditis	$\sqsubseteq$	$\exists$has-location.Endocardium
A_{10} :	Inflammation $\sqcap \exists$has-location.Endocardium	$\sqsubseteq$	Endocarditis
A_{11} :	Inflammation	$\sqsubseteq$	Disease
A_{12} :	Inflammation	$\sqsubseteq$	$\exists$acts-on.Tissue
B_1 :	Ventricle	$\sqsubseteq$	$\exists$part-of.Heart
B_2 :	HeartDisease	$\sqsubseteq$	Disease
B_3 :	HeartDisease	$\sqsubseteq$	$\exists$has-location.Heart
B_4 :	Disease $\sqcap \exists$has-location.Heart	$\sqsubseteq$	HeartDisease
R_1 :	part-of $\circ$ part-of	$\sqsubseteq$	part-of
R_2 :	has-location $\circ$ part-of	$\sqsubseteq$	has-location

Figure 5.9: Ontology $\mathcal{O}_{Med}$.

We have the following sets of symbols (we indicate the abbreviations used in what follows):

$$\begin{aligned} N_C^A &= \{\text{Endocardium (Em), Tissue (T), HeartWall (HW),} \\ &\quad \text{LeftVentricle (LV), RightVentricle (RV), Ventricle (V)} \\ &\quad \text{Disease (D), Inflammation (I), Endocarditis (Es)}\} \\ N_C^B &= \{\text{Heart (H), HeartDisease (HD), Disease (D), Ventricle (V)}\} \\ N_C^{AB} &= \{\text{Disease (D), Ventricle (V)}\} \end{aligned}$$

Consider the subsumption $\mathsf{Endocarditis} \sqsubseteq \mathsf{HeartDisease}$, where $\mathsf{Endocarditis} \in N_C^A$ and $\mathsf{Heartdisease} \in N_C^B$.

Additionally, the following hold:

$$\mathcal{T}_A \cup \mathcal{T}_B \cup RI \models \mathsf{Endocarditis} \sqsubseteq \mathsf{HeartDisease}$$
$$\mathcal{T}_A \cup RI \not\models \mathsf{Endocarditis} \sqsubseteq \mathsf{HeartDisease}$$
$$\mathcal{T}_B \cup RI \not\models \mathsf{Endocarditis} \sqsubseteq \mathsf{HeartDisease}$$

Therefore, we can use Algorithm 5 to compute an intermediate term containing only shared symbols for the subsumption $\mathsf{Endocarditis} \sqsubseteq \mathsf{HeartDisease}$, which serves as an explanation for the subsumption.

Step 1.1: We translate the ontology to the theory of semilattices with monotone operators. We now state the monotonicity axioms for each role explicitly. Figure 5.10 shows the ontology after the translation to the theory of semilattices. Note that from here on we use the abbreviations for concept names indicated in the sets N_C^A, N_C^B and N_C^{AB} above and also abbreviations for role names, i.e. po for part-of, hl for has-location and ao for acts-on.

$$\begin{array}{lrcl} A_1: & \mathsf{Em} & \leq & \mathsf{T} \\ A_2: & \mathsf{Em} & \leq & \mathsf{po(HW)} \\ A_3: & \mathsf{HW} & \leq & \mathsf{BW} \\ A_4: & \mathsf{HW} & \leq & \mathsf{po(LV)} \\ A_5: & \mathsf{HW} & \leq & \mathsf{po(RV)} \\ A_6: & \mathsf{LV} & \leq & \mathsf{V} \\ A_7: & \mathsf{RV} & \leq & \mathsf{V} \\ A_8: & \mathsf{Es} & \leq & \mathsf{I} \\ A_9: & \mathsf{Es} & \leq & \mathsf{hl(Em)} \\ A_{10}: & \mathsf{I} \wedge \mathsf{hl(Em)} & \leq & \mathsf{Es} \\ A_{11}: & \mathsf{I} & \leq & \mathsf{D} \\ A_{12}: & \mathsf{I} & \leq & \mathsf{ao(T)} \end{array} \qquad \begin{array}{lrcl} B_1: & \mathsf{V} & \leq & \mathsf{po(H)} \\ B_2: & \mathsf{HD} & \leq & \mathsf{D} \\ B_3: & \mathsf{HD} & \leq & \mathsf{hl(H)} \\ B_4: & \mathsf{D} \wedge \mathsf{hl(H)} & \leq & \mathsf{HD} \end{array}$$

$$\begin{array}{lrcl} R_1: & \forall \mathsf{X}\colon \mathsf{po(po(X))} & \leq & \mathsf{po(X)} \\ R_2: & \forall \mathsf{X}\colon \mathsf{hl(po(X))} & \leq & \mathsf{hl(X)} \end{array}$$

$$\begin{array}{lrcl} M_1: & \forall \mathsf{X,Y}\colon \mathsf{X} \leq \mathsf{Y} & \rightarrow & \mathsf{po(X)} \leq \mathsf{po(Y)} \\ M_2: & \forall \mathsf{X,Y}\colon \mathsf{X} \leq \mathsf{Y} & \rightarrow & \mathsf{hl(X)} \leq \mathsf{hl(Y)} \\ M_3: & \forall \mathsf{X,Y}\colon \mathsf{X} \leq \mathsf{Y} & \rightarrow & \mathsf{ao(X)} \leq \mathsf{ao(Y)} \end{array}$$

Figure 5.10: $\mathcal{O}_{Med}$ after translation to SLat with monotone operators.

Step 1.2: Based on the axioms from Figure 5.10 we compute a minimal unsatisfiable core of the set $\{A_1, ..., A_{12}\} \cup \{B_1, ..., B_4\} \cup \{M_1, ..., M_3\} \cup \{R_1, R_2\} \cup \{\neg(\mathsf{Es} \leq \mathsf{HD})\}$ and obtain the minimal axiom set

$$min = \{A_2, A_4, A_6, A_8, A_9, A_{11}, B_1, B_4, R_2\}$$

from which $\mathsf{Es} \leq \mathsf{HD}$ can be derived.

This means that for the following instantiation step we only have to consider the role axiom R_2 and none of the monotonicity axioms is needed.

Step 2.1: Let $\mathcal{T}_0 = \mathsf{SLat}$ and $\mathcal{T}_1 = \mathsf{SLat} \cup R_2$ be the extension of $\mathcal{T}_0$ with axiom R_2. We know that $\mathcal{T}_0 \subseteq \mathcal{T}_1$ is a local theory extension, so we can use hierarchical reasoning. We first flatten the role axiom R_2 in the following way:

$$R_2^{\mathsf{flat}}: \quad \forall \mathsf{X},\, \mathsf{Y}\colon \quad \mathsf{X} \leq \mathsf{po(Y)} \quad \rightarrow \quad \mathsf{hl(X)} \leq \mathsf{hl(Y)}$$

The set of ground terms is $T = \{\mathsf{po(HW)}, \mathsf{po(LV)}, \mathsf{po(H)}, \mathsf{hl(Em)}, \mathsf{hl(H)}\}$.

We use the closure operator $\Psi_{RI}(T)$ described in Section 5.1 to extend our set of ground terms. This means that for every term $\mathsf{po(X)}$ in G we have to add the term $\mathsf{hl(X)}$ and vice versa. This leads to the following extended set T' of ground terms:

$$T' = \{\ \mathsf{po(Em)},\ \mathsf{po(HW)},\ \mathsf{po(LV)},\ \mathsf{po(H)},\ \mathsf{hl(Em)},\ \mathsf{hl(HW)},\ \mathsf{hl(LV)},\ \mathsf{hl(H)}\ \}$$

From T' we get the following instances of the axiom R_2^{flat}:

$$\begin{array}{rrcl}
I_1: & \mathsf{Em} \leq \mathsf{po(HW)} & \rightarrow & \mathsf{hl(Em)} \leq \mathsf{hl(HW)} \\
I_2: & \mathsf{Em} \leq \mathsf{po(LV)} & \rightarrow & \mathsf{hl(Em)} \leq \mathsf{hl(LV)} \\
I_3: & \mathsf{Em} \leq \mathsf{po(H)} & \rightarrow & \mathsf{hl(Em)} \leq \mathsf{hl(H)} \\
I_4: & \mathsf{HW} \leq \mathsf{po(Em)} & \rightarrow & \mathsf{hl(HW)} \leq \mathsf{hl(Em)} \\
I_5: & \mathsf{HW} \leq \mathsf{po(LV)} & \rightarrow & \mathsf{hl(HW)} \leq \mathsf{hl(LV)} \\
I_6: & \mathsf{HW} \leq \mathsf{po(H)} & \rightarrow & \mathsf{hl(HW)} \leq \mathsf{hl(H)} \\
I_7: & \mathsf{LV} \leq \mathsf{po(Em)} & \rightarrow & \mathsf{hl(LV)} \leq \mathsf{hl(Em)} \\
I_8: & \mathsf{LV} \leq \mathsf{po(HW)} & \rightarrow & \mathsf{hl(LV)} \leq \mathsf{hl(HW)} \\
I_9: & \mathsf{LV} \leq \mathsf{po(H)} & \rightarrow & \mathsf{hl(LV)} \leq \mathsf{hl(H)} \\
I_{10}: & \mathsf{H} \leq \mathsf{po(Em)} & \rightarrow & \mathsf{hl(H)} \leq \mathsf{hl(Em)} \\
I_{11}: & \mathsf{H} \leq \mathsf{po(HW)} & \rightarrow & \mathsf{hl(H)} \leq \mathsf{hl(HW)} \\
I_{12}: & \mathsf{H} \leq \mathsf{po(LV)} & \rightarrow & \mathsf{hl(H)} \leq \mathsf{hl(LV)}
\end{array}$$

We purify all formulae by introducing new constants for the terms starting with a function symbol, i.e. role names. We save the definitions in the following set:

$$\begin{array}{c}
\mathsf{Def} = \{\mathsf{po_{HW}} = \mathsf{po(HW)}, \mathsf{po_{LV}} = \mathsf{po(LV)}, \mathsf{po_{H}} = \mathsf{po(H)}, \mathsf{hl_{EM}} = \mathsf{hl(EM)}, \\
\mathsf{hl_{HW}} = \mathsf{hl(HW)}, \mathsf{hl_{LV}} = \mathsf{hl(LV)}, \mathsf{hl_{HC}} = \mathsf{hl(HC)}, \mathsf{hl_{H}} = \mathsf{hl(H)}\}
\end{array}$$

We then have the set $A_0 \cup B_0 \cup I_0$, where A_0, B_0 and I_0 are the purified versions of $A = \{A_2, A_4, A_6, A_8, A_9, A_{11}\}$, $B = \{B_1, B_4\}$ and $I = \{I_1, ..., I_{10}\}$, respectively.

Step 2.2: To reduce the number of instances we compute a minimal unsatisfiable core of $A_0 \cup B_0 \cup I_0 \cup \{\neg(\mathsf{Es} \leq \mathsf{HD})\}$ and obtain the set of axioms

$$min' = \{A_2, A_4, A_6, A_8, A_9, A_{11}, B_1, B_4, I_1, I_5, I_9\}$$

from which $\mathsf{Es} \leq \mathsf{HD}$ can be derived. We have $\mathcal{H} = \{I_1, I_5, I_9\}$. The first two instances are not mixed, as their premises contain only symbols in N_C^A. Since $\mathsf{LV} \in N_C^A \backslash N_C^B$ and $\mathsf{H} \in N_C^B \backslash N_C^A$, we have $\mathcal{H}_{\mathsf{mix}} = \{I_9\}$.

Step 3: To separate the mixed instance $\mathsf{LV} \leq \mathsf{po_H} \rightarrow \mathsf{hl_{LV}} \leq \mathsf{hl_H}$ we use Algorithm 4 to compute an intermediate term t in the common signature such that $\mathsf{LV} \leq t$ and $t \leq \mathsf{po_H}$. We obtain $t = \mathsf{V}$. We get $\mathcal{H}_{\mathsf{sep}} = \{I_9^A, I_9^B\}$, where:

$$\begin{array}{rrcl}
I_9^A: & \mathsf{LV} \leq \mathsf{V} & \rightarrow & \mathsf{hl_{LV}} \leq \mathsf{hl_V} \\
I_9^B: & \mathsf{V} \leq \mathsf{po_H} & \rightarrow & \mathsf{hl_V} \leq \mathsf{hl_H}
\end{array}$$

Note that I_9^A is an instance of the monotonicity axiom for the $\mathsf{has\text{-}location}$ role and I_9^B is an instance of axiom R_2^{flat}.

Step 4.1: Since for every instance that is necessary to derive the consequence it is true that $\mathcal{T}_A \cup \mathcal{T}_B$ entails its premise, it is sufficient to consider only the corresponding conclusions. Note that w.r.t. SLat the formula $A_0 \wedge I_1 \wedge I_5 \wedge I_9^A$ is equivalent to $\overline{A}_0$ and the formula $B_0 \wedge I_9^B$ is equivalent to $\overline{B}_0$, where:

$$\begin{aligned}\overline{A}_0 &= \mathsf{Em} \leq \mathsf{po}_{\mathsf{HW}} \wedge \mathsf{HW} \leq \mathsf{po}_{\mathsf{LV}} \wedge \mathsf{LV} \leq \mathsf{V} \wedge \mathsf{Es} \leq \mathsf{I} \wedge \mathsf{Es} \leq \mathsf{hl}_{\mathsf{Em}} \wedge \mathsf{I} \leq \mathsf{D} \\ &\quad \wedge \mathsf{hl}_{\mathsf{EM}} \leq \mathsf{hl}_{\mathsf{HW}} \wedge \mathsf{hl}_{\mathsf{HW}} \leq \mathsf{hl}_{\mathsf{LV}} \wedge \mathsf{hl}_{\mathsf{LV}} \leq \mathsf{hl}_{\mathsf{V}} \\ \overline{B}_0 &= \mathsf{V} \leq \mathsf{po}_{\mathsf{H}} \wedge (\mathsf{D} \wedge \mathsf{hl}_{\mathsf{H}}) \leq \mathsf{HD} \wedge \mathsf{hl}_{\mathsf{V}} \leq \mathsf{hl}_{\mathsf{H}}\end{aligned}$$

As a whole, we have the formula $\overline{A}_0 \wedge \overline{B}_0$ with the following sets of symbols:

- $S_A^* = \{\mathsf{Em}, \mathsf{po}_{\mathsf{HW}}, \mathsf{HW}, \mathsf{po}_{\mathsf{LV}}, \mathsf{LV}, \mathsf{Es}, \mathsf{I}, \mathsf{hl}_{\mathsf{Em}}, \mathsf{hl}_{\mathsf{HW}}, \mathsf{hl}_{\mathsf{LV}}\}$, symbols occurring only in $\overline{A}_0$
- $S_B^* = \{\mathsf{po}_{\mathsf{H}}, \mathsf{hl}_{\mathsf{H}}, \mathsf{HD}\}$, the set of symbols occurring only in $\overline{B}_0$
- $S_{AB} = \{\mathsf{V}, \mathsf{D}, \mathsf{hl}_{\mathsf{V}}\}$, the set of symbols occurring in $\overline{A}_0$ and $\overline{B}_0$

Using the translation to propositional logic described in the proof of Theorem 5.11 we obtain the following set N_{AB} of Horn clauses:

$$\begin{aligned}N_{AB} = \ \{\ &(\neg\mathsf{Em} \vee \mathsf{po}_{\mathsf{HW}}),\ (\neg\mathsf{HW} \vee \mathsf{po}_{\mathsf{LV}}),\ (\neg\mathsf{LV} \vee \mathsf{V}),\ (\neg\mathsf{Es} \vee \mathsf{I}),\ (\neg\mathsf{Es} \vee \mathsf{hl}_{\mathsf{Em}}), \\ &(\neg\mathsf{I} \vee \mathsf{D}),\ (\neg\mathsf{hl}_{\mathsf{EM}} \vee \mathsf{hl}_{\mathsf{HW}}),\ (\neg\mathsf{hl}_{\mathsf{HW}} \vee \mathsf{hl}_{\mathsf{LV}}),\ (\neg\mathsf{hl}_{\mathsf{LV}} \vee \mathsf{hl}_{\mathsf{V}}),\ (\neg\mathsf{V} \vee \mathsf{po}_{\mathsf{H}}), \\ &(\neg\mathsf{D} \vee \neg\mathsf{hl}_{\mathsf{H}} \vee \mathsf{HD}),\ (\neg\mathsf{hl}_{\mathsf{V}} \vee \mathsf{hl}_{\mathsf{H}})\ \}\end{aligned}$$

We saturate the set $N_{AB} \cup \{\mathsf{Es}\}$ under positive unit resolution as described in the proof of Theorem 5.11, restricting to inferences in which one clause is a positive unit clause:

- Resolution of Es and $\neg\mathsf{Es} \vee \mathsf{I}$ yields I.
- Resolution of I and $\neg\mathsf{I} \vee \mathsf{D}$ yields D.
- Resolution of Es and $\neg\mathsf{Es} \vee \mathsf{hl}_{\mathsf{Em}}$ yields $\mathsf{hl}_{\mathsf{Em}}$.
- Resolution of $\mathsf{hl}_{\mathsf{Em}}$ and $\neg\mathsf{hl}_{\mathsf{EM}} \vee \mathsf{hl}_{\mathsf{HW}}$ yields $\mathsf{hl}_{\mathsf{HW}}$.
- Resolution of $\mathsf{hl}_{\mathsf{HW}}$ and $\neg\mathsf{hl}_{\mathsf{HW}} \vee \mathsf{hl}_{\mathsf{LV}}$ yields $\mathsf{hl}_{\mathsf{LV}}$.
- Resolution of $\mathsf{hl}_{\mathsf{LV}}$ and $\neg\mathsf{hl}_{\mathsf{LV}} \vee \mathsf{hl}_{\mathsf{V}}$ yields hl_{V}.
- Resolution of hl_{V} and $\neg\mathsf{hl}_{\mathsf{V}} \vee \mathsf{hl}_{\mathsf{H}}$ yields hl_{H}.
- Resolution of D and $\neg\mathsf{D} \vee \neg\mathsf{hl}_{\mathsf{H}} \vee \mathsf{HD}$ yields $\neg\mathsf{hl}_{\mathsf{H}} \vee \mathsf{HD}$.
- Resolution of hl_{H} and $\neg\mathsf{D} \vee \neg\mathsf{hl}_{\mathsf{H}} \vee \mathsf{HD}$ yields $\neg\mathsf{D} \vee \mathsf{HD}$.
- Resolution of D and $\neg\mathsf{D} \vee \mathsf{HD}$ yields HD.

We obtained two resolvents containing only common symbols: D and hl_{V}. Thus, $\mathsf{D} \wedge \mathsf{hl}_{\mathsf{V}}$ is an intermediate term for $\mathsf{Es} \leq \mathsf{HD}$.

Step 4.2: Translating the formula back to description logic yields the following formula:

$$J = \mathsf{Disease} \ \sqcap \ \exists \mathsf{has\text{-}location}.\mathsf{Ventricle}$$

Indeed, the following properties hold:

$$\begin{aligned}\mathcal{T}_A \cup \mathcal{T}_B \cup RI &\models \mathsf{Endocarditis} \sqsubseteq J \\ \mathcal{T}_A \cup \mathcal{T}_B \cup RI &\models J \sqsubseteq \mathsf{HeartDisease}\end{aligned}$$

Thus, J is a high-level explanation for the subsumption $\mathsf{Endocarditis} \sqsubseteq \mathsf{HeartDisease}$. ■

Evaluation

We evaluate the program on the three examples we have shown, i.e. Examples 5.17, 5.18 and 5.19. For this we measure the execution times of the program and its steps and analyze how the number of axioms and instances changes due to the unsatisfiable core computations in Steps 1 and 2. We compare the two versions used for Step 1, i.e. the heuristic using Z3 and the precise formalization using SPASS. The data obtained when using Z3 for Step 1 is shown in Figure 5.11, the data obtained when using SPASS for Step 1 is shown in Figure 5.12. All times are wall clock times given in milliseconds and rounded to 5 digits after the comma. These are average times obtained by running the program 100 times. Tests were run on a computer with Windows 10, an Intel Core i5-10210U processor and 16 GB RAM.

	$\mathcal{O}_{Amp}$	$\mathcal{O}_{Oxi}$	$\mathcal{O}_{Med}$
Time in total	69,86022 ms	91,71622 ms	117,79624 ms
Time for preprocessing	0,31730 ms	0,48127 ms	0,43949 ms
Time for Step 1	24,23889 ms	23,11613 ms	25,73731 ms
Time for Step 2	21,88105 ms	22,98602 ms	45,45626 ms
Time for Step 3	0,09455 ms	22,39066 ms	22,85824 ms
Time for Step 4	23,32525 ms	22,72220 ms	23,30493 ms
TBox axioms	23	8	16
Role inclusions	0	1	2
Axioms in Z3	25	12	21
Axioms after Step 1	6	7	9
Instances after instantiation	2	6	36
Instances after Step 2	1	2	3
Mixed instances	0	1	1
Axioms after Step 2	6	7	11

Figure 5.11: Evaluation of tests using Z3 for Step 1.

	$\mathcal{O}_{Amp}$	$\mathcal{O}_{Oxi}$	$\mathcal{O}_{Med}$
Time in total	69,22637 ms	92,68173 ms	117,38175 ms
Time for preprocessing	0,25467 ms	0,47784 ms	0,53041 ms
Time for Step 1	25,13976 ms	23,79103 ms	27,06715 ms
Time for Step 2	21,08219 ms	22,91415 ms	42,93753 ms
Time for Step 3	0,06179 ms	22,24924 ms	23,15206 ms
Time for Step 4	22,68795 ms	23,24947 ms	23,68460 ms
TBox axioms	23	8	16
Role inclusions	0	1	2
Axioms in SPASS	23	9	18
Axioms after Step 1	6	7	11
Instances after instantiation	2	6	36
Instances after Step 2	1	2	3
Mixed instances	0	1	1
Axioms after Step 2	6	7	11

Figure 5.12: Evaluation of tests using SPASS for Step 1.

First of all, one can see that the running times of the examples are all really low and do not take much more than 100 milliseconds. Unsurprisingly, the examples containing mixed instances take a little more time than the ones without mixed instances, which is

due to Step 3. Aside from this, the only significant difference in time in a given step is in Step 2, which takes about twice as long for $\mathcal{O}_{Med}$ than for the other two examples. This is due to the fact that for $\mathcal{O}_{Med}$ a lot more instances are computed in this step. Since after unsatisfiable core computation in Step 2 the number of instances is again reduced significantly, this does not affect the running times of the following steps.

A comparison of the two variants of Step 1, i.e. the heuristic using Z3 and the precise formalization in SPASS, shows that the times needed do not differ much. If anything, Step 1 takes slightly longer with SPASS. The unsatisfiable cores computed at the end of Step 1 with Z3 and SPASS are the same for $\mathcal{O}_{Amp}$ and $\mathcal{O}_{Oxi}$, but different for $\mathcal{O}_{Med}$. Here the unsatisfiable core computed with SPASS is a proper superset of the one computed by Z3, so indeed it is not a minimal unsatisfiable core in this case. However, since another (always minimal) unsatisfiable core is computed with Z3 at the end of Step 2 of the implementation, it does not affect the following steps.

5.5 Conclusion

We analyzed a possibility of giving high-level justifications for subsumption in description logics $\mathcal{EL}$ and $\mathcal{EL}^+$. For this, we used the encoding of TBox subsumption as a uniform word problem in classes of semilattices with monotone operators for $\mathcal{EL}$ and the $\leq$-interpolation property in these classes of algebras, as well as extensions to these results in the presence of role inclusions. This can be seen as a first step towards providing short, high-level explanations for subsumption. If more explanations are needed, they can be obtained by pinpointing and analyzing the resolution derivation of the $\leq$-interpolating terms.

Our approach specifically aims at providing explanations for subsumption relations resulting from a combination or extension of ontologies in $\mathcal{EL}$ and $\mathcal{EL}^+$. Pinpointing in $\mathcal{EL}$ and $\mathcal{EL}^+$, as described [10] and [11], is a good method for narrowing down the cause of why a certain unwanted consequence is present in a given TBox $\mathcal{T}$. In [11] also extensions of TBoxes are considered. However, by computing minimal axiom sets one cannot capture the interplay between two TBoxes $\mathcal{T}_A$ and $\mathcal{T}_B$ in the same way that interpolation does. Therefore, we believe that our approach is better suited in such a setting, since an interpolant gives us useful information about the connection between the two TBoxes.

5.5.1 Future Work

In this thesis, we analyzed the property of P-interpolation in theory extensions and formulated the subsumption problem for $\mathcal{EL}$ and $\mathcal{EL}^+$ as a $\leq$-interpolation problem in a theory of semilattices with monotone operators and proposed a method for solving it based on hierarchical reasoning and satisfiability modulo theories. The general approach we propose opens the possibility of applying similar methods to more general classes of non-classical logics, e.g. substructural logics or fuzzy logics with monotone operators studied in [101], or in verification, where one could consider more general extensions than those with uninterpreted function symbols analyzed in [87].

There has been work on other forms of interpolation in the family of $\mathcal{EL}$ description logics. For instance, a variant of interpolation is proved in [76] and possibilities for uniform interpolation are analyzed in [71] and [75] (it is well known that neither $\mathcal{ALC}$ nor $\mathcal{EL}$ allow uniform interpolation). In future work we would like to analyze possibilities of symbol elimination and abduction in such logics, which are related to uniform interpolation.

6 Reasoning About Classes of Graphs

In this chapter we consider certain types of graph classes and transformations thereof and analyze possibilities for reasoning about those classes. We can for instance prove properties of graphs by showing containedness between suitable graph classes. The motivation for the type of graph problems shown here stems from research in wireless network theory.

A *graph* is an ordered pair $G = (V, E)$, where V is a set of *vertices* and $E \subseteq V \times V$ a set of *edges*. One can distinguish two kinds of graphs: *directed graphs*, containing only directed edges, and *undirected graphs*, containing only undirected edges. A *directed edge* (u, v) has an orientation, i.e. an outgoing vertex u and an ingoing vertex v. Thus, in a directed graph the edges (u, v) and (v, u) are different from each other. An *undirected edge* does not have an orientation. Thus, in an undirected graph the edge (u, v) is the same as the edge (v, u) and can therefore also be regarded as the set of vertices $\{u, v\}$. A *subgraph* G' of a graph $G = (V, E)$, denoted $G' \subseteq G$, is a graph $G' = (V', E')$ where $V' \subseteq V$, $E' \subseteq V' \times V'$ and $E' \subseteq E$. A *spanning subgraph* G'' of a graph $G = (V, E)$ is a graph $G'' = (V'', E'')$ where $V'' = V$ and $E'' \subseteq E$.

We look at classes of geometric graphs which occur for instance in wireless networks. These classes can be described using axioms which specify geometric conditions for the existence or non-existence of an edge. The axioms typically refer to points and their coordinates in $\mathbb{R}^2$ and often to distances between points (or costs associated with edges in a graph). For describing geometric conditions we can often use a distance function d, which can for instance be the Euclidean distance or an arbitrary metric (or a cost function). Sometimes we can also use transformations on such graph classes, for example in order to make directed graphs symmetric, i.e. guaranteeing that if a directed edge from a vertex u to a vertex v exists, then an edge from v to u exists as well. Applying such transformations to all graphs of a certain class leads to a new class of graphs.

We are interested in checking the inclusion between two graph classes (possibly obtained from a given graph class using a certain transformation). We will present an approach based on general and property-directed symbol elimination in combination with hierarchical reasoning for checking whether an inclusion holds. If the inclusion cannot be proved, our methods allow us to find a counterexample, i.e. a graph which is contained in the first class but not in the second. In case the problem is parametric in some way, we can generate conditions on the parameters which guarantee that the inclusion holds.

We consider graphs with vertices in a set X, which can for instance be the set of points in the Euclidean space $\mathbb{R}^2$ or the set of points in an abstract metric space (X, d), or at least in a space X with an additional cost function d (possibly having additional properties). For modeling graph classes we use a unary predicate V and a binary predicate E such that

- for every element $x \in X$, $V(x)$ is true if and only if x is a vertex of the graph, and
- for all elements $x, y \in X$, $E(x, y)$ is true if and only if x and y are vertices and there is an edge between x and y.

We first present locality results for extensions with distance functions satisfying the axioms of a metric (or a subset of these axioms). Then we present three simple classes of undirected graphs – namely Gabriel graphs, relative neighborhood graphs and plane drawings – and prove certain relations between them: We prove that every relative neighborhood graph is a spanning subgraph of a Gabriel graph and that Gabriel graphs and relative neighborhood graphs are always plane drawings. Afterwards we look at classes of directed graphs and define transformations on them which make them symmetric. We analyze containedness relations between the graph classes obtained by those transformations.

Parts of the results in this chapter were already published in [18] and [84].

6.1 Locality of Theory Extensions Involving Distances

Many geometric graph classes are described using geometric conditions for the existence or non-existence of edges. In many cases such conditions refer to the distance between vertices or costs associated with pairs of vertices. Therefore, we prove that axiomatizations for such distance or cost functions define local theory extensions.

Let $\mathcal{T}$ be a theory used for formalizing points, distance and vertices, with two sorts p (points, uninterpreted) and num (reals, interpreted). If we model the Euclidian plane with the Euclidean distance, $\mathcal{T}$ can be defined starting from the combination of a theory P of points over $\mathbb{R} \times \mathbb{R}$ and the theory $\mathbb{R}$ of real numbers using a chain of theory extensions

$$(E) \qquad P \cup \mathbb{R} \ \subseteq \ P \cup \mathbb{R} \cup \mathsf{UIF}_{x,y} \ \subseteq \ \mathcal{T} = P \cup \mathbb{R} \cup \mathsf{UIF}_{x,y} \cup \mathcal{T}_d^e$$

where x and y are the coordinate functions, i.e. functions $x : P \to \mathbb{R}$ and $y : P \to \mathbb{R}$ with $x(u)$ describing the x-coordinate and $y(u)$ describing the y-coordinate of point u, and $\mathcal{T}_d^e$ is the theory specifying that $d(u, v)$ is the Euclidean distance between points u and v, i.e. $d(u, v) = \sqrt{(x(u)-x(v))^2+(y(u)-y(v))^2}$. Note that the symbol $\sqrt{}$ is not part of the signature, but the square root function can be defined using an extension by definition as described in Example 2.23. Alternatively, we can express the Euclidean distance using the following two axioms:

(1) $\forall u, v \ \ d(u, v) \geq 0$

(2) $\forall u, v \ \ (d(u, v))^2 \approx (x(u)-x(v))^2+(y(u)-y(v))^2$

If we model arbitrary metric spaces (X, d), $\mathcal{T}$ is the theory extension

$$(M) \qquad P \cup \mathbb{R} \ \subseteq \ \mathcal{T} = P \cup \mathbb{R} \cup \mathcal{T}_d^m$$

where P is the theory of pure equality $\mathcal{E}$[1] and $\mathcal{T}_d^m$ are the axioms of a metric. One can prove that all these extensions are local. In (E) we have extensions with uninterpreted function symbols and definitional extensions, which are local according to Examples 2.21 and 2.23, respectively. In the following we prove that in (M) we have a Ψ-local theory extension for a suitable closure operator Ψ.

We first formalize the properties of metric spaces (X, d), i.e. sets endowed with a distance function d satisfying the usual axioms of a metric, and prove a Ψ-locality property. We then consider also variants that contain only some of these axioms and show that they define Ψ-local extensions as well for suitably defined closure operators.

[1] We later use the theory of an infinite set $\mathcal{T}_{\mathcal{IS}}$, which is the model completion of the theory of pure equality $\mathcal{E}$.

Theorem 6.1. *Let $\mathcal{T}_0$ be the disjoint two-sorted combination of the theory $\mathcal{E}$ of pure equality (sort p) and the theory $LI(\mathbb{R})$ of linear real arithmetic (sort num). Let $\mathcal{T}_d^m$ be the extension of $\mathcal{T}_0$ with a function d with arity $a(d) = \mathsf{p}, \mathsf{p} \to \mathsf{num}$ satisfying the following set $\mathcal{K}_m$ of axioms:*

$$\begin{array}{lll} (d_1) & \forall x, y & d(x,y) \geq 0 \\ (d_2) & \forall x, y, z & d(x,y) \leq d(x,z) + d(z,y) \\ (d_3) & \forall x, y & d(x,y) \approx d(y,x) \\ (d_4) & \forall x, y & x \approx y \to d(x,y) \approx 0 \\ (d_5) & \forall x, y & d(x,y) \approx 0 \to x \approx y \end{array}$$

Let Ψ_m be defined for every set T of ground terms by

$$\Psi_m(T) = T \cup \{d(a,b) \mid a, b \text{ are constants of sort } \mathsf{p} \text{ occurring in } T\}.$$

Then the following hold:

(1) Ψ_m is a closure operator on ground terms.

(2) For every finite set T of ground terms, $\Psi_m(T)$ is finite.

(3) $\mathcal{T}_d^m$ is a Ψ_m-local extension of $\mathcal{T}_0$ satisfying condition (Comp_f^Ψ).

Proof: (1) We have to show that Ψ_m satisfies the conditions from the definition of a closure operator (Definition 2.11):

- Clearly, for every set T of ground terms, T and $\Psi_m(T)$ contain the same constants of sort p, so $\Psi_m(\Psi_m(T)) = \Psi_m(T)$.
- Since the only extension function symbol is d, it holds that $\mathsf{est}(\mathcal{K}, T) \subseteq \Psi_m(T)$ for every set T of ground terms.
- From the definition of Ψ_m it follows that if $T_1 \subseteq T_2$, then we have $\Psi_m(T_1) \subseteq \Psi_m(T_2)$.
- It is easy to check that for every map $h : C \to C$, we have $\overline{h}(\Psi_m(T)) = \Psi_m(\overline{h}(T))$, i.e. Ψ_m is stable under renaming of constants.

(2) If T is finite, then it contains a finite number n of constants. Then $\Psi_m(T)$ has n^2 elements.

(3) To prove that $\mathcal{T}_d^m$ is a Ψ_m-local extension of T_0 we prove that it satisfies the embeddability condition ($\mathsf{Comp}_\mathsf{f}^\Psi$). Let $\mathcal{P} = (P, \mathbb{R}, d_P)$ be a partial model of $\mathcal{T}_d^m = \mathcal{T}_0 \cup \mathcal{K}_m$, where P is the support of sort p, $\mathbb{R}$ the support of sort num, and d_P a partial function from $P \times P$ to $\mathbb{R}$, such that the following properties hold:

(i) All function symbols in Σ_0 are defined everywhere and d is partially defined.

(ii) The set $T(\mathcal{P}) = \{d(a_1, a_2) \mid a_1, a_2 \in P,\ d_P(a_1, a_2) \text{ is defined}\}$ is finite and closed under Ψ_m.

To prove embeddability we have to show that d_P can be extended to a total function on P that satisfies the axioms $\mathcal{K}_m$.

From the axioms of a metric it follows that

- whenever $d_P(p_1, p_2)$ is defined, $d_P(p_1, p_2) \geq 0$;
- if $p_1 \approx p_2$, then $d_P(p_1, p_2) \approx 0$ whenever defined;

- $d_P(p,p) \approx 0$ whenever it is defined;
- if $d_P(p_1,p_2)$ and $d_P(p_2,p_1)$ are defined, then $d_P(p_1,p_2) \approx d_P(p_2,p_1)$; and
- if $d_P(p_1,p_2)$, $d_P(p_2,p_3)$ and $d_P(p_1,p_3)$ are defined, then $d_P(p_1,p_2) \leq d_P(p_2,p_3) + d_P(p_1,p_3)$.

Let $E = \{(p_1,p_2) \mid d_P(p_1,p_2) \text{ is defined}\}$.

Let $P_1 = \{p \in P \mid \exists q \in P : d_P(p,q) \text{ is defined or } d_P(q,p) \text{ is defined}\}$. By the assumption that $d_P(p_1,p_2)$ is defined only for finitely many tuples (p_1,p_2), P_1 is finite and by condition (ii) above (as $T(\mathcal{P})$ is closed under Ψ_m) we have $E = P_1 \times P_1$. Thus, $P = P_1 \cup P_2$ with $d_1 = d_{P|P_1}$ being totally defined and d being defined nowhere on P_2 ($d_P(p_1,p_2)$ is undefined for any two different elements $p_1,p_2 \in P_2$, and for every $p \in P_2$ there is no $q \in P_1$ such that $d_P(p,q)$ or $d_P(q,p)$ is defined). Since P_1 is finite, the maximum distance $m_1 = \mathsf{max}\{d_P(p,q) \mid p,q \in P_1\}$ exists.

Consider an arbitrary distance function d_2 on P_2 such that $\mathsf{sup}\{d_2(p_1,p_2) \mid p_1,p_2 \in P_2\}$ is finite. Such a function is guaranteed to exist, since the distance axioms are consistent: We can for instance bijectively map all points in P_2 to points in the interior of the unit circle and consider the euclidian distances between these points. Thus, the distance function d_2 on P_2 is totally defined and bounded. Let m_2 be such that $d_2(p,q) \leq m_2$ for all $p,q \in P_2$.

We now show how to extend d on $P_1 \cup P_2$. If P_1 or P_2 are empty, we have a total extension of d already. Assume they are both non-empty. Let $p_1 \in P_1$ and $p_2 \in P_2$. We construct a totally defined function $d : (P_1 \cup P_2)^2 \to \mathbb{R}$ as follows:

$$d(p,q) = \begin{cases} d_1(p,q) & \text{if } p,q \in P_1 \\ d_2(p,q) & \text{if } p,q \in P_2 \\ d_0 + d_1(p,p_1) + d_2(p_2,q) & \text{if } p \in P_1 \text{ and } q \in P_2 \\ d_0 + d_1(q,p_1) + d_2(p_2,p) & \text{if } p \in P_2 \text{ and } q \in P_1 \end{cases}$$

where $d_0 \in \mathbb{R}$ is such that $d_0 \approx \mathsf{m} + 1$ with $\mathsf{m} = \mathsf{max}(m_1,m_2)$.

We show that d is a total function that satisfies all the axioms $\mathcal{K}_m$:

- It is clear that d is a total function and that $d(x,y) \geq 0$ for all $x,y \in P_1 \cup P_2$, i.e. it satisfies axiom (d_1).
- Let $p \in P_1 \cup P_2$. Then $p \in P_i$ with $i = 1$ or $i = 2$. Since d_i satisfies axiom (d_4), we have $d(p,p) \approx d_i(p,p) \approx 0$. Thus, d satisfies axiom (d_4).
- Let $p,q \in P_1 \cup P_2$. We make a case distinction:

  Case 1: $p,q \in P_i$ for $i = 1$ or $i = 2$.

  Then $d(p,q) \approx d_i(p,q) \approx d_i(q,p) \approx d(q,p)$, since d_i satisfies axiom (d_3).

  Case 2: $p \in P_1$ and $q \in P_2$.

  Then $d(p,q) \approx d_0 + d_1(p,p_1) + d_2(p_2,q) \approx d(q,p)$ follows immediately from the definition of d.

  Case 3: $p \in P_2$ and $q \in P_1$.

  Then $d(p,q) \approx d_0 + d_1(q,p_1) + d_2(p_2,p) \approx d(q,p)$ follows immediately from the definition of d.

  Thus, d satisfies axiom (d_3).

- Let $p, q \in P_1 \cup P_2$. We make a case distintion:

  Case 1: $p, q \in P_i$ with $i = 1$ or $i = 2$.

  If $d(p, q) \approx 0$, then $d_i(p, q) \approx 0$, so as d_i satisfies axiom (d_5), we have $p \approx q$.

  Case 2: $p \in P_1$ and $q \in P_2$, or $p \in P_2$ and $q \in P_1$.

  Then by definition $d(p, q) \geq d_0 > 0$, so we cannot have $d(p, q) \approx 0$.

  Thus, d satisfies axiom (d_5).

- We show that d satisfies the triangle inequality (axiom (d_2)).

  Let $p, q, r \in P_1 \cup P_2$. We show that $d(p, q) \leq d(p, r) + d(r, q)$. We distinguish the following cases:

  Case 1: $p \in P_1, q \in P_2$. Then $d(p, q) \approx d_0 + d_1(p, p_1) + d_2(p_2, q)$.

  Subcase 1a: $r \in P_1$.

  Then $d(p, r) + d(r, q) \approx d_1(p, r) + d_0 + d_1(r, p_1) + d_2(p_2, q) \geq d_0 + d_1(p, p_1) + d_2(p_2, q) \approx d(p, q)$.

  Subcase 1b: $r \in P_2$.

  Then $d(p, r) + d(r, q) \approx d_0 + d_1(p, p_1) + d_2(p_2, r) + d_2(r, q) \geq d_0 + d_1(p, p_1) + d_2(p_2, q) \approx d(p, q)$.

  Case 2: $p \in P_2, q \in P_1$. Then $d(p, q) \approx d_0 + d_1(q, p_1) + d_2(p_2, p)$.

  Subcase 2a: $r \in P_2$.

  Then $d(p, r) + d(r, q) \approx d_2(p, r) + d_0 + d_1(r, p_2) + d_2(p_1, q) \approx d_2(r, p) + d_0 + d_1(q, p_1) + d_2(p_2, r) \geq d_0 + d_1(q, p_1) + d_2(p_2, p) \approx d(p, q)$.

  Subcase 2b: $r \in P_1$.

  Then $d(p, r) + d(r, q) \approx d_0 + d_2(p, p_2) + d_2(p_1, r) + d_1(r, q) \geq d_0 + d_1(p_1, q) + d_2(p, p_2) \approx d_0 + d_1(q, p_1) + d_2(p_2, p) \approx d(p, q)$.

  Case 3: $p, q \in P_1$. Then $d(p, q) \approx d_1(p, q)$.

  Subcase 3a: $r \in P_1$.

  Then $d(p, q) \approx d_1(p, q) \leq d_1(p, r) + d_1(r, q) \approx d(p, r) + d(r, q)$, since d_1 satisfies axiom (d_2).

  Subcase 3b: $r \in P_2$.

  Then $d(p, q) \approx d_1(p, q) \leq \mathsf{m} < d_0 \leq d(p, r) + d(r, q)$.

  Case 4: $p, q \in P_2$. Then $d(p, q) \approx d_2(p, q)$.

  Subcase 4a: $r \in P_2$.

  Then $d(p, q) \approx d_2(p, q) \leq d_2(p, r) + d_2(r, q) \approx d(p, r) + d(r, q)$, since d_2 satisfies axiom (d_2).

  Subcase 4b: $r \in P_1$.

  Then $d(p, q) \approx d_2(p, q) \leq \mathsf{m} < d_0 \leq d(p, r) + d(r, q)$.

In [64] it was proved that condition $(\mathsf{Comp}_{\mathsf{f}}^{\Psi})$ for $\mathcal{T}_0 \subseteq \mathcal{T}_0 \cup \mathcal{K}$ implies Ψ-locality of the extension if the clauses in $\mathcal{K}$ are flat and linear. The clauses in $\mathcal{K}_m$ are flat, but are not linear. In the proof of the fact that embeddability implies locality linearity is needed in order to ensure that if we have a model $\mathcal{B}$ of $\mathcal{T}_0 \cup \mathcal{K}[\Psi(G)] \cup G$, we can define a partial model $\mathcal{A}$ of $\mathcal{T}_0 \cup \mathcal{K} \cup G$ and argue that (by $(\mathsf{Comp}_{\mathsf{f}}^{\Psi})$) this model embeds into a total model of $\mathcal{T}_0 \cup \mathcal{K} \cup G$.

We construct $\mathcal{A}$ as follows: Its universe(s) are the same as for $\mathcal{B}$, and $f(a_1, \ldots, a_n)$ is defined in $\mathcal{A}$ if there exist constants $c_1, \ldots, c_n$ which interpret in $\mathcal{A}$ as $a_1, \ldots, a_n$ and $f(c_1, \ldots, c_n)$ occurs in $\Psi(G)$. This definition is used to associate with every valuation in $\mathcal{A}$ in which all terms in a clause C are defined a substitution σ such that $C\sigma \in \mathcal{K}[\Psi(G)]$.

If the clause C is linear, the substitution can be defined without problems. If C contains a variable in different terms, it might be difficult to define Σ because for different occurrences of x we might find different suitable terms.

This problem does not occur here because of the fact that Ψ_m adds all necessary instances that allow to define σ without problems. □

We can still obtain local theory extensions if we leave out some of the axioms of a metric. Below we first consider extensions with a function d in which all the axioms of a metric except for the triangle inequality hold. Afterwards we consider extensions with an uninterpreted function d and extensions with a function d satisfying only the positivity axiom d_1 or only the symmetry axiom d_3.

Theorem 6.2. *Let $\mathcal{T}_0$ be the disjoint two-sorted combination of the theory $\mathcal{E}$ of pure equality (sort p) and the theory $LI(\mathbb{R})$ of linear real arithmetic (sort num). Let $\mathcal{T}_d^n$ be the extension of $\mathcal{T}_0$ with a function d with arity $a(d) = \mathsf{p}, \mathsf{p} \to \mathsf{num}$ satisfying the following set $\mathcal{K}_n$ of axioms:*

$$\begin{array}{lll} (d_1) & \forall x, y & d(x,y) \geq 0 \\ (d_3) & \forall x, y & d(x,y) \approx d(y,x) \\ (d_4) & \forall x, y & x \approx y \to d(x,y) \approx 0 \\ (d_5) & \forall x, y & d(x,y) \approx 0 \to x \approx y \end{array}$$

Let Ψ_n be defined for every set T of ground terms by

$$\begin{array}{rcl} \Psi_n(T) = T & \cup & \{d(t_2, t_1) \mid d(t_1, t_2) \in T\} \\ & \cup & \{d(a,a) \mid a \text{ is a constant of sort } \mathsf{p} \text{ occurring in } T\}. \end{array}$$

Then $\mathcal{T}_d^n$ is a Ψ_n-local extension of $\mathcal{T}_0$.

Proof: To prove locality we have to show that every partial model of $\mathcal{T}_d^n = \mathcal{T}_0 \cup \mathcal{K}_n$ which is closed under Ψ_n can be extended to a total model. Let $\mathcal{P} = (P, \mathbb{R}, d_P)$ be a partial model of $\mathcal{T}_d^n = \mathcal{T}_0 \cup \mathcal{K}_n$ (where P is the support of sort p, $\mathbb{R}$ the support of sort num, and d_P a partial function from $P \times P$ to $\mathbb{R}$) satisfying the conditions above. We construct a total function $d : P \times P \to \mathbb{R}$ as follows:

$$d(p,q) = \begin{cases} d_P(p,q) & \text{if } d_P(p,q) \text{ is defined} \\ 0 & \text{if } d_P(p,q) \text{ is not defined and } p \approx q \\ 1 & \text{if } d_P(p,q) \text{ is not defined and } p \not\approx q \end{cases}$$

It is easy to check that d satisfies all the axioms in $\mathcal{K}_n$. The considerations in the previous proof can be used also in this case to show that embeddability implies locality in spite of the non-linearity due to the choice of the closure operator. □

Theorem 6.3. *Let $\mathcal{T}_0$ be the disjoint two-sorted combination of the theory $\mathcal{E}$ of pure equality (sort p) and the theory $LI(\mathbb{R})$ of linear real arithmetic (sort num). The following extensions of $\mathcal{T}_0$ with a function d (sort $\mathsf{p}\times\mathsf{p}\to\mathsf{num}$) are Ψ-local, with Ψ being the identity function.*

(i) $\mathcal{T}_d^u$, *the extension of* $\mathcal{T}_0$ *with an uninterpreted function* d.

(ii) $\mathcal{T}_d^p = \mathcal{T}_0 \cup \mathcal{K}_p$, *where* $\mathcal{K}_p = \{\forall x, y\ d(x,y) \geq 0\}$.

The extension $\mathcal{T}_d^s = \mathcal{T}_0 \cup \mathcal{K}_s$, *where* $\mathcal{K}_s = \{\forall x, y\ d(x,y) \approx d(y,x)\}$ *is* Ψ_s*-local, where* $\Psi_s(T) = T \cup \{d(a,b) \mid d(b,a) \in T\}$.

Proof: (i) and (ii) are local theory extensions according to Examples 2.21 and 2.23, respectively. The locality proof for $\mathcal{T}_d^s$ is similar to the one for $\mathcal{T}_d^n$. □

We present all the results together in the following theorem:

Theorem 6.4. *Let $\mathcal{T}_0$ be the disjoint two-sorted combination of the theory $\mathcal{E}$ of pure equality (sort p) and the theory $LI(\mathbb{R})$ of linear real arithmetic (sort num). The following extensions of $\mathcal{T}_0$ with a function d (sort $\mathsf{p}\times\mathsf{p}\to\mathsf{num}$) are Ψ-local for a suitable closure operator Ψ:*

(1) $\mathcal{T}_d^m = \mathcal{T}_0 \cup \mathcal{K}_m$, *where* $\mathcal{K}_m$ *are the axioms of a metric, is* Ψ_m*-local, where*

$$\Psi_m(T) = T \cup \{d(a,b) \mid a, b \text{ constants of sort } \mathsf{p} \text{ occurring in } T\}.$$

(2) $\mathcal{T}_d^n = \mathcal{T}_0 \cup \mathcal{K}_n$, *where* $\mathcal{K}_n$ *contains all axioms of a metric except for the triangle inequality, is* Ψ_n*-local, where*

$$\Psi_n(T) = T \cup \{d(b,a) \mid d(a,b) \in T\} \cup \{d(a,a) \mid a \text{ constant of sort } \mathsf{p} \text{ occurring in } T\}.$$

(3) $\mathcal{T}_d^u$, *the extension of* $\mathcal{T}_0$ *with an uninterpreted function* d, *is* Ψ*-local, where* $\Psi(T) = T$ *(i.e. it is local).*

(4) $\mathcal{T}_d^p = \mathcal{T}_0 \cup \mathcal{K}_p$, *where* $\mathcal{K}_p = \{\forall x, y\ d(x,y) \geq 0\}$, *is* Ψ*-local, where* $\Psi(T) = T$ *(i.e. it is local).*

(5) $\mathcal{T}_d^s = \mathcal{T}_0 \cup \mathcal{K}_s$, *where* $\mathcal{K}_s = \{\forall x, y\ d(x,y) = d(y,x)\}$, *is* Ψ_s*-local, where*

$$\Psi_s(T) = T \cup \{d(a,b) \mid d(b,a) \in T\}.$$

6.2 Graph Classes Related to Planarity Conditions

In the following we present a type of graph classes which can be proved to be plane drawings, i.e. they can be drawn in the plane such that none of their edges intersect. We here assume that the graphs we consider are undirected and do not contain self-loops. These properties can be described by the axioms $\mathcal{K}_0$:

$$\mathcal{K}_0 = \{\ \forall x\,(\neg E(x,x)),\ \ \forall x, y\,(E(x,y) \to E(y,x)),\ \ \forall x, y\,(E(x,y) \to V(x) \wedge V(y))\ \}$$

We consider the following three classes of graphs:

- The class **P** of plane drawings
- The class **G** of Gabriel graphs [20]
- The class **R** of relative neighborhood graphs [70]

We also look at classes $\mathbf{G}^{\to}$ and $\mathbf{R}^{\to}$ containing all the spanning subgraphs of graphs in **G** and **R**, respectively.

Plane Drawings

A plane drawing is a graph which can be drawn in the plane in such a way that no edges of the graph intersect. Guaranteeing that the graph is intersection-free means that if there is an edge between two vertices w and s, and there are two vertices u and v such that the line segments uv and ws intersect, then an edge between u and v cannot exist. This can be described by a suitable formula.

The **class P of plane drawings** is the set of graphs satisfying the following axiom:

(P) $\quad \forall u, v, w, s : E(w, s) \wedge \pi_{\mathbf{P}}(u, v, w, s) \to \neg E(u, v)$

Here, $\pi_{\mathbf{P}}(u, v, w, s)$ is a formula which is true if and only if u, v, w and s are all vertices, w and s are in different half-planes defined by the line uv passing through u and v, and u and v are in different half-planes defined by the line ws passing through w and s.[2] If we express the fact that a point p lies on the segment xy by $d(x, p) + d(p, y) \approx d(x, y)$, then we can express $\pi_{\mathbf{P}}(u, v, w, s)$ (if d is a metric) by the following formula:

$$\begin{aligned}\pi_{\mathbf{P}}(u, v, w, s) \quad \coloneqq \quad & V(u) \wedge V(v) \wedge V(s) \wedge V(w) \wedge u \not\approx v \wedge w \not\approx s \wedge \\ & \exists m(d(u, m) + d(m, v) \approx d(u, v) \wedge d(w, m) + d(m, s) \approx d(w, s))\end{aligned}$$

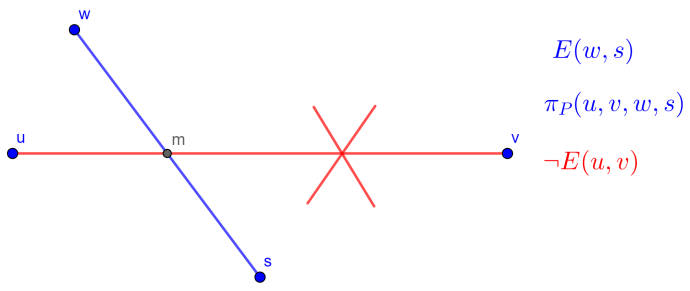

Figure 6.1: Condition for a plane drawing.

Gabriel Graphs

A Gabriel graph is a graph in which an edge between two vertices u and v exists if and only if there is no other vertex inside the closed disk with diameter uv. We define the class of Gabriel graphs as the (infinite) set of all possible Gabriel graphs.

The **class G of Gabriel graphs** is axiomatized by $\mathcal{K}_0$ together with

(G) $\quad \forall u, v \; E(u, v) \leftrightarrow \pi_{\mathbf{G}}(u, v)$

where $\pi_{\mathbf{G}}(u, v)$ expresses the fact that u and v are vertices and every vertex different from u and v lies outside of the minimal circle passing through u and v. If $m(u, v)$ is the middle of the segment uv, we can express this by

$$\begin{aligned}\pi_{\mathbf{G}}(u, v) \coloneqq \quad & V(u) \wedge V(v) \wedge u \not\approx v \wedge \\ & \forall w \left(w \not\approx u \wedge w \not\approx v \wedge V(w) \to d(m(u, v), w) > d(m(u, v), u)\right).\end{aligned}$$

[2] For the existence of these lines it is necessary that $u \neq v$ and $w \neq s$. If, for instance, the vertex w is located on the line through u and v, but not equal to u or v, it is located in both half-planes; hence, in this case the segments uv and ws intersect.

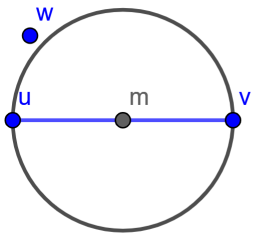

Figure 6.2: Inclusion condition for an edge in a Gabriel graph.

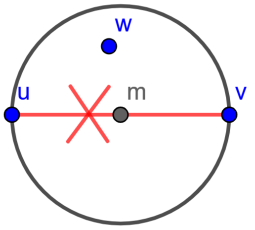

Figure 6.3: Exclusion condition for an edge in a Gabriel graph.

We can define a superclass $\mathbf{G}^{\rightarrow}$ by keeping the "$\rightarrow$" implication in (G) and leaving out the "$\leftarrow$" implication. The superclass $\mathbf{G}^{\rightarrow}$ contains all the spanning subgraphs of a Gabriel graph.

Relative Neighborhood Graphs

A relative neighborhood graph is a graph in which an edge between two vertices u and v exists if and only if there is no other vertex inside the intersection of the open disks around u and v with radius $|uv|$. We define the class of relative neighborhood graphs as the (infinite) set of all possible relative neighborhood graphs.

The **class R of relative neighborhood graphs** is axiomatized by $\mathcal{K}_0$ together with

(R) $\quad \forall u, v \; E(u,v) \leftrightarrow \pi_{\mathbf{R}}(u,v)$

where $\pi_{\mathbf{R}}$ expresses the fact that u and v are vertices which are different from each other and all other vertices are farther away from u than v and farther away from v than u. Geometrically this means that there is no vertex w inside of the intersecting area of the open disk with center u and radius $|uv|$ and the open disk with center v and radius $|uv|$, which can be expressed by

$$\pi_{\mathbf{R}}(u,v) \coloneqq V(u) \wedge V(v) \wedge u \not\approx v \wedge \forall w \, (V(w) \rightarrow d(u,w) \geq d(u,v) \vee d(v,w) \geq d(u,v)).$$

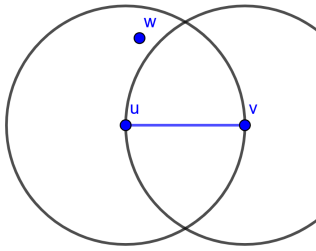

Figure 6.4: Inclusion condition for an edge in a relative neighborhood graph.

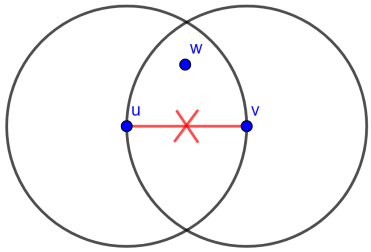

Figure 6.5: Exclusion condition for an edge in a relative neighborhood graph.

We can define a superclass $\mathbf{R}^{\rightarrow}$ by only keeping the "$\rightarrow$" implication in (R) and leaving out the "$\leftarrow$" implication. The superclass $\mathbf{R}^{\rightarrow}$ contains all the spanning subgraphs of a relative neighborhood graph.

6.2.1 Proof Tasks

We are interested in proving certain properties of graphs and relationships between the graph classes described above. We can first observe that a Gabriel graph or a relative neighborhood graph can never have any intersecting edges, i.e. they are plane drawings. The overall goal will be to prove automatically that all graphs in $\mathbf{G}$ and $\mathbf{R}$ are plane drawings. This can be done by showing that the respective class of graphs is contained in the class of plane drawings.

As an intermediate step we will also analyze the relationship between the classes of Gabriel graphs and relative neighborhood graphs. For this we look first at Figure 6.6.

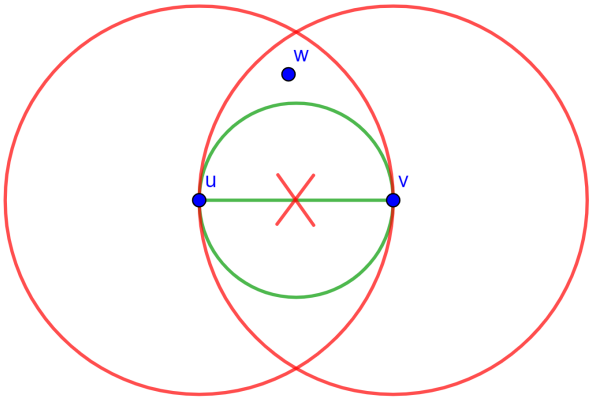

Figure 6.6: Relation between the graph classes $\mathbf{G}$ and $\mathbf{R}$.

We can observe the following:

- If some vertex lies inside the smallest circle passing through vertices u and v (green circle), then this vertex also lies inside the intersecting area of the open disk around u with radius uv and the open disk around v with radius uv (defined by the red circles). Therefore, if an edge is not allowed to exist in a Gabriel graph, then it also does not exist in the corresponding relative neighborhood graph.
- If all vertices are located outside of the intersecting area of the open disks defined by the red circles, then they are also located outside of the green circle. Therefore, if an edge exists in a relative neighborhood graph, then it also exists in the corresponding Gabriel graph.
- If a vertex w is located inside the intersecting area of the the open disks defined by the red circles, but outside of the green circle (as shown in Figure 6.6), then the edge between u and v exists in the Gabriel graph, but not in the corresponding relative neighborhood graph.

We can conclude that, for a given set of vertices, the relative neighborhood graph obtained from the set of vertices may be missing some of the edges of the corresponding Gabriel graph that is obtained from the same set of vertices. This means that a relative neighborhood graph will always be a spanning subgraph of the corresponding Gabriel graph. This can also be proved automatically by showing that the class of all relative neighborhood graphs ($\mathbf{R}$) is contained in the class of spanning subgraphs of Gabriel graphs ($\mathbf{G}^{\rightarrow}$). Since $\mathbf{R} \subseteq \mathbf{R}^{\rightarrow}$, it is sufficient to prove that $\mathbf{R}^{\rightarrow}$ is contained in $\mathbf{G}^{\rightarrow}$. It can also be shown that this class inclusion does not hold in the opposite direction.

In what follows we will prove the following class inclusions in the order given here:

(1) $\mathbf{R}^{\rightarrow} \subseteq \mathbf{G}^{\rightarrow}$

(2) $\mathbf{G}^{\rightarrow} \not\subseteq \mathbf{R}^{\rightarrow}$

(3) $\mathbf{G} \subseteq \mathbf{P}$

(4) $\mathbf{R} \subseteq \mathbf{P}$

For the proofs we will first interpret the distance function d to be the Euclidean distance and prove (1)-(4). In some cases we will also analyze the inclusion for a general metric d. We will see that for a general metric the results may differ, as (1) can for instance not be proved in this case. If class inclusion cannot be proved, we will compute counterexamples.

6.2.2 Checking Graph Class Inclusion for Simple Graph Classes

In the following we explain our method for checking inclusion of graph classes. Let $\mathsf{Ax}_{\mathbf{C}_1}$ and $\mathsf{Ax}_{\mathbf{C}_2}$ be axiomatizations for graph classes $\mathbf{C}_1$ and $\mathbf{C}_2$ using predicates π_{C_1} and π_{C_2}, respectively. We know that $\mathbf{C}_1 \subseteq \mathbf{C}_2$ if and only if $\mathsf{Ax}_{\mathbf{C}_1} \wedge \neg\mathsf{Ax}_{\mathbf{C}_2}$ is unsatisfiable. To check satisfiability we use a two-layered approach:

a) Use *symbol elimination* to eliminate the predicate E in order to obtain conditions only containing the predicates π_{C_1} and π_{C_2}.

b) Use *hierarchical reasoning* to reduce the problem to the base theory and check satisfiability of the reduced problem.

Note that for step a) we consider π_{C_1} and π_{C_2} to be abstract predicates. For step b) we then use the geometric interpretation of these predicates.

Symbol Elimination

We present some simple examples in which symbol elimination allows us to determine constraints on abstract predicates used in the description of the classes under which inclusions hold. We consider the type of axioms occuring in the classes $\mathbf{P}$, $\mathbf{G}$ and $\mathbf{R}$ using a binary predicate E for the edges and (abstract) predicates representing additional conditions.

Example 6.5. Consider the classes $\mathbf{C}_1$ and $\mathbf{C}_2$ described by axioms of the following form:

$$\mathsf{Ax}_{\mathbf{C}_i}: \ \forall u, v \ E(u,v) \leftrightarrow \pi_i(u,v) \qquad i = 1, 2$$

We know that $\mathbf{C}_1 \subseteq \mathbf{C}_2$ if and only if $\mathsf{Ax}_{\mathbf{C}_1} \wedge \neg\mathsf{Ax}_{\mathbf{C}_2}$ is unsatisfiable. We have:

$$\begin{aligned} \mathsf{Ax}_{\mathbf{C}_1} \wedge \neg\mathsf{Ax}_{\mathbf{C}_2} \equiv \ & \forall u, v \ (E(u,v) \leftrightarrow \pi_1(u,v)) \wedge \exists a, b \ (E(a,b) \wedge \neg\pi_2(a,b)) \quad \vee \\ & \forall u, v \ (E(u,v) \leftrightarrow \pi_1(u,v)) \wedge \exists a, b \ (\neg E(a,b) \wedge \pi_2(a,b)) \end{aligned}$$

Consider the second-order formula $\exists E \ (\mathsf{Ax}_{\mathbf{C}_1} \wedge \neg\mathsf{Ax}_{\mathbf{C}_2})$. This formula is equivalent to the formula $\exists a, b \ ((\pi_1(a,b) \wedge \neg\pi_2(a,b)) \vee (\neg\pi_1(a,b) \wedge \pi_2(a,b)))$ which does not contain the existentially quantified predicate E:

$$\begin{aligned} & \exists E \ (\forall u, v \ (E(u,v) \leftrightarrow \pi_1(u,v)) \wedge \exists a, b \ (E(a,b) \wedge \neg\pi_2(a,b))) \vee \\ & \quad (\forall u, v \ (E(u,v) \leftrightarrow \pi_1(u,v)) \wedge \exists a, b \ (\neg E(a,b) \wedge \pi_2(a,b))) \\ \equiv \ & \exists a, b \ ((\pi_1(a,b) \wedge \neg\pi_2(a,b)) \vee (\neg\pi_1(a,b) \wedge \pi_2(a,b))). \end{aligned}$$

Proof: For better readability, in what follows we express the predicates E, π_1 and π_2 as binary functions with codomain $\{0,1\}$ (where value 0 means the predicate is false and value 1 means it is true). We prove by hand that the equivalence above holds, i.e. we show that for any structure $\mathcal{A}$ over the signature $\Pi = (\{\pi_1, \pi_2\}, \emptyset)$ we have $\mathcal{A} \models F$ if and only if $\mathcal{A} \models G$, where:

$$\begin{aligned} F = \quad & \exists E\ (\forall u, v\ (E(u,v) \approx 1 \leftrightarrow \pi_1(u,v) \approx 1) \wedge \exists a, b\ (E(a,b) \approx 1 \wedge \pi_2(a,b) \approx 0)) \vee \\ & (\forall u, v\ (E(u,v) \approx 1 \leftrightarrow \pi_1(u,v) \approx 1) \wedge \exists a, b\ (E(a,b) \approx 0 \wedge \pi_2(a,b) \approx 1)) \\ G = \quad & \exists a, b\ ((\pi_1(a,b) \approx 1 \wedge \pi_2(a,b) \approx 0) \vee (\pi_1(a,b) \approx 0 \wedge \pi_2(a,b) \approx 1)) \end{aligned}$$

1. ("⇒"): We show that $\mathcal{A} \models F$ implies $\mathcal{A} \models G$.

Assume $\mathcal{A} \models F$. Then there exists a structure $\mathcal{B}$ with the same universe and the same interpretations for π_1 and π_2 as $\mathcal{A}$ and, in addition, an interpretation $E_\mathcal{B}$ for E such that

(1) $\mathcal{B} \models \forall u, v\ (E(u,v) \approx 1 \leftrightarrow \pi_1(u,v) \approx 1)$ and

(2) $\mathcal{B} \models \exists a, b\ ((E(a,b) \approx 1 \wedge \pi_2(a,b) \approx 0) \vee (E(a,b) \approx 0 \wedge \pi_2(a,b) \approx 1))$.

Then, by (1), $E_\mathcal{B} = \pi_{1\mathcal{B}}$. For (2) we make a case distinction.

Case 1: $\mathcal{B} \models \exists a, b\ (E(a,b) \approx 1 \wedge \pi_2(a,b) \approx 0)$.

Then, since $E_\mathcal{B} = \pi_{1\mathcal{B}}$, we have $\mathcal{B} \models \exists a, b\ (\pi_1(a,b) \approx 1 \wedge \pi_2(a,b) \approx 0)$. Since all the predicate symbols except for E are interpreted the same in $\mathcal{A}$ and $\mathcal{B}$, we then also have $\mathcal{A} \models \exists a, b\ (\pi_1(a,b) \approx 1 \wedge \pi_2(a,b) \approx 0)$.

Case 2: $\mathcal{B} \models \exists a, b\ (E(a,b) \approx 0 \wedge \pi_2(a,b) \approx 1)$.

Then, since $E_\mathcal{B} = \pi_{1\mathcal{B}}$, we have $\mathcal{B} \models \exists a, b\ (\pi_1(a,b) \approx 0 \wedge \pi_2(a,b) \approx 1)$. Since all the predicate symbols except for E are interpreted the same in $\mathcal{A}$ and $\mathcal{B}$, we then also have $\mathcal{A} \models \exists a, b\ (\pi_1(a,b) \approx 0 \wedge \pi_2(a,b) \approx 1)$.

Thus, $\mathcal{A} \models G$, so $\mathcal{A} \models F$ implies $\mathcal{A} \models G$.

2. ("⇐"): We show that $\mathcal{A} \models G$ implies $\mathcal{A} \models F$.

Assume $\mathcal{A} \models G$. We define a new structure $\mathcal{B}$ which has the same universe and the same interpretations for π_1 and π_2 as $\mathcal{A}$ and, in addition, an interpretation $E_\mathcal{B}$ with $E_\mathcal{B} = \pi_{1\mathcal{A}}$. We make a case distinction:

Case 1: $\mathcal{A} \models \exists a, b\ (\pi_1(a,b) \approx 1 \wedge \pi_2(a,b) \approx 0)$.

Then the following hold:

(1) $\mathcal{B} \models \forall u, v\ (E(u,v) \approx 1 \leftrightarrow \pi_1(u,v) \approx 1)$ and

(2) $\mathcal{B} \models \exists a, b\ (E(a,b) \approx 1 \wedge \pi_2(a,b) \approx 0)$.

Then, since $\mathcal{A}$ is interpreted the same as $\mathcal{B}$ on all predicate symbols except for E and there exists an interpretation $E_\mathcal{B}$ with (1) and (2), we have

$$\mathcal{A} \models \exists E\ (\forall u, v\ (E(u,v) \approx 1 \leftrightarrow \pi_1(u,v) \approx 1) \wedge \exists a, b\ (E(a,b) \approx 1 \wedge \pi_2(a,b) \approx 0)).$$

Case 2: $\mathcal{A} \models \exists a, b\ (\pi_1(a,b) \approx 0 \wedge \pi_2(a,b) \approx 1)$.

Then the following hold:

(1) $\mathcal{B} \models \forall u, v\ (E(u,v) \approx 1 \leftrightarrow \pi_1(u,v) \approx 1)$ and

(2) $\mathcal{B} \models \exists a, b\ (E(a,b) \approx 0 \wedge \pi_2(a,b) \approx 1)$.

Then, since $\mathcal{A}$ is interpreted the same as $\mathcal{B}$ on all predicate symbols except for E and there exists an interpretation $E_{\mathcal{B}}$ with (1) and (2), we have

$$\mathcal{A} \models \exists E\ (\forall u, v\ (E(u,v) \approx 1 \leftrightarrow \pi_1(u,v) \approx 1) \wedge \exists a, b\ (E(a,b) \approx 0 \wedge \pi_2(a,b) \approx 1)).$$

Thus, $\mathcal{A} \models F$, so $\mathcal{A} \models G$ implies $\mathcal{A} \models F$. □

We have shown that $\mathbf{C}_1 \subseteq \mathbf{C}_2$ if and only if $(\pi_1(a,b) \wedge \neg\pi_2(a,b)) \vee (\neg\pi_1(a,b) \wedge \pi_2(a,b))$ is unsatisfiable w.r.t. $\mathcal{T}$, where a and b are Skolem constants introduced for the existentially quantified variables of the same name.

We can obtain this result also using SEH-PILoT. Since SEH-PILoT cannot eliminate predicate symbols, we interpret the predicates E, π_1 and π_2 as binary functions with codomain $\{0,1\}$ (where value 0 means the predicate is false and value 1 means it is true), then we can treat π_1 and π_2 as parameters and use a variant of Algorithm 2 to eliminate E in the formula

$$\begin{aligned}\exists E\ (\forall u, v\ (E(u,v) \approx 1 \leftrightarrow \pi_1(u,v) \approx 1) \wedge \exists a, b\ (E(a,b) \approx 1 \wedge \pi_2(a,b) \approx 0)) \vee \\ (\forall u, v\ (E(u,v) \approx 1 \leftrightarrow \pi_1(u,v) \approx 1) \wedge \exists a, b\ (E(a,b) \approx 0 \wedge \pi_2(a,b) \approx 1)).\end{aligned}$$

We say that we use a "variant" of the algorithm, because we do not need to perform Step 5 of the algorithm, in which the result of the quantifier elimination is negated. Therefore, we use SEH-PILoT with specification type HPILOT and specification theory REAL_CLOSED_FIELDS in mode SYMBOL_ELIMINATION.

We obtain the formula $(\pi_1(a,b) \approx 1 \wedge \pi_2(a,b) \approx 0) \vee (\pi_1(a,b) \approx 0 \wedge \pi_2(a,b) \approx 1)$. ■

Remark 6.6. *Eliminating the existentially quantified predicate symbol E can be done using Algorithm 2 for symbol elimination in this simple case. However, we do not claim that this works in general. As a more general approach one can also use a version of ordered resolution (which is implemented in the system SCAN [47]) for second-order quantifier elimination (cf. [84]).*

Example 6.7. Let the classes $\mathbf{C}_1^{\rightarrow}$ and $\mathbf{C}_2^{\rightarrow}$ be described by axioms of the following form:

$$\mathsf{Ax}_{\mathbf{C}_i^{\rightarrow}} :\ \forall u, v\ E(u,v) \rightarrow \pi_i(u,v) \qquad i = 1, 2$$

$\mathbf{C}_1^{\rightarrow} \subseteq \mathbf{C}_2^{\rightarrow}$ if and only if $\mathsf{Ax}_{\mathbf{C}_1^{\rightarrow}} \wedge \neg\mathsf{Ax}_{\mathbf{C}_2^{\rightarrow}}$ is unsatisfiable w.r.t. $\mathcal{T}$. We have:

$$\mathsf{Ax}_{\mathbf{C}_1^{\rightarrow}} \wedge \neg\mathsf{Ax}_{\mathbf{C}_2^{\rightarrow}} \equiv \forall u, v\ (E(u,v) \rightarrow \pi_1(u,v)) \wedge \exists a, b\ (E(a,b) \wedge \neg\pi_2(a,b))$$

Consider the second-order formula $\exists E\ (\mathsf{Ax}_{\mathbf{C}_1^{\rightarrow}} \wedge \neg\mathsf{Ax}_{\mathbf{C}_2^{\rightarrow}})$. This formula is equivalent to the formula $\exists a, b\ (\pi_1(a,b) \wedge \neg\pi_2(a,b))$ which does not contain the existentially quantified predicate E:

$$\begin{aligned}&\exists E\ (\forall u, v\ (E(u,v) \rightarrow \pi_1(u,v)) \wedge \exists a, b\ (E(a,b) \wedge \neg\pi_2(a,b)) \\ \equiv\ &\exists a, b\ (\pi_1(a,b) \wedge \neg\pi_2(a,b))\end{aligned}$$

Proof: For better readability, in what follows we express the predicates E, π_1 and π_2 as binary functions with codomain $\{0,1\}$ (where value 0 means the predicate is false and value 1 means it is true). We prove by hand that the equivalence above holds, i.e. we show that for any structure $\mathcal{A}$ over the signature $\Pi = (\{\pi_1, \pi_2\}, \emptyset)$ we have $\mathcal{A} \models F$ if and only if $\mathcal{A} \models G$, where:

$$\begin{aligned}F =\ &\exists E\ (\forall u, v\ (E(u,v) \approx 1 \rightarrow \pi_1(u,v) \approx 1) \wedge \exists a, b\ (E(a,b) \approx 1 \wedge \pi_2(a,b) \approx 0)) \\ G =\ &\exists a, b\ (\pi_1(a,b) \approx 1 \wedge \pi_2(a,b) \approx 0)\end{aligned}$$

1. ("⇒"): We show that $\mathcal{A} \models F$ implies $\mathcal{A} \models G$.

Assume $\mathcal{A} \models F$. Then there exists a structure $\mathcal{B}$ with the same universe and the same interpretations for π_1 and π_2 as $\mathcal{A}$ and, in addition, an interpretation $E_\mathcal{B}$ for E such that

(1) $\mathcal{B} \models \forall u, v\ (E(u, v) \approx 1 \rightarrow \pi_1(u, v) \approx 1)$ and

(2) $\mathcal{B} \models \exists a, b\ (E(a, b) \approx 1 \wedge \pi_2(a, b) \approx 0)$.

By (1) we know that whenever $E(a, b) \approx 1$ in $E_\mathcal{B}$, then $\pi_1(a, b) \approx 1$ in $\pi_{1\mathcal{B}}$. It then follows from (2) that $\mathcal{B} \models \exists a, b\ (\pi_1(a, b) \approx 1 \wedge \pi_2(a, b) \approx 0)$.

Since all the predicate symbols except for E are interpreted the same in $\mathcal{A}$ and $\mathcal{B}$, we then also have $\mathcal{A} \models \exists a, b\ (\pi_1(a, b) \approx 1 \wedge \pi_2(a, b) \approx 0)$.

Thus, $\mathcal{A} \models G$, so $\mathcal{A} \models F$ implies $\mathcal{A} \models G$.

2. ("⇐"): We show that $\mathcal{A} \models G$ implies $\mathcal{A} \models F$.

Assume $\mathcal{A} \models G$. We define a new structure $\mathcal{B}$ which has the same universe and the same interpretations for π_1 and π_2 as $\mathcal{A}$ and, in addition, an interpretation $E_\mathcal{B}$ with $E_\mathcal{B} = \pi_{1\mathcal{A}}$. Then the following hold:

(1) $\mathcal{B} \models \forall u, v\ (E(u, v) \approx 1 \rightarrow \pi_1(u, v) \approx 1)$ and

(2) $\mathcal{B} \models \exists a, b\ (E(a, b) \approx 1 \wedge \pi_2(a, b) \approx 0)$.

Then, since $\mathcal{A}$ is interpreted the same as $\mathcal{B}$ on all predicate symbols except for E and there exists an interpretation $E_\mathcal{B}$ with (1) and (2), we have

$$\mathcal{A} \models \exists E\ (\forall u, v\ (E(u, v) \approx 1 \rightarrow \pi_1(u, v) \approx 1) \wedge \exists a, b\ (E(a, b) \approx 1 \wedge \pi_2(a, b) \approx 0)).$$

Thus, $\mathcal{A} \models F$, so $\mathcal{A} \models G$ implies $\mathcal{A} \models F$. □

We have shown that $\mathbf{C}_1^{\rightarrow} \subseteq \mathbf{C}_2^{\rightarrow}$ if and only if $\pi_1(a, b) \wedge \neg\pi_2(a, b)$ is unsatisfiable w.r.t. $\mathcal{T}$, where a and b are Skolem constants introduced for the existentially quantified variables of the same name.

We can obtain this result also using SEH-PILoT. We consider all predicates to be functions with codomain $\{0, 1\}$ and eliminate E in the formula

$$\exists E\ (\forall u, v\ (E(u, v) \approx 1 \rightarrow \pi_1(u, v) \approx 1) \wedge \exists a, b\ (E(a, b) \approx 1 \wedge \pi_2(a, b) \approx 0)).$$

For this we use again SEH-PILoT with specification type HPILOT and specification theory REAL_CLOSED_FIELDS in mode SYMBOL_ELIMINATION.

We obtain the formula $\pi_1(a, b) \approx 1 \wedge \pi_2(a, b) \approx 0$. ■

Example 6.8. Let $\mathbf{C}$ be a class of graphs described by axiom $\mathsf{Ax}_\mathbf{C}$ and $\mathbf{P}$ the class of plane drawings described by axiom $\mathsf{Ax}_\mathbf{P}$, where:

$$\begin{array}{lll} \mathsf{Ax}_\mathbf{C}: & \forall u, v & E(u, v) \leftrightarrow \pi_\mathbf{C}(u, v) \\ \mathsf{Ax}_\mathbf{P}: & \forall u, v, w, x & E(w, x) \wedge \pi_\mathbf{P}(u, v, w, x) \rightarrow \neg E(u, v) \end{array}$$

$\mathbf{C} \subseteq \mathbf{P}$ if and only if $\mathsf{Ax}_\mathbf{C} \wedge \neg\mathsf{Ax}_\mathbf{P}$ is unsatisfiable w.r.t. $\mathcal{T}$. We have:

$$\mathsf{Ax}_\mathbf{C} \wedge \neg\mathsf{Ax}_\mathbf{P} \equiv \forall u, v\ (E(u, v) \leftrightarrow \pi_\mathbf{C}(u, v)) \wedge \exists a, b, c, d\ (E(c, d) \wedge \pi_\mathbf{P}(a, b, c, d) \wedge E(a, b))$$

Consider the second-order formula $\exists E\ (\mathsf{Ax}_{\mathbf{C}} \wedge \neg \mathsf{Ax}_{\mathbf{P}})$. It is equivalent to the formula $\exists a, b, c, d\ (\pi_{\mathbf{P}}(a, b, c, d) \wedge \pi_{\mathbf{C}}(c, d) \wedge \pi_{\mathbf{C}}(a, b))$ which does not contain the existentially quantified predicate E:

$$\begin{aligned} & \exists E\ (\forall u, v\ (E(u, v) \leftrightarrow \pi_{\mathbf{C}}(u, v)) \wedge \exists a, b, c, d\ (E(c, d) \wedge \pi_{\mathbf{P}}(a, b, c, d) \wedge E(a, b))) \\ \equiv\ & \exists a, b, c, d\ (\pi_{\mathbf{P}}(a, b, c, d) \wedge \pi_{\mathbf{C}}(c, d) \wedge \pi_{\mathbf{C}}(a, b)) \end{aligned}$$

Proof: For better readability, in what follows we express the predicates E, $\pi_{\mathbf{C}}$ and $\pi_{\mathbf{P}}$ as binary functions with codomain $\{0, 1\}$ (where value 0 means the predicate is false and value 1 means it is true). We prove by hand that the equivalence above holds, i.e. we show that for any structure $\mathcal{A}$ over the signature $\Pi = (\{\pi_{\mathbf{C}}, \pi_{\mathbf{P}}\}, \emptyset)$ we have $\mathcal{A} \models F$ if and only if $\mathcal{A} \models G$, where:

$$\begin{aligned} F = \ & \exists E\ (\forall u, v\ (E(u, v) \approx 1 \leftrightarrow \pi_{\mathbf{C}}(u, v) \approx 1) \wedge \\ & \quad \exists a, b, c, d\ (E(c, d) \approx 1 \wedge \pi_{\mathbf{P}}(a, b, c, d) \approx 1 \wedge E(a, b) \approx 1)) \\ G = \ & \exists a, b, c, d\ (\pi_{\mathbf{P}}(a, b, c, d) \approx 1 \wedge \pi_{\mathbf{C}}(c, d) \approx 1 \wedge \pi_{\mathbf{C}}(a, b) \approx 1) \end{aligned}$$

1. ("⇒"): We show that $\mathcal{A} \models F$ implies $\mathcal{A} \models G$.

Assume $\mathcal{A} \models F$. Then there exists a structure $\mathcal{B}$ with the same universe and the same interpretations for $\pi_{\mathbf{C}}$ and $\pi_{\mathbf{P}}$ as $\mathcal{A}$ and, in addition, an interpretation $E_{\mathcal{B}}$ for E such that

(1) $\mathcal{B} \models \forall u, v\ (E(u, v) \approx 1 \leftrightarrow \pi_{\mathbf{C}}(u, v) \approx 1)$ and

(2) $\mathcal{B} \models \exists a, b, c, d\ (E(c, d) \approx 1 \wedge \pi_{\mathbf{P}}(a, b, c, d) \approx 1 \wedge E(a, b) \approx 1)$.

As a consequence of (1) we then have $E_{\mathcal{B}} = \pi_{\mathbf{C}\mathcal{B}}$. It then follows from (2) that $\mathcal{B} \models \exists a, b, c, d\ (\pi_{\mathbf{C}}(c, d) \approx 1 \wedge \pi_{\mathbf{P}}(a, b, c, d) \approx 1 \wedge \pi_{\mathbf{C}}(a, b) \approx 1)$.

Since all the predicate symbols except for E are interpreted the same in $\mathcal{A}$ and $\mathcal{B}$, we then also have $\mathcal{A} \models \exists a, b, c, d\ (\pi_{\mathbf{C}}(c, d) \approx 1 \wedge \pi_{\mathbf{P}}(a, b, c, d) \approx 1 \wedge \pi_{\mathbf{C}}(a, b) \approx 1)$.

Thus, $\mathcal{A} \models G$, so $\mathcal{A} \models F$ implies $\mathcal{A} \models G$.

2. ("⇐"): We show that $\mathcal{A} \models G$ implies $\mathcal{A} \models F$.

Assume $\mathcal{A} \models G$. We define a new structure $\mathcal{B}$ which has the same universe and the same interpretations for $\pi_{\mathbf{C}}$ and $\pi_{\mathbf{P}}$ as $\mathcal{A}$ and, in addition, an interpretation $E_{\mathcal{B}}$ with $E_{\mathcal{B}} = \pi_{\mathbf{C}\mathcal{A}}$. Then the following hold:

(1) $\mathcal{B} \models \forall u, v\ (E(u, v) \approx 1 \leftrightarrow \pi_{\mathbf{C}}(u, v) \approx 1)$ and

(2) $\mathcal{B} \models \exists a, b, c, d\ (E(c, d) \approx 1 \wedge \pi_{\mathbf{P}}(a, b, c, d) \approx 1 \wedge E(a, b) \approx 1)$.

Then, since $\mathcal{A}$ is interpreted the same as $\mathcal{B}$ on all predicate symbols except for E and there exists an interpretation $E_{\mathcal{B}}$ with (1) and (2), we have

$$\begin{aligned} \mathcal{A} \models \ & \exists E\ (\forall u, v\ (E(u, v) \approx 1 \leftrightarrow \pi_{\mathbf{C}}(u, v) \approx 1) \wedge \\ & \quad \exists a, b, c, d\ (E(c, d) \approx 1 \wedge \pi_{\mathbf{P}}(a, b, c, d) \approx 1 \wedge E(a, b) \approx 1)). \end{aligned}$$

Thus, $\mathcal{A} \models F$, so $\mathcal{A} \models G$ implies $\mathcal{A} \models F$. □

We have shown that $\mathbf{C} \subseteq \mathbf{P}$ if and only if $\pi_{\mathbf{P}}(a, b, c, d) \wedge \pi_{\mathbf{C}}(c, d) \wedge \pi_{\mathbf{C}}(a, b)$ is unsatisfiable w.r.t. $\mathcal{T}$, where a, b, c and d are Skolem constants introduced for the existentially quantified variables of the same name.

We can obtain this result also using SEH-PILoT. We consider all predicates to be functions with codomain $\{0,1\}$ and eliminate E in the formula

$$\begin{aligned}&\exists E\ (\forall u,v\ (E(u,v)\approx 1 \leftrightarrow \pi_{\mathbf{C}}(u,v)\approx 1)\ \wedge\\ &\quad\exists a,b,c,d\ (E(c,d)\approx 1 \wedge \pi_{\mathbf{P}}(a,b,c,d)\approx 1 \wedge E(a,b)\approx 1)).\end{aligned}$$

For this we use again SEH-PILoT with specification type HPILOT and specification theory REAL_CLOSED_FIELDS in mode SYMBOL_ELIMINATION.

We obtain the formula $\pi_{\mathbf{P}}(a,b,c,d)\approx 1 \wedge \pi_{\mathbf{C}}(c,d)\approx 1 \wedge \pi_{\mathbf{C}}(a,b)\approx 1$. ■

Example 6.9. Let $\mathbf{C}^{\rightarrow}$ be a class of graphs described by axiom $\mathsf{Ax}_{\mathbf{C}^{\rightarrow}}$ and $\mathbf{P}$ the class of plane drawings described by axiom $\mathsf{Ax}_{\mathbf{P}}$, where:

$$\begin{aligned}&\mathsf{Ax}_{\mathbf{C}^{\rightarrow}}: && \forall u,v && E(u,v)\rightarrow \pi_{\mathbf{C}}(u,v)\\ &\mathsf{Ax}_{\mathbf{P}}: && \forall u,v,w,x && E(w,x)\wedge \pi_{\mathbf{P}}(u,v,w,x)\rightarrow \neg E(u,v)\end{aligned}$$

$\mathbf{C}^{\rightarrow}\subseteq\mathbf{P}$ if and only if $\mathsf{Ax}_{\mathbf{C}^{\rightarrow}}\wedge\neg\mathsf{Ax}_{\mathbf{P}}$ is unsatisfiable w.r.t. $\mathcal{T}$. We have:

$$\begin{aligned}\mathsf{Ax}_{\mathbf{C}^{\rightarrow}}\wedge\neg\mathsf{Ax}_{\mathbf{P}} \equiv\ & \forall u,v\ (E(u,v)\rightarrow \pi_{\mathbf{C}}(u,v))\ \wedge\\ & \exists a,b,c,d\ (E(c,d)\wedge \pi_{\mathbf{P}}(a,b,c,d)\wedge E(a,b))\end{aligned}$$

Consider the second-order formula $\exists E\ (\mathsf{Ax}_{\mathbf{C}^{\rightarrow}}\wedge\neg\mathsf{Ax}_{\mathbf{P}})$. This formula is equivalent to the formula $\exists a,b,c,d\ (\pi_{\mathbf{P}}(a,b,c,d)\approx 1 \wedge \pi_{\mathbf{C}}(c,d)\approx 1 \wedge \pi_{\mathbf{C}}(a,b)\approx 1)$ which does not contain the existentially quantified predicate E:

$$\begin{aligned}&\exists E\ (\forall u,v\ (E(u,v)\approx 1\rightarrow \pi_{\mathbf{C}}(u,v)\approx 1)\ \wedge\\ &\qquad\exists a,b,c,d\ (E(c,d)\approx 1 \wedge \pi_{\mathbf{P}}(a,b,c,d)\approx 1 \wedge E(a,b)\approx 1))\\ \equiv\ & \exists a,b,c,d\ (\pi_{\mathbf{P}}(a,b,c,d)\approx 1 \wedge \pi_{\mathbf{C}}(c,d)\approx 1 \wedge \pi_{\mathbf{C}}(a,b)\approx 1)\end{aligned}$$

Proof: For better readability, in what follows we express the predicates E, $\pi_{\mathbf{C}}$ and $\pi_{\mathbf{P}}$ as binary functions with codomain $\{0,1\}$ (where value 0 means the predicate is false and value 1 means it is true). We prove by hand that the equivalence above holds, i.e. we show that for any structure $\mathcal{A}$ over the signature $\Pi=(\{\pi_{\mathbf{C}},\pi_{\mathbf{P}}\},\emptyset)$ we have $\mathcal{A}\models F$ if and only if $\mathcal{A}\models G$, where:

$$\begin{aligned}F=\ & \exists E\ (\forall u,v\ (E(u,v)\approx 1\rightarrow \pi_{\mathbf{C}}(u,v)\approx 1)\ \wedge\\ &\qquad\exists a,b,c,d\ (E(c,d)\approx 1 \wedge \pi_{\mathbf{P}}(a,b,c,d)\approx 1 \wedge E(a,b)\approx 1))\\ G=\ & \exists a,b,c,d\ (\pi_{\mathbf{P}}(a,b,c,d)\approx 1 \wedge \pi_{\mathbf{C}}(c,d)\approx 1 \wedge \pi_{\mathbf{C}}(a,b)\approx 1)\end{aligned}$$

1. ("⇒"): We show that $\mathcal{A}\models F$ implies $\mathcal{A}\models G$.

Assume $\mathcal{A}\models F$. Then there exists a structure $\mathcal{B}$ with the same universe and the same interpretations for $\pi_{\mathbf{C}}$ and $\pi_{\mathbf{P}}$ as $\mathcal{A}$ and, in addition, an interpretation $E_{\mathcal{B}}$ for E such that

(1) $\mathcal{B}\models\forall u,v\ (E(u,v)\approx 1\rightarrow \pi_{\mathbf{C}}(u,v)\approx 1)$ and

(2) $\mathcal{B}\models\exists a,b,c,d\ (E(c,d)\approx 1 \wedge \pi_{\mathbf{P}}(a,b,c,d)\approx 1 \wedge E(a,b)\approx 1)$.

By (1) we know that whenever $E(a,b)\approx 1$ in $E_{\mathcal{B}}$, then $\pi_{\mathbf{C}}(a,b)\approx 1$ in $\pi_{\mathbf{C}\mathcal{B}}$. It then follows from (2) that $\mathcal{B}\models\exists a,b,c,d\ (\pi_{\mathbf{C}}(c,d)\approx 1 \wedge \pi_{\mathbf{P}}(a,b,c,d)\approx 1 \wedge \pi_{\mathbf{C}}(a,b)\approx 1)$.

Since all the predicate symbols except for E are interpreted the same in $\mathcal{A}$ and $\mathcal{B}$, we then also have $\mathcal{A}\models\exists a,b,c,d\ (\pi_{\mathbf{C}}(c,d)\approx 1 \wedge \pi_{\mathbf{P}}(a,b,c,d)\approx 1 \wedge \pi_{\mathbf{C}}(a,b)\approx 1)$.

Thus, $\mathcal{A}\models G$, so $\mathcal{A}\models F$ implies $\mathcal{A}\models G$.

2. ("⇐"): We show that $\mathcal{A} \models G$ implies $\mathcal{A} \models F$.

Assume $\mathcal{A} \models G$. We define a new structure $\mathcal{B}$ which has the same universe and the same interpretations for $\pi_{\mathbf{C}}$ and $\pi_{\mathbf{P}}$ as $\mathcal{A}$ and, in addition, an interpretation $E_{\mathcal{B}}$ with $E_{\mathcal{B}} = \pi_{\mathbf{C}\mathcal{A}}$. Then the following hold:

(1) $\mathcal{B} \models \forall u, v\ (E(u, v) \approx 1 \to \pi_{\mathbf{C}}(u, v) \approx 1)$ and

(2) $\mathcal{B} \models \exists a, b, c, d\ (E(c, d) \approx 1 \wedge \pi_{\mathbf{P}}(a, b, c, d) \approx 1 \wedge E(a, b) \approx 1)$.

Then, since $\mathcal{A}$ is interpreted the same as $\mathcal{B}$ on all predicate symbols except for E and there exists an interpretation $E_{\mathcal{B}}$ with (1) and (2), we have

$$\begin{aligned}\mathcal{A} \models \quad & \exists E\ (\forall u, v\ (E(u, v) \approx 1 \to \pi_{\mathbf{C}}(u, v) \approx 1) \wedge \\ & \quad \exists a, b, c, d\ (E(c, d) \approx 1 \wedge \pi_{\mathbf{P}}(a, b, c, d) \approx 1 \wedge E(a, b) \approx 1)).\end{aligned}$$

Thus, $\mathcal{A} \models F$, so $\mathcal{A} \models G$ implies $\mathcal{A} \models F$. □

We have shown that $\mathbf{C}^{\to} \subseteq \mathbf{P}$ if and only if $\pi_{\mathbf{P}}(a, b, c, d) \wedge \pi_{\mathbf{C}}(c, d) \wedge \pi_{\mathbf{C}}(a, b)$ is unsatisfiable w.r.t. $\mathcal{T}$, where a, b, c and d are Skolem constants introduced for the existentially quantified variables of the same name. This is the same condition for graph inclusion that we obtained in Example 6.8, so it follows that $\mathbf{C}^{\to} \subseteq \mathbf{P}$ if and only if $\mathbf{C} \subseteq \mathbf{P}$.

We can obtain this result also using SEH-PILoT. We consider all predicates to be functions with codomain $\{0, 1\}$ and eliminate E in the formula

$$\begin{aligned}& \exists E\ (\forall u, v\ (E(u, v) \approx 1 \to \pi_{\mathbf{C}}(u, v) \approx 1) \wedge \\ & \quad \exists a, b, c, d\ (E(c, d) \approx 1 \wedge \pi_{\mathbf{P}}(a, b, c, d) \approx 1 \wedge E(a, b) \approx 1)).\end{aligned}$$

For this we use again SEH-PILoT with specification type HPILOT and specification theory REAL_CLOSED_FIELDS in mode SYMBOL_ELIMINATION.

We obtain the formula $\pi_{\mathbf{P}}(a, b, c, d) \approx 1 \wedge \pi_{\mathbf{C}}(c, d) \approx 1 \wedge \pi_{\mathbf{C}}(a, b) \approx 1$. ■

For better reference, we show an overview of the conditions obtained for the different types of class inclusions in Figure 6.7.

Graph class axiomatizations:

$\mathsf{Ax}_{\mathbf{C}_i}$:	$\forall u, v$	$E(u, v) \leftrightarrow \pi_i(u, v)$	$i = 1, 2$
$\mathsf{Ax}_{\mathbf{C}_i^{\to}}$:	$\forall u, v$	$E(u, v) \to \pi_i(u, v)$	$i = 1, 2$
$\mathsf{Ax}_{\mathbf{P}}$:	$\forall u, v, w, s$	$E(w, s) \wedge \pi_{\mathbf{P}}(u, v, w, s) \to \neg E(u, v)$	

Conditions for graph class inclusion:

$\mathbf{C}_1 \subseteq \mathbf{C}_2$	iff	$(\pi_1(a, b) \wedge \neg\pi_2(a, b)) \vee (\neg\pi_1(a, b) \wedge \pi_2(a, b))$ is unsatisfiable
$\mathbf{C}_1^{\to} \subseteq \mathbf{C}_2^{\to}$	iff	$\pi_1(a, b) \wedge \neg\pi_2(a, b)$ is unsatisfiable
$\mathbf{C}_1 \subseteq \mathbf{P}$	iff	$\pi_{\mathbf{P}}(a, b, c, d) \wedge \pi_1(c, d) \wedge \pi_1(a, b)$ is unsatisfiable
$\mathbf{C}_1^{\to} \subseteq \mathbf{P}$	iff	$\mathbf{C}_1 \subseteq \mathbf{P}$

Figure 6.7: Overview of conditions for graph class inclusion.

Hierarchical Reasoning

Having obtained conditions for class inclusion for certain graph classes we can now in a second step check the concrete proof tasks stated in Section 6.2.1. We start with proof tasks (1) and (2).

Example 6.10. We analyze the relation between the classes $\mathbf{G}^{\rightarrow}$ and $\mathbf{R}^{\rightarrow}$. According to Figure 6.7, $\mathbf{G}^{\rightarrow} \subseteq \mathbf{R}^{\rightarrow}$ if and only if $\pi_{\mathbf{G}}(u,v) \wedge \neg\pi_{\mathbf{R}}(u,v)$ is unsatisfiable w.r.t. $\mathcal{T}$, and $\mathbf{R}^{\rightarrow} \subseteq \mathbf{G}^{\rightarrow}$ if and only if $\pi_{\mathbf{R}}(u,v) \wedge \neg\pi_{\mathbf{G}}(u,v)$ is unsatisfiable w.r.t. $\mathcal{T}$. The predicates $\pi_{\mathbf{G}}$ and $\pi_{\mathbf{R}}$ are defined (for an arbitrary distance function d) as follows:

$$\begin{aligned}\pi_{\mathbf{G}}(u,v) \coloneqq\ & V(u) \wedge V(v) \wedge u \not\approx v \wedge \\ & \forall w \left(w \not\approx u \wedge w \not\approx v \wedge V(w) \rightarrow d(m(u,v),w) > d(m(u,v),u)\right)\end{aligned}$$

$$\pi_{\mathbf{R}}(u,v) \coloneqq\ V(u) \wedge V(v) \wedge u \not\approx v \wedge \forall w\,(V(w) \rightarrow d(u,w) \geq d(u,v) \vee d(w,v) \geq d(u,v))$$

Here, $m(u,v)$ is the middle point of the segment uv. If u and v are considered to be constants, then both the extension of $\mathcal{T}$ with a function V satisfying condition $\pi_{\mathbf{G}}(u,v)$ and the extension of $\mathcal{T}$ with a function V satisfying condition $\pi_{\mathbf{R}}(u,v)$ can be proved to be local (embeddability, hence also locality, holds, as it is easy to construct a total model from a partial model by setting all the terms V(x) which are undefined to false). Therefore, we can use H-PILoT for the proof tasks.

We first consider the case where d is the Euclidean distance. Then we analyze this problem for the more general case where d is an arbitrary metric.

Case 1: d is the Euclidean distance.

We check both class inclusions.

(i) To prove that $\mathbf{R}^{\rightarrow} \subseteq \mathbf{G}^{\rightarrow}$ holds, it is sufficient to show that $\pi_{\mathbf{R}}(u,v) \wedge \neg\pi_{\mathbf{G}}(u,v)$ is unsatisfiable. We have the following formula:

$$\begin{aligned}& V(u) \wedge V(v) \wedge u \not\approx v \wedge \forall s(V(s) \rightarrow d(u,s) \geq d(u,v) \vee d(s,v) \geq d(u,v)) \wedge \\ & w \not\approx u \wedge w \not\approx v \wedge V(w) \wedge d(m,w) \leq d(m,u)\end{aligned}$$

The formula obtained after Skolemization and simplification from $\neg\pi_{\mathbf{G}}(u,v)$ is a ground formula. Due to the locality property described above, for the universal clause in $\pi_{\mathbf{R}}(u,v)$ it is sufficient to analyze all instances where s is replaced with u, v, w and m. We use functions $x(p)$ and $y(p)$ to describe the coordinates of a point p. The distance $d(p,q)$ between two points p and q is then defined as follows:

$$d(p,q) \approx \sqrt{(x(q) - x(p))^2 + (y(q) - y(p))^2}$$

Since provers may not be able to handle square root expressions, we can use a trick to avoid them. We only have atoms of the form $d(u,v) \rhd d(x,w)$ with $\rhd \in \{\approx, \not\approx, <, >, \leq, \geq\}$ (we do not have sums of distances for instance), so we can use the squares of the distances, as $d(u,v) \rhd d(x,w)$ if and only if $d(u,v)^2 \rhd d(x,w)^2$ for $\rhd \in \{\approx, \not\approx, <, >, \leq, \geq\}$.

The coordinates of the middle m of uv can be computed as usual:

$$x(m) \approx \frac{x(u) + x(v)}{2} \qquad\qquad y(m) \approx \frac{y(u) + y(v)}{2}$$

Stating that two points p and q are distinct can be done using the following condition:

$$x(p) \not\approx x(q) \vee y(p) \not\approx y(q)$$

We check the satisfiability of the obtained formula using H-PILoT with Z3 as an external prover. We derive unsatisfiability, which proves that $\mathbf{R}^{\rightarrow} \subseteq \mathbf{G}^{\rightarrow}$.

(ii) For proving that $\mathbf{G}^{\rightarrow} \not\subseteq \mathbf{R}^{\rightarrow}$ we show that $\pi_{\mathbf{G}}(u,v) \wedge \neg\pi_{\mathbf{R}}(u,v)$ is satisfiable. H-PILoT with external prover Z3 gives the answer "satisfiable" and produces the following model:

$$u \approx (1,0) \qquad w \approx (-1,-5)$$
$$v \approx (7,-8) \qquad m \approx (4,-4)$$

As can be seen in Figure 6.8, geometrically this describes a situation in which a vertex w lies outside of the smallest circle passing through u and v (green circle), but inside the intersecting area of the open disk with center u and radius $|uv|$ and the open disk with center v and radius $|uv|$ (red circles). This is the counterexample shown in Section 6.2.1.

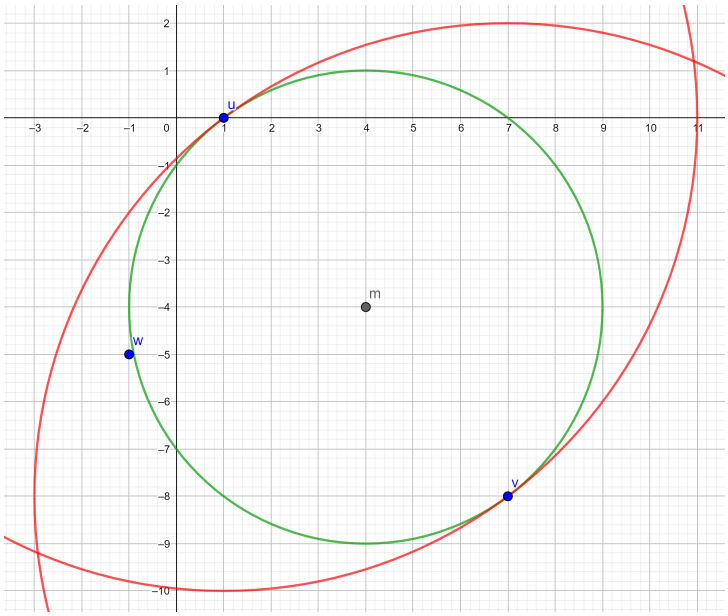

Figure 6.8: Counterexample for $\mathbf{G}^{\rightarrow} \subseteq \mathbf{R}^{\rightarrow}$ (Euclidean distance).

Case 2: d is an arbitrary metric.

We again check for both directions of inclusion whether the corresponding formula holds. For d we have the metric axioms, proved to be Ψ-local in Section 6.1.

For the direction $\pi_{\mathbf{R}}(u,v) \rightarrow \pi_{\mathbf{G}}(u,v)$ H-PILoT with external prover Z3 answers "satisfiable" and returns the following model:

$$d(u,v) \approx 6 \qquad d(u,m) \approx 3$$
$$d(v,m) \approx 3 \qquad d(u,w) \approx 6$$
$$d(v,w) \approx 5 \qquad d(w,m) \approx 3$$

Looking at Figure 6.9 it can be seen that this describes a situation which is not possible in Euclidean space: $d(u,w) \approx 6$ means that w lies on the blue circle and $d(v,w) \approx 5$ means that w lies on the red circle, but then $d(w,m) \approx 3$ is not possible, since this would mean that w lies in addition on the green circle. However, in an arbitrary metric space this is a valid counterexample. Thus, the inclusion $\mathbf{R}^{\rightarrow} \subseteq \mathbf{G}^{\rightarrow}$ holds in Euclidean space, but not in arbitrary metric spaces.

For the direction $\pi_{\mathbf{G}}(u,v) \to \pi_{\mathbf{R}}(u,v)$ H-PILoT also returns the answer "satisfiable" and the following model:

$$\begin{aligned} d(u,v) &\approx 6 & d(u,m) &\approx 3 \\ d(v,m) &\approx 3 & d(u,w) &\approx 1 \\ d(v,w) &\approx 5 & d(w,m) &\approx 4 \end{aligned}$$

This model again describes a situation which cannot occur in Euclidean space, as can be seen in Figure 6.10: $d(u,v) \approx 6$ together with $d(u,w) \approx 1$ and $d(v,w) \approx 5$ implies that w lies on the line segment uv, but then it cannot be that $d(w,m) \approx 4$.

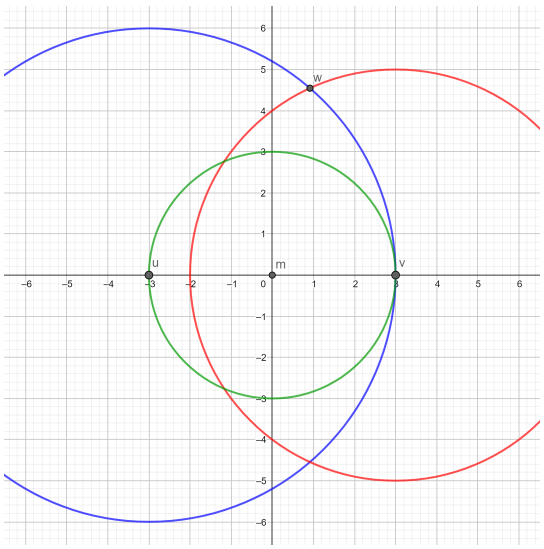

Figure 6.9: Counterexample for $\mathbf{R}^{\to} \subseteq \mathbf{G}^{\to}$ (arbitrary metric).

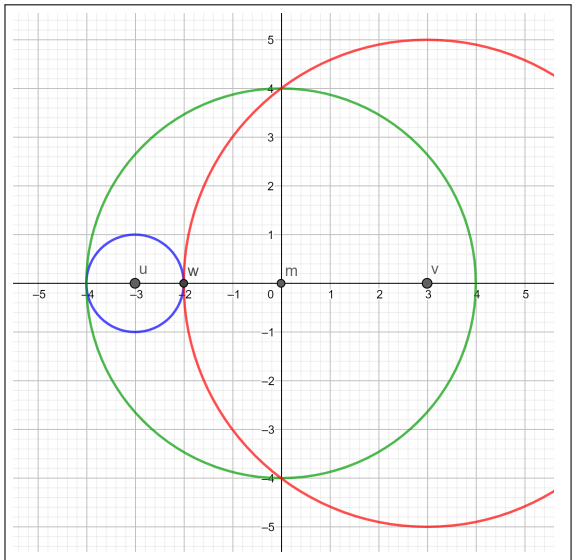

Figure 6.10: Counterexample for $\mathbf{G}^{\to} \subseteq \mathbf{R}^{\to}$ (arbitrary metric).

We proved that if we are in Euclidean space, the class $\mathbf{R}^{\to}$ is contained in $\mathbf{G}^{\to}$. Since $\mathbf{R}^{\to}$ is a superclass of $\mathbf{R}$, it follows that $\mathbf{R} \subseteq \mathbf{G}^{\to}$, i.e. every relative neighborhood graph is a spanning subgraph of a Gabriel graph. In an arbitrary metric space the inclusion $\mathbf{R}^{\to} \subseteq \mathbf{G}^{\to}$ does not hold. It was also shown that the converse direction of inclusion holds neither in Euclidean space nor in arbitrary metric spaces. ■

Example 6.11. We want to show that Gabriel graphs and relative neighborhood graphs are always plane drawings, i.e. that the inclusions $\mathbf{G} \subseteq \mathbf{P}$ and $\mathbf{R} \subseteq \mathbf{P}$ hold. For this we first prove that $\mathbf{G}^{\to} \subseteq \mathbf{P}$.

According to Figure 6.7, $\mathbf{G}^{\to} \subseteq \mathbf{P}$ holds if and only if $\pi_{\mathbf{P}}(u,v,w,s) \wedge \pi_{\mathbf{G}}(w,s) \wedge \pi_{\mathbf{G}}(u,v)$ is unsatisfiable. We first check this using the encoding of d as the Euclidean metric.

Case 1: d is the Euclidean distance.

We can express that line segments uv and ws intersect by stating that there is an intersection point p such that

- u, v, p are collinear and w, s, p are collinear,
- p lies on the line segment uv, i.e. $d(u,p) \leq d(u,v)$ and $d(v,p) \leq d(u,v)$,
- p lies on the line segment ws, i.e. $d(w,p) \leq d(w,s)$ and $d(s,p) \leq d(w,s)$.

We assume that u, v, w and s are distinct vertices, because if any of them are equal we do not have a proper intersection. $\pi_{\mathbf{G}}$ can be expressed using the midpoints m_1 and m_2 of the segments uv and ws, respectively.

We use H-PILoT for checking satisfiability of the formula. The prover Z3 (and therefore also H-PILoT) does not terminate after 3 minutes. Next we try the encoding of d as an arbitrary metric.

Case 2: d is an arbitrary metric.

In this case we can express intersection in a simpler way, by stating that there is an intersection point p such that the following hold:

$$d(u,p) + d(p,v) \approx d(u,v)$$
$$d(w,p) + d(p,s) \approx d(w,s)$$

Note that this simpler formalization cannot be used for the Euclidean distance, because there we work with the squares of the distances, for which addition of distances is not the same, i.e. $d(u,p) + d(p,v) \approx d(u,v)$ does not imply $d(u,p)^2 + d(p,v)^2 \approx d(u,v)^2$.

We use H-PILoT to check satisfiability. We obtain "satisfiable" as an answer, so we let H-PILoT generate a model. In this model we have

$$\begin{aligned} d(w,m_2) &\approx 231, \\ d(w,p) &\approx 101, \\ d(m_2,p) &\approx 205, \end{aligned}$$

where m_2 is the midpoint of w and s. Since w, m_2 and p are supposed to be collinear, this situation is not possible in Euclidean space.

We can conclude that the class inclusion does not hold in general for arbitrary metrics. However, we can impose the following additional assumptions to rule out the obtained counterexample:

A1) $d(u,m_1) \approx d(u,p) + d(p,m_1)$ or $d(v,m_1) \approx d(v,p) + d(p,m_1)$

A2) $d(w,m_2) \approx d(w,p) + d(p,m_2)$ or $d(s,m_2) \approx d(s,p) + d(p,m_2)$

With these additional assumptions we check satisfiability using H-PILoT again. This time we get the answer "unsatisfiable".

We have therefore shown that $\mathbf{G}^{\rightarrow}$ is included in $\mathbf{P}$ under the additional assumptions $A1$ and $A2$. In Euclidean space $A1$ and $A2$ are direct consequences of the following conditions:

(1) $d(u,p) + d(p,v) \approx d(u,v)$

(2) $d(w,p) + d(p,s) \approx d(w,s)$

(3) $2 \cdot d(u,m_1) \approx d(u,v)$

(4) $2 \cdot d(v,m_1) \approx d(u,v)$

(5) $2 \cdot d(w,m_2) \approx d(w,s)$

(6) $2 \cdot d(s,m_2) \approx d(w,s)$

We can therefore conclude that in Euclidean space $\mathbf{G}^{\rightarrow} \subseteq \mathbf{P}$ holds. We can also conclude that in every metric space in which $(1) - (6)$ imply $A1$ and $A2$ the class inclusion holds.

Since we know by Example 6.10 that $\mathbf{R}^{\rightarrow} \subseteq \mathbf{G}^{\rightarrow}$ w.r.t. the Euclidean metric, we can conclude that also $\mathbf{R}^{\rightarrow} \subseteq \mathbf{P}$ w.r.t. the Euclidean metric. As according to Figure 6.7 it is known that $\mathbf{C}_1^{\rightarrow} \subseteq \mathbf{P}$ if and only if $\mathbf{C}_1 \subseteq \mathbf{P}$, it follows that $\mathbf{G} \subseteq \mathbf{P}$ and $\mathbf{R} \subseteq \mathbf{P}$ w.r.t. the Euclidean metric. ■

In Example 6.10 we proved that in Euclidean space every relative neighborhood graph is a spanning subgraph of a Gabriel graph (and the opposite is not true) and in Example 6.11 we proved that in Euclidean space the class $\mathbf{G}$ of Gabriel graphs and the class $\mathbf{R}$ of relative neighborhood graphs are subclasses of the class $\mathbf{P}$ of plane drawings. We have therefore successfully carried out all the proof tasks from Section 6.2.1.

6.3 Graph Classes Obtained by Transformations

We now consider graph classes $\mathbf{C}(\overline{p})$ (where $\overline{p}$ is a sequence of symbols denoting *parameters*) which can be described using *inclusion*, *exclusion* and *transfer* axioms. We assume given a set of vertices V.

The *inclusion axioms* specify which edges have to exist. For a graph class $\mathbf{C}$, the condition under which an edge $E(u, v)$ must exist can be described by a formula $\pi_C^i(u, v)$. Therefore, inclusion axioms have the following form:

(1) $\quad \forall u, v \;\; \pi_C^i(u, v) \rightarrow E(u, v)$

The *exclusion axioms* specify which edges are not allowed to exist. For a graph class $\mathbf{C}$, the condition under which an edge $E(u, v)$ is not allowed to exist can be described by a formula $\pi_C^e(u, v)$. Therefore, exclusion axioms have the following form:

(2) $\quad \forall u, v \;\; \pi_C^e(u, v) \rightarrow \neg E(u, v)$

The *transfer axioms* specify which edges $E(u, w)$ must exist as a consequence of the existence of another edge $E(u, v)$. For a graph class $\mathbf{C}$, we describe these conditions by a formula $\pi_C^t(u, v, w)$. Therefore, transfer axioms have the following form:

(3) $\quad \forall u, v, w \;\; \pi_C^t(u, v, w) \wedge E(u, w) \rightarrow E(u, v).$

We refer to the formulae π_C^i, π_C^e and π_C^t as inclusion, exclusion and transfer predicates, respectively. If it is clear from the context which class $\mathbf{C}$ these predicates are referring to, we often use the simplified notations π^i, π^e and π^t.

If the description of the graph class $\mathbf{C}$ depends on parameters $\overline{p}$, the formulae π_C^i, π_C^e and π_C^t might contain parameters. We will sometimes indicate this by adding the parameters to the arguments, i.e. writing $\pi_C^i(u, v, \overline{p}), \pi_C^e(u, v, \overline{p})$ and $\pi_C^t(u, v, w, \overline{p})$, respectively.

Example 6.12. We define the classes $\mathbf{MinDG}(r)$ of minimum disk graphs, $\mathbf{MaxDG}(r)$ of maximum disk graphs and $\mathbf{CRG}$ of connected region graphs (cf. [17]).

- $\mathsf{MinDG}(r)$: axiom (1), where $\pi^i(u, v, r)$ is the formula $u \not\approx v \wedge d(u, v) \leq r$.
- $\mathsf{MaxDG}(r)$: axiom (2), where $\pi^e(u, v, r)$ is the formula $d(u, v) > r$.
- CRG: axiom (3), where $\pi^t(u, v, w)$ is the formula $u \not\approx w \wedge d(u, v) \leq d(u, w)$.

In these axioms r is supposed to be a parameter. If r is a constant, for instance $r = 1$, then the third argument of π^i and π^e is not necessary and we write only $\pi^i(u, v)$ and $\pi^e(u, v)$. In what follows we will sometimes also use the notation $\pi^i_r(u, v)$ instead of $\pi^i(u, v, r)$.

The inclusion axiom $\mathsf{MinDG}(r)$ states that if $u \not\approx v$ and $d(u, v) \leq r$, then an edge from u to v must exist. Intuitively, this means that two vertices are always connected if they are closer to each other than a given radius r.

The exclusion axiom $\mathsf{MaxDG}(r)$ states that if $d(u, v) > r$, then we are not allowed to have an edge from u to v. This means that two nodes are never connected if they are farther apart from each other than a given radius r.

The transfer axiom CRG states that if u and w are different and there is an edge from u to v and $d(u, w) \leq d(u, v)$, then there must exist an edge also from u to w. The intuition behind it is that if a node is connected to some other node, then it must also be connected to any node that is closer or equally close to it.

By combining such axioms we obtain axiomatizations for new graph classes. If the classes $\mathbf{A}$ and $\mathbf{B}$ of graphs are axiomatized by axioms $\mathsf{Ax}_{\mathbf{A}}$ and $\mathsf{Ax}_{\mathbf{B}}$, then $\mathsf{Ax}_{\mathbf{A}} \wedge \mathsf{Ax}_{\mathbf{B}}$ is an axiomatization for the intersection $\mathbf{A} \cap \mathbf{B}$.

For instance, the class $\mathbf{UDG} = \mathbf{MinDG}(1) \cap \mathbf{MaxDG}(1)$ of unit disk graphs is axiomatized by $\mathsf{MinDG}(1) \wedge \mathsf{MaxDG}(1)$. This class describes all graphs in which any two nodes are connected if one node is contained in the closed unit disk (i.e. disk with radius 1) around the other node, i.e. their distance is smaller or equal to 1, and two nodes are not connected if this is not the case, i.e. their distance is greater than 1. ■

Transformations

From now on we consider directed graphs defined by inclusion, exclusion and transfer axioms. By applying transformations γ on graphs that transform the edges and leave the set of vertices unchanged we can define further graph classes $\gamma(\mathbf{C}) = \{\gamma(G) \mid G \in \mathbf{C}\}$. We consider two transformations on directed graphs which make them symmetric. A graph is called symmetric if $\forall x, y\ E(x, y) \leftrightarrow E(y, x)$ holds. A non-symmetric graph can be made symmetric by considering all edges $E(x, y)$ for which $E(y, x)$ does not exist and either

(a) deleting all edges $E(x, y)$, or

(b) adding all missing edges $E(y, x)$.

We can describe (a) and (b) formally by transformations $\cdot^-$ and $\cdot^+$, respectively. Given a graph $G = (V, E)$, we can build the symmetric subgraph $G^- = (V, E^-)$ and symmetric supergraph $G^+ = (V, E^+)$ using axioms $\mathsf{Tr}^-(E, E^-)$ and $\mathsf{Tr}^+(E, E^+)$ defined as follows:

$$\begin{aligned} \mathsf{Tr}^-(E, F): &\quad \forall x, y\ (F(x, y) \leftrightarrow (E(x, y) \wedge E(y, x))) \\ \mathsf{Tr}^+(E, F): &\quad \forall x, y\ (F(x, y) \leftrightarrow (E(x, y) \vee E(y, x))) \end{aligned}$$

We can thus define the classes $\mathbf{C}^- = \{G^- \mid G \in \mathbf{C}\}$ and $\mathbf{C}^+ = \{G^+ \mid G \in \mathbf{C}\}$.

In what follows, when talking about graphs obtained by transformation from other graphs, we will use the notation $E(x, y)$ to refer to edges in the original graph and $F(x, y)$ to refer to edges in the transformed graph, i.e. $F(x, y)$ is true if there is an edge between vertices x and y in the transformed graph and false if not.

Example 6.13. The class of quasi unit disk graphs [15, 74] can be described by

$$\mathbf{QUDG}(r) = (\mathbf{MinDG}(r) \cap \mathbf{MaxDG}(1))^{-},$$

where r is a parameter which specifies the maximum communication range $r(x) < 1$ of a vertex x.

We then have an axiomatization $\mathsf{Ax}_{\mathbf{QUDG}} = \exists E\big(\mathbf{MinDG}(r) \wedge \mathbf{MaxDG}(1) \wedge \mathsf{Tr}^{-}(E, F)\big)$.

In a quasi unit disk graph an edge from a vertex x to a vertex y must exist if y is within the maximum communication range $r(x)$ of vertex x. In Figure 6.11, as vertex v is within the communication range $r(u)$ of vertex u (blue circle), there is an edge between u and v.
Two vertices must not be connected if their distance is greater than 1. As shown in Figure 6.11, since t lies outside the unit disk of vertex u (green circle), there is no edge between u and t.
For any vertices located in between the communication range and the unit disk of a vertex x, an edge to x may or may not exist. This is the case for vertices w and s in Figure 6.11, of which only w is connected to u. ■

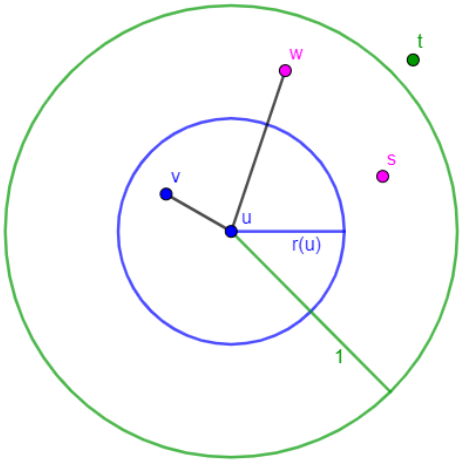

Figure 6.11: Depiction of the inclusion and exclusion conditions in a quasi unit disk graph.

Let $\mathbf{A}$ be a class of graphs described by axioms $\mathsf{Ax}_{\mathbf{A}}$ and $\mathbf{B}$ be a class of graphs described by axioms $\mathsf{Ax}_{\mathbf{B}}$. Let $\mathcal{T}$ be a theory used for expressing these axioms. Consider the transformations $\cdot^{+}$ and $\cdot^{-}$ described above. Then $\mathbf{A}^{+} \subseteq \mathbf{B}^{-}$ holds if and only if the following equivalent conditions are true:

- For every graph $H = (V, F) \in \mathbf{A}^{+}$ we have $H \in \mathbf{B}^{-}$.
- For every graph $H = (V, F)$, if there exists a graph $G_A = (V, E_A) \in \mathbf{A}$ with $H = G_A^{+}$, then there exists a graph $G_B = (V, E_B) \in \mathbf{B}$ with $H = G_B^{-}$.
- It holds that $\exists E_A\ (\mathsf{Ax}_{\mathbf{A}} \wedge \mathsf{Tr}^{+}(E_A, F)) \models_{\mathcal{T}} \exists E_B\ (\mathsf{Ax}_{\mathbf{B}} \wedge \mathsf{Tr}^{-}(E_B, F))$.

Let $\mathcal{T}$ be a theory with the signature $\Pi = (S, \Sigma, \mathsf{Pred})$ and let $\overline{P}_1 = P_1^1, \ldots, P_{n_1}^1$ and $\overline{P}_2 = P_1^2, \ldots, P_{n_2}^2$ be finite sequences of different predicate symbols with $P_j^i \notin \mathsf{Pred}$ and $\Pi_i = (\Sigma, \mathsf{Pred} \cup \{P_j^i \mid 1 \leq j \leq n_i\})$ for $i \in \{1, 2\}$. Let F_1 be a universal Π_1-formula and F_2 be a universal Π_2-formula. We analyze the problem of checking whether "$\exists \overline{P}_1\ F_1$ entails $\exists \overline{P}_2\ F_2$ w.r.t. $\mathcal{T}$" holds.

Assume that there exist Π-formulae G_1 and G_2 such that $G_1 \equiv_{\mathcal{T}} \exists \overline{P}_1 F_1$ and $G_2 \equiv_{\mathcal{T}} \exists \overline{P}_2 F_2$. Such formulae can be found for instance by saturation under a version of ordered resolution by successively eliminating $P_1^i, \ldots, P_{n_i}^i$ (cf. [84]). In this case, $\exists \overline{P}_1\ F_1 \models_{\mathcal{T}} \exists \overline{P}_2\ F_2$ if and only if $G_1 \models_{\mathcal{T}} G_2$, which is the case if and only if $G_1 \wedge \neg G_2 \models_{\mathcal{T}} \bot$. The problem of checking whether $G_1 \wedge \neg G_2 \models_{\mathcal{T}} \bot$ is in general undecidable, even if G_1 and G_2 are universal formulae and $\mathcal{T}$ is the extension of Presburger arithmetic or real arithmetic with a new function or predicate symbol (cf. [109]).

If $G_1 \wedge \neg G_2$ is in a fragment of $\mathcal{T}$ for which checking satisfiability is decidable, then we can effectively check whether $\exists \overline{P}_1\ F_1 \models_{\mathcal{T}} \exists \overline{P}_2\ F_2$. This is obviously the case if $\mathcal{T}$ is a decidable theory. We will show that a similar condition can be obtained for local extensions of theories allowing quantifier elimination if G_1 and G_2 are universal formulae and the extensions satisfy a certain "flatness property" which allows finite complete instantiation, and that in both cases we can also generate constraints on parameters under which entailment holds.

Theorem 6.14. *Assume that there exist* Π*-formulae* G_1 *and* G_2 *such that* $G_1 \equiv_{\mathcal{T}} \exists \overline{P}_1 F_1$ *and* $G_2 \equiv_{\mathcal{T}} \exists \overline{P}_2 F_2$.

(1) If $\mathcal{T}$ *is a decidable theory, then we can effectively check whether* $\exists \overline{P}_1\ F_1 \models_{\mathcal{T}} \exists \overline{P}_2\ F_2$.

(2) If $\mathcal{T}$ *allows quantifier elimination and the formulae* F_1 *and* F_2 *contain parametric constants, we can use quantifier elimination in* $\mathcal{T}$ *to derive conditions on these parameters under which* $\exists \overline{P}_1\ F_1 \models_{\mathcal{T}} \exists \overline{P}_2\ F_2$.

Proof: Since G_1 and G_2 are equivalent w.r.t. $\mathcal{T}$ to $\exists \overline{P}_1 F_1$ and $\exists \overline{P}_2 F_2$, repectively, checking whether $\exists \overline{P}_1\ F_1 \models_{\mathcal{T}} \exists \overline{P}_2\ F_2$ is equivalent to checking whether $G_1 \models_{\mathcal{T}} G_2$, which is the same as checking whether $G_1 \wedge \neg G_2 \models_{\mathcal{T}} \bot$.

For (1), note that for a decidable theory $\mathcal{T}$, checking $G_1 \wedge \neg G_2 \models_{\mathcal{T}} \bot$ is possible.

For (2), note that, assuming $\mathcal{T}$ allows quantifier elimination, we can use Algorithm 2 to compute a constraint Γ over the parameters such that $G_1 \wedge \Gamma \wedge \neg G_2 \models_{\mathcal{T}} \bot$. □

Theorem 6.15. *Assume that there exist two* universal Π*-formulae* G_1 *and* G_2 *such that* $G_1 \equiv_{\mathcal{T}} \exists \overline{P}_1 F_1$ *and* $G_2 \equiv_{\mathcal{T}} \exists \overline{P}_2 F_2$*, and that* $\mathcal{T} = \mathcal{T}_0 \cup \mathcal{K}$*, where* $\mathcal{T}_0$ *is a decidable theory with signature* $\Pi_0 = (S_0, \Sigma_0, \mathsf{Pred}_0)$*, where* S_0 *is a set of interpreted sorts and* $\mathcal{K}$ *is a set of (universally quantified) clauses over* $\Pi = (S_0 \cup S_1, \Sigma_0 \cup \Sigma_1, \mathsf{Pred}_0 \cup \mathsf{Pred}_1)$*, where*

(i) S_1 *is a new set of uninterpreted sorts,*

(ii) Σ_1 *and* Pred_1 *are sets of new function and predicate symbols, respectively, which have only arguments of uninterpreted sort* $\in S_1$*, and all function symbols in* Σ_1 *have interpreted output sort* $\in S_0$.

Assume in addition that all variables and constants of sort $\in S_1$ *in* $\mathcal{K}, G_1$ *and* $\neg G_2$ *occur below function symbols in* Σ_1*. Then:*

(1) We can use the decision procedure for $\mathcal{T}_0$ *to effectively check whether* $G_1 \wedge \neg G_2 \models_{\mathcal{T}} \bot$ *(hence whether* $\exists \overline{P}_1\ F_1 \models_{\mathcal{T}} \exists \overline{P}_2\ F_2$*).*

(2) If $\mathcal{T}_0$ *allows quantifier elimination and the formulae* F_1 *and* F_2 *(hence also* G_1 *and* G_2*) contain parametric constants and functions, we can use Algorithm 2 for obtaining constraints on the parameters under which* $\exists \overline{P}_1\ F_1 \models_{\mathcal{T}} \exists \overline{P}_2\ F_2$.

Proof: Let C be the set of constants of uninterpreted sort $s \in S_1$ occurring in $\mathcal{K}, G_1$ and $\neg G_2$. Note that $G_1 \wedge \neg G_2$ is satisfiable w.r.t. $\mathcal{T} = \mathcal{T}_0 \cup \mathcal{K}$ if and only if $(\mathcal{K} \wedge G_1)^{[C]} \wedge \neg G_2$ is satisfiable, where $(\mathcal{K} \wedge G_1)^{[C]}$ is the set of all instances of $\mathcal{K} \wedge G_1$ in which the variables of sort $s \in S_1$ are replaced with constants of sort s in C.

(1) The hierarchical reasoning method in Theorem 2.25 allows us to reduce testing whether $G_1 \wedge \neg G_2 \models_{\mathcal{T}} \bot$ to a satisfiability test w.r.t. $\mathcal{T}_0$.

(2) If $\mathcal{T}_0$ allows quantifier elimination, we can use Theorem 3.4. □

Example 6.16. Consider the following two classes:

$$A = \mathbf{QUDG}(r) = (\mathbf{MinDG}(r) \cap \mathbf{MaxDG}(1))^-$$
$$B = (\mathbf{MinDG}(r) \cap \mathbf{MaxDG}(1))^+$$

We are interested in checking whether (or under which conditions) $A = B$ holds, i.e. when the classes obtained by the transformations $.^+$ and $.^-$ from $\mathbf{MinDG}(r) \cap \mathbf{MaxDG}(1)$ are the same. For this we test whether $A \subseteq B$ and whether $B \subseteq A$. For such tasks we can use a two-layered approach:

(1) Using methods for general symbol elimination we want to eliminate E and obtain axiomatizations for the classes that depend only on the predicates $\pi_r^i(x, y)$, $\pi^e(x, y)$ and $F(x, y)$.

(2) Using the axiomatizations from (1) we can by Theorem 6.15 check satisfiability to test whether the inclusion holds, and if this is not the case we can by Theorem 6.15 use property-directed symbol elimination to derive conditions on the parameters which guarantee that the inclusion holds.

General Symbol Elimination

From the axiomatization $\mathsf{Ax}_A = \mathbf{MinDG}(r) \wedge \mathbf{MaxDG}(1) \wedge \mathsf{Tr}^-(E, F)$, which contains the predicate symbols π_r^i, π^e, E and F, we want to obtain an axiomatization G_A only containing π_r^i, π^e and F such that $G_A \equiv \exists E\ \mathbf{MinDG}(r) \wedge \mathbf{MaxDG}(1) \wedge \mathsf{Tr}^-(E, F)$. In what follows we describe how to use a variant of Algorithm 2 to eliminate the function symbol E.

Since we want to use SEH-PILoT for symbol elimination and SEH-PILoT cannot eliminate predicate symbols, we express all predicates as functions with codomain $\{0, 1\}$ (where value 0 means the predicate is false and value 1 means it is true). From the axiomatization $\mathsf{Ax}_A = \mathbf{MinDG}(r) \wedge \mathbf{MaxDG}(1) \wedge \mathsf{Tr}^-(E, F)$ we then get the following clauses:

$$\begin{array}{lll}
\forall x, y & \pi_r^i(x, y) \approx 1 \rightarrow E(x, y) \approx 1 & (A_1) \\
\forall x, y & \pi^e(x, y) \approx 1 \rightarrow E(x, y) \approx 0 & (A_2) \\
\forall x, y & F(x, y) \approx 1 \rightarrow E(x, y) \approx 1 & (A_3) \\
\forall x, y & F(x, y) \approx 1 \rightarrow E(y, x) \approx 1 & (A_4) \\
\forall x, y & E(x, y) \approx 1 \wedge E(y, x) \approx 1 \rightarrow F(x, y) \approx 1 & (A_5)
\end{array}$$

In addition we have the following clauses stating that the predicates are either true or false:

$$\begin{array}{ll}
\forall x, y & \pi_r^i(x, y) \approx 1 \vee \pi_r^i(x, y) \approx 0 \\
\forall x, y & \pi^e(x, y) \approx 1 \vee \pi^e(x, y) \approx 0 \\
\forall x, y & E(x, y) \approx 1 \vee E(x, y) \approx 0 \\
\forall x, y & F(x, y) \approx 1 \vee F(x, y) \approx 0
\end{array}$$

We can use SEH-PILoT to eliminate E from the conjunction of the clauses above. We enforce the instantiation of the clauses with exactly two variables a and b by adding in the query of the input file for SEH-PILoT the following trivial formulae:

$$\begin{array}{llll}
\pi_r^i(a, b) \approx \pi_r^i(a, b) & \pi^e(a, b) \approx \pi^e(a, b) & E(a, b) \approx E(a, b) & F(a, b) \approx F(a, b) \\
\pi_r^i(b, a) \approx \pi_r^i(b, a) & \pi^e(b, a) \approx \pi^e(b, a) & E(b, a) \approx E(b, a) & F(b, a) \approx F(b, a)
\end{array}$$

Since we perform general symbol elimination and not property-directed symbol elimination here, we do not need to negate the result of the quantifier elimination, i.e. we are interested in the formula obtained after Steps 1-4 of Algorithm 2. SEH-PILoT therefore has to be used with specification type HPILOT and specification theory REAL_CLOSED_FIELDS in mode SYMBOL_ELIMINATION. As SEH-PILoT does not simplify the result of the quantifier elimination very well, the obtained formula is quite large. However, by analyzing the formulae we are able to manually show that the result is equivalent to the following conjunction of clauses:

$$\begin{aligned}
&\forall x, y \quad \pi_r^i(x,y) \approx 1 \wedge \pi^e(x,y) \approx 1 \rightarrow \bot && (A_1') \\
&\forall x, y \quad \pi_r^i(x,y) \approx 1 \wedge \pi_r^i(y,x) \approx 1 \rightarrow F(y,x) \approx 1 && (A_2') \\
&\forall x, y \quad \pi^e(x,y) \approx 1 \rightarrow F(x,y) \approx 0 && (A_3') \\
&\forall x, y \quad \pi^e(x,y) \approx 1 \rightarrow F(y,x) \approx 0 && (A_4') \\
&\forall x, y \quad F(x,y) \approx 1 \rightarrow F(y,x) \approx 1 && (A_5')
\end{aligned}$$

We prove the correctness of the newly obtained axiomatization.

Proof: Let $F_A = \exists E\ (A_1 \wedge ... \wedge A_5)$ and $G_A = A_1' \wedge ... \wedge A_5'$. We show that for any structure $\mathcal{A}$ over the signature $\Pi = (\{\pi_r^i, \pi^e, F\}, \emptyset)$ we have $\mathcal{A} \models F_A$ if and only if $\mathcal{A} \models G_A$.

1. ("⇒"): We show that $\mathcal{A} \models F_A$ implies $\mathcal{A} \models G_A$.

Assume $\mathcal{A} \models F_A$. Then there exists a structure $\mathcal{B}$ with the same universe and the same interpretations for π^e, π_r^i and F as $\mathcal{A}$ and, in addition, an interpretation $E_\mathcal{B}$ for E such that

(1) $\mathcal{B} \models \forall x, y\ (\pi_r^i(x,y) \approx 1 \rightarrow E(x,y) \approx 1)$,

(2) $\mathcal{B} \models \forall x, y\ (\pi^e(x,y) \approx 1 \rightarrow E(x,y) \approx 0)$,

(3) $\mathcal{B} \models \forall x, y\ (F(x,y) \approx 1 \rightarrow E(x,y) \approx 1)$,

(4) $\mathcal{B} \models \forall x, y\ (F(x,y) \approx 1 \rightarrow E(y,x) \approx 1)$,

(5) $\mathcal{B} \models \forall x, y\ (E(x,y) \approx 1 \wedge E(y,x) \approx 1 \rightarrow F(x,y) \approx 1)$.

We can observe the following:

- From (1) and (2) it follows that $\mathcal{B} \models \forall x, y\ (\pi_r^i(x,y) \approx 1 \wedge \pi^e(x,y) \approx 1 \rightarrow \bot)$, i.e. $\mathcal{B} \models A_1'$.
- From (1) and (5) it follows that $\mathcal{B} \models \forall x, y\ (\pi_r^i(x,y) \approx 1 \wedge \pi_r^i(y,x) \approx 1 \rightarrow F(y,x) \approx 1)$, i.e. $\mathcal{B} \models A_2'$.
- From (2) and (3) it follows that $\mathcal{B} \models \forall x, y\ (\pi^e(x,y) \approx 1 \rightarrow F(x,y) \approx 0)$, i.e. $\mathcal{B} \models A_3'$.
- From (2) and (4) it follows that $\mathcal{B} \models \forall x, y\ (\pi^e(x,y) \approx 1 \rightarrow F(y,x) \approx 0)$, i.e. $\mathcal{B} \models A_4'$.
- From (3), (4) and (5) it follows that $\mathcal{B} \models \forall x, y\ (F(x,y) \approx 1 \rightarrow F(y,x) \approx 1)$, i.e. $\mathcal{B} \models A_5'$.

Since all the predicate symbols except for E are interpreted the same in $\mathcal{A}$ and $\mathcal{B}$, we then also have $\mathcal{A} \models A_i'$ for $i \in \{1, ..., 5\}$. Thus, $\mathcal{A} \models G_A$, so $\mathcal{A} \models F_A$ implies $\mathcal{A} \models G_A$.

2. ("⇐"): We show that $\mathcal{A} \models G_A$ implies $\mathcal{A} \models F_A$.

Assume $\mathcal{A} \models G_A$. Then the following hold:

(1) $\mathcal{A} \models \forall x, y\ (\pi_r^i(x,y) \approx 1 \wedge \pi^e(x,y) \approx 1 \rightarrow \bot)$,

(2) $\mathcal{A} \models \forall x, y\ (\pi_r^i(x,y) \approx 1 \wedge \pi_r^i(y,x) \approx 1 \rightarrow F(y,x) \approx 1)$,

(3) $\mathcal{A} \models \forall x, y\ (\pi^e(x,y) \approx 1 \rightarrow F(x,y) \approx 0)$,

(4) $\mathcal{A} \models \forall x, y\ (\pi^e(x,y) \approx 1 \rightarrow F(y,x) \approx 0)$,

(5) $\mathcal{A} \models \forall x, y\ (F(x,y) \approx 1 \rightarrow F(y,x) \approx 1)$.

We define a new structure $\mathcal{B}$ which has the same universe and the same interpretations for π^e, π_r^i and F as $\mathcal{A}$ and, in addition, an interpretation $E_\mathcal{B}$ such that $E(x,y) \approx 1$ if and only if $F(x,y) \approx 1 \vee \pi_r^i(x,y) \approx 1$. We can observe the following:

- From the definition of $E_\mathcal{B}$ it follows that $\mathcal{B} \models \forall x, y\ (\pi_r^i(x,y) \approx 1 \rightarrow E(x,y) \approx 1)$, i.e. $\mathcal{B} \models A_1$.
- From the definition of $E_\mathcal{B}$ together with (1), (3) and (4) it follows that $\mathcal{B} \models \forall x, y\ (\pi^e(x,y) \approx 1 \rightarrow E(x,y) \approx 0)$, i.e. $\mathcal{B} \models A_2$.
- From the definition of $E_\mathcal{B}$ it follows that $\mathcal{B} \models \forall x, y\ (F(x,y) \approx 1 \rightarrow E(x,y) \approx 1)$, i.e. $\mathcal{B} \models A_3$.
- From the definition of $E_\mathcal{B}$ together with (5) it follows that $\mathcal{B} \models \forall x, y\ (F(x,y) \approx 1 \rightarrow E(y,x) \approx 1)$, i.e. $\mathcal{B} \models A_4$.
- From the definition of $E_\mathcal{B}$ together with (2) and (5) it follows that $\mathcal{B} \models \forall x, y\ (E(x,y) \approx 1 \wedge E(y,x) \approx 1 \rightarrow F(x,y) \approx 1)$, i.e. $\mathcal{B} \models A_5$.

Then, since $\mathcal{A}$ is interpreted the same as $\mathcal{B}$ on all predicates except for E and there exists an interpretation $E_\mathcal{B}$ such that $\mathcal{B} \models A_i$ for $i \in \{1, ..., 5\}$, we have $\mathcal{A} \models \exists E\ (A_1 \wedge ... \wedge A_5)$. Thus, $\mathcal{A} \models F_A$, so $\mathcal{A} \models G_A$ implies $\mathcal{A} \models F_A$. □

An axiomatization G_B such that $G_B \equiv \exists E\ \mathbf{MinDG}(r) \wedge \mathbf{MaxDG}(1) \wedge \mathsf{Tr}^+(E,F)$ can be computed in the same way. We consider all predicates to be functions with codomain $\{0,1\}$. From the axiomatization $\mathsf{Ax}_A = \mathbf{MinDG}(r) \wedge \mathbf{MaxDG}(1) \wedge \mathsf{Tr}^-(E,F)$ we then get the following clauses:

$$\begin{aligned}
&\forall x, y \quad \pi_r^i(x,y) \approx 1 \rightarrow E(x,y) \approx 1 && (B_1)\\
&\forall x, y \quad \pi^e(x,y) \approx 1 \rightarrow E(x,y) \approx 0 && (B_2)\\
&\forall x, y \quad F(x,y) \approx 1 \rightarrow E(x,y) \vee E(y,x) \approx 1 && (B_3)\\
&\forall x, y \quad E(x,y) \approx 1 \rightarrow F(x,y) \approx 1 && (B_4)\\
&\forall x, y \quad E(y,x) \approx 1 \rightarrow F(x,y) \approx 1 && (B_5)
\end{aligned}$$

Eliminating E using SEH-PILoT we obtain a result that can be manually simplified to the following set of clauses:

$$\begin{aligned}
&\forall x, y \quad \pi_r^i(x,y) \approx 1 \wedge \pi^e(x,y) \approx 1 \rightarrow \bot && (B_1')\\
&\forall x, y \quad \pi^e(x,y) \approx 1 \wedge \pi^e(y,x) \approx 1 \rightarrow F(y,x) \approx 0 && (B_2')\\
&\forall x, y \quad \pi_r^i(x,y) \approx 1 \rightarrow F(x,y) \approx 1 && (B_3')\\
&\forall x, y \quad \pi_r^i(x,y) \approx 1 \rightarrow F(y,x) \approx 1 && (B_4')\\
&\forall x, y \quad F(x,y) \approx 1 \rightarrow F(y,x) \approx 1 && (B_5')
\end{aligned}$$

We prove the correctness of the newly obtained axiomatization.

Proof: Let $F_B = \exists E\ (B_1 \wedge ... \wedge B_5)$ and $G_B = B'_1 \wedge ... \wedge B'_5$. We show that for any structure $\mathcal{A}$ we have $\mathcal{A} \models F_B$ if and only if $\mathcal{A} \models G_B$.

1. ("⇒"): We show that $\mathcal{A} \models F_B$ implies $\mathcal{A} \models G_B$.

Assume $\mathcal{A} \models F_B$. Then there exists a structure $\mathcal{B}$ with the same universe and the same interpretations for π^e, π^i_r and F as $\mathcal{A}$ and, in addition, an interpretation $E_\mathcal{B}$ for E such that

(1) $\mathcal{B} \models \forall x, y\ (\pi^i_r(x, y) \approx 1 \rightarrow E(x, y) \approx 1)$,

(2) $\mathcal{B} \models \forall x, y\ (\pi^e(x, y) \approx 1 \rightarrow E(x, y) \approx 0)$,

(3) $\mathcal{B} \models \forall x, y\ (F(x, y) \approx 1 \rightarrow E(x, y) \approx 1 \vee E(y, x) \approx 1)$,

(4) $\mathcal{B} \models \forall x, y\ (E(x, y) \approx 1 \rightarrow F(x, y) \approx 1)$,

(5) $\mathcal{B} \models \forall x, y\ (E(y, x) \approx 1 \rightarrow F(x, y) \approx 1)$.

We can observe the following:

- From (1) and (2) it follows that $\mathcal{B} \models \forall x, y\ (\pi^i_r(x, y) \approx 1 \wedge \pi^e(x, y) \approx 1 \rightarrow \bot)$, i.e. $\mathcal{B} \models B'_1$.
- From (2) and (3) it follows that $\mathcal{B} \models \forall x, y\ (\pi^e(x, y) \approx 1 \wedge \pi^e(y, x) \approx 1 \rightarrow F(y, x) \approx 0)$, i.e. $\mathcal{B} \models B'_2$.
- From (1) and (4) it follows that $\mathcal{B} \models \forall x, y\ (\pi^i_r(x, y) \approx 1 \rightarrow F(x, y) \approx 1)$, i.e. $\mathcal{B} \models B'_3$.
- From (1) and (5) it follows that $\mathcal{B} \models \forall x, y\ (\pi^i_r(x, y) \approx 1 \rightarrow F(y, x) \approx 1)$, i.e. $\mathcal{B} \models B'_4$.
- From (3), (4) and (5) it follows that $\mathcal{B} \models \forall x, y\ (F(x, y) \approx 1 \rightarrow F(y, x) \approx 1)$, i.e. $\mathcal{B} \models B'_5$.

Since all the predicate symbols except for E are interpreted the same in $\mathcal{A}$ and $\mathcal{B}$, we then also have $\mathcal{A} \models B'_i$ for $i \in \{1, ..., 5\}$. Thus, $\mathcal{A} \models G_B$, so $\mathcal{A} \models F_B$ implies $\mathcal{A} \models G_B$.

2. ("⇐"): We show that $\mathcal{A} \models G_B$ implies $\mathcal{A} \models F_B$.

Assume $\mathcal{A} \models G_B$. Then the following hold:

(1) $\mathcal{A} \models \forall x, y\ (\pi^i_r(x, y) \approx 1 \wedge \pi^e(x, y) \approx 1 \rightarrow \bot)$,

(2) $\mathcal{A} \models \forall x, y\ (\pi^e(x, y) \approx 1 \wedge \pi^e(y, x) \approx 1 \rightarrow F(y, x) \approx 0)$,

(3) $\mathcal{A} \models \forall x, y\ (\pi^i_r(x, y) \approx 1 \rightarrow F(x, y) \approx 1)$,

(4) $\mathcal{A} \models \forall x, y\ (\pi^i_r(x, y) \approx 1 \rightarrow F(y, x) \approx 1)$,

(5) $\mathcal{A} \models \forall x, y\ (F(x, y) \approx 1 \rightarrow F(y, x) \approx 1)$.

We define a new structure $\mathcal{B}$ which has the same universe and the same interpretations for π^e, π^i_r and F as $\mathcal{A}$ and, in addition, an interpretation $E_\mathcal{B}$ such that $E(x, y) \approx 0$ if and only if $F(x, y) \approx 0 \vee \pi^e(x, y) \approx 1$. We can observe the following:

- From the definition of $E_\mathcal{B}$ together with (1) and (3) it follows that $\mathcal{B} \models \forall x, y\ (\pi^i_r(x, y) \approx 1 \rightarrow E(x, y) \approx 1)$, i.e. $\mathcal{B} \models B_1$.

- From the definition of $E_{\mathcal{B}}$ it follows that $\mathcal{B} \models \forall x, y\ (\pi^e(x,y) \approx 1 \rightarrow E(x,y) \approx 0)$, i.e. $\mathcal{B} \models B_2$.
- From the definition of $E_{\mathcal{B}}$ together with (2) and (5)it follows that $\mathcal{B} \models \forall x, y\ (F(x,y) \approx 1 \rightarrow E(x,y) \approx 1 \vee E(y,x) \approx 1)$, i.e. $\mathcal{B} \models B_3$.
- From the definition of $E_{\mathcal{B}}$ it follows that $\mathcal{B} \models \forall x, y\ (E(x,y) \approx 1 \rightarrow F(x,y) \approx 1)$, i.e. $\mathcal{B} \models B_4$.
- From the definition of $E_{\mathcal{B}}$ together with (5) it follows that $\mathcal{B} \models \forall x, y\ E(y,x) \approx 1 \rightarrow F(x,y) \approx 1)$, i.e. $\mathcal{B} \models B_5$.

Then, since $\mathcal{A}$ is interpreted the same as $\mathcal{B}$ on all predicates except for E and there exists an interpretation $E_{\mathcal{B}}$ such that $\mathcal{B} \models B_i$ for $i \in \{1, ..., 5\}$, we have $\mathcal{A} \models \exists E\ (B_1 \wedge ... \wedge B_5)$. Thus, $\mathcal{A} \models F_B$, so $\mathcal{A} \models G_B$ implies $\mathcal{A} \models F_B$. □

Hierarchical Reasoning / Property-directed Symbol Elimination

In the previous step we have obtained the axiomatizations G_A and G_B for graph classes A and B, respectively.

G_A			
$\forall x, y$	$\pi^i_r(x,y) \wedge \pi^e(x,y)$	$\rightarrow$	$\bot$
$\forall x, y$	$\pi^i_r(x,y) \wedge \pi^i_r(y,x)$	$\rightarrow$	$F(y,x)$
$\forall x, y$	$\pi^e(x,y)$	$\rightarrow$	$\neg F(x,y)$
$\forall x, y$	$\pi^e(x,y)$	$\rightarrow$	$\neg F(y,x)$
$\forall x, y$	$F(x,y)$	$\rightarrow$	$F(y,x)$

G_B			
$\forall x, y$	$\pi^i_r(x,y) \wedge \pi^e(x,y)$	$\rightarrow$	$\bot$
$\forall x, y$	$\pi^e(x,y) \wedge \pi^e(y,x)$	$\rightarrow$	$\neg F(y,x)$
$\forall x, y$	$\pi^i_r(x,y)$	$\rightarrow$	$F(x,y)$
$\forall x, y$	$\pi^i_r(x,y)$	$\rightarrow$	$F(y,x)$
$\forall x, y$	$F(x,y)$	$\rightarrow$	$F(y,x)$

We first analyze whether graph class A is contained in graph class B, i.e. whether $A \subseteq B$. For this we have to check whether $G_A \models_{\mathcal{T}} G_B$, i.e. whether $G_A \wedge \neg G_B$ is unsatisfiable w.r.t. $\mathcal{T}$, where $\neg G_B$ is the disjunction of the following ground formulae (we ignore the negation of the first clause and last clause, as these clauses are obviously implied by G_A):

(g_1) $\pi^e(a,b) \wedge \pi^e(b,a) \wedge F(b,a)$
(g_2) $\pi^i_r(a,b) \wedge \neg F(a,b)$
(g_3) $\pi^i_r(a,b) \wedge \neg F(b,a)$

Here we have $\mathcal{T} = \mathcal{T}_d \cup \mathit{UIF}_r$, where $\mathcal{T}_d$ is a theory describing the properties of d and r is considered to be an uninterpreted function symbol. Let $\mathcal{T}_0$ be the disjoint combination of the theory of an infinite set $\mathcal{T}_{\mathcal{IS}}$ (sort p) and linear real arithmetic (sort num). We consider $\mathcal{T}_d$ to be one of the following extensions of $\mathcal{T}_0$:

(1) $\mathcal{T}^m_d = \mathcal{T}_0 \cup \mathcal{K}_m$, where $\mathcal{K}_m$ are the axioms of a metric, i.e.

(d_1) $\forall x, y$ $d(x,y) \geq 0$
(d_2) $\forall x, y, z$ $d(x,y) \leq d(x,z) + d(z,y)$
(d_3) $\forall x, y$ $d(x,y) \approx d(y,x)$
(d_4) $\forall x, y$ $x \approx y \rightarrow d(x,y) \approx 0$
(d_5) $\forall x, y$ $d(x,y) \approx 0 \rightarrow x \approx y$

(2) $\mathcal{T}^s_d$, the extension of $\mathcal{T}_0$ with axiom (d_3)

(3) $\mathcal{T}^p_d$, the extension of $\mathcal{T}_0$ with axiom (d_1)

(4) $\mathcal{T}^u_d$, the extension of $\mathcal{T}_0$ with an uninterpreted function d

In Section 6.1 we proved that all these theories satisfy suitable locality properties. For testing entailment, we can consider the set of all instances of G_A in which the variables of sort p are replaced with constants a, b and then use a method for checking ground satisfiability of $G_A[T] \wedge g_i$ w.r.t. $\mathcal{T}_d \cup UIF_r$, where $\mathcal{T}_d \in \{\mathcal{T}_d^u, \mathcal{T}_d^p, \mathcal{T}_d^s, \mathcal{T}_d^m\}$. For this we use H-PILoT, in which we enforce the right instantiation by adding relevant instances to the query. This allows us to check that $G_A[T] \wedge g_i$ is unsatisfiable for $i = 1$, but satisfiable for $i \in \{2, 3\}$ (this is the case for all four theories). For cases 2 and 3 can use Algorithm 2 to derive conditions on parameters under which $G_A[T] \wedge g_i$ is unsatisfiable. We use SEH-PILoT for this, with specification type HPILOT and specification theory REAL_CLOSED_FIELDS in mode GENERATE_CONSTRAINTS. The computed formulae are then constraints over the given parameters which guarantee that the inclusion between the classes holds. We here give two examples:

(i) We consider d and r to be parameters, i.e. we eliminate only F from $G_A[T] \wedge g_i$. For $\mathcal{T}_d^m$ we obtain using SEH-PILoT the condition

$C^{d,r} = \forall x, y(x \not\approx y \wedge d(x, y) \leq 1 \wedge d(x, y) \leq r(x) \rightarrow d(y, x) \leq r(y))$.

(ii) We consider only r to be a parameter, i.e. we eliminate the symbols F and d. For $\mathcal{T}_d^s$ we obtain using SEH-PILoT the condition

$C^r = \forall x, y(r(y) < 1 \wedge x \not\approx y \rightarrow r(y) \geq r(x))$.

This condition holds for example if $r(x) \approx r(y)$ for all x, y, i.e. if r is a constant function. Adding this as an additional condition we get unsatisfiability of $G_A[T] \wedge g_i$ with $i \in \{2, 3\}$ for $\mathcal{T}_d^m$ and $\mathcal{T}_d^s$, but not for $\mathcal{T}_d^u$ and $\mathcal{T}_d^p$.

Checking the Other Inclusion

We now analyze whether $B \subseteq A$. For this we check whether $G_B \wedge \neg G_A$ is unsatisfiable w.r.t. $\mathcal{T}$, where $\neg G_A$ is the disjunction of the following ground formulae (we ignore the negation of the first and last clause, as these clauses are obviously implied by G_B):

(g_1) $\pi^i(a, b) \wedge \pi^i(b, a) \wedge \neg F(b, a)$
(g_2) $\pi^e(a, b) \wedge F(a, b)$
(g_3) $\pi^e(a, b) \wedge F(b, a)$

We check ground satisfiability of $G_B[T] \wedge g_i$ w.r.t. $\mathcal{T} \in \{\mathcal{T}_d^u, \mathcal{T}_d^p, \mathcal{T}_d^s, \mathcal{T}_d^m\}$. For $\mathcal{T}_d^s$ and $\mathcal{T}_d^m$ we obtain unsatisfiability of $G_B[T] \wedge g_i$ for $i \in \{1, 2, 3\}$, thus we have proved that the inclusion holds for these two theories. For $\mathcal{T}_d^p$ and $\mathcal{T}_d^u$ we get satisfiability for cases 2 and 3. We can use Algorithm 2 to obtain conditions on parameters such that $G_B[T] \wedge g_2$ and $G_B[T] \wedge g_3$ is unsatisfiable. If we consider d and r to be parameters, i.e. we eliminate only F from $G_B[T] \wedge g_i$, we obtain the condition

$C^{d,r} = \forall x, y(d(y, x) > 1 \vee d(x, y) \leq 1 \vee d(x, y) \leq r(x) \vee x \approx y)$.

It is easy to see that this condition holds if d is symmetric, since from $d(x, y) \approx d(y, x)$ it follows that $d(y, x) > 1 \vee d(x, y) \leq 1$ holds.

Combining the conditions for $A \subseteq B$ and $B \subseteq A$ we can conclude that the two graph classes $A = (\mathbf{MinDG}(r) \cap \mathbf{MaxDG}(1))^-$ and $B = (\mathbf{MinDG}(r) \cap \mathbf{MaxDG}(1))^+$ are identical for instance if the communication range r is constant (i.e. all vertices have the same communication range) and the distance function d is symmetric (i.e. it satisfies condition (d_3) of the metric axioms). ■

6.4 Conclusion

In this chapter we demonstrated by example that hierarchical reasoning and quantifier elimination is a powerful tool to analyze properties of graph classes defined by general and Euclidean metrics. For this, we established locality results for theory extensions involving distances.

First, we looked at simple geometric graph classes which can be described by specifying when an edge between two vertices in the graph exists and when it does not exist. We illustrated how proving properties of such graph classes can be done by showing inclusion between graph classes. As an example, we proved planarity of Gabriel graphs and relative neighborhood graphs by showing that the classes of Gabriel graphs and relative neighborhood graphs are contained in the class of plane drawings.

Secondly, we considered classes of geometric graphs which can be described using suitable inclusion, exclusion and transfer predicates. We defined simple transformations on such graph classes which lead to a class of symmetric subgraphs or to a class symmetric supergraphs. Checking containedness of such transformed graph classes can be done similar to the case of the simple classes of graphs. As an example, we analyzed the inclusions between the class of symmetric supergraphs and symmetric subgraphs of quasi unit disk graphs.

For checking containedness of graph classes we use a two-layered approach combining general second-order symbol elimination and property-directed symbol elimination. For eliminating existentially quantified predicates from universal first-order formulae we used hierarchical reasoning and symbol elimination, but other methods can also be used, for instance a version of the ordered resolution calculus. This first step yields solely geometric conditions for class inclusion, which can be checked easily. If class containment cannot be proved, we can compute conditions on parameters that guarantee containedness, using property-directed symbol elimination.

6.4.1 Future Work

There are several ways in which the work described in this chapter can be continued. On the one hand, we plan to significantly expand the set of graph classes that can be analyzed with our tool set. On the other hand, we would like to consider more transformations than the ones presented in this thesis. In [17] Böltz and Frey define a set of 8 base graph classes (including **MinDG**, **MaxDG** and **QUDG**) and a class of 16 simple graph transformations (including $.^+$ and $.^-$) and analyze, using different methods, all possible containedness conditions between the 128 classes obtained by applying those transformations on the base graph classes. For future work it would be interesting to check these results automatically with the methods presented in this thesis.

As a further direction for subsequent work it is also possible to investigate more kinds of graph properties, for example spanner properties (Euclidean, topological, energy) and degree limitation. These concepts are of interest for algorithm design in wireless graph models but also for graphs in general.

7 Conclusion

In this thesis we studied methods for reasoning in complex theories based on hierarchical reasoning and symbol elimination, and investigated the applicability of these approaches in three different application areas. First, we showed how to use hierarchical reasoning and symbol elimination in the verification of parametric systems, by proposing an algorithm for the goal-directed generation of inductive invariants. Secondly, we investigated an approach for finding high-level explanations for subsumption in $\mathcal{EL}$ and $\mathcal{EL}^+$, based on the computation of $\leq$-interpolating terms in the theory of semilattices with monotone operators. Thirdly, we illustrated on different examples how to reason about geometric graph classes and transformations thereof, using general and property-directed symbol elimination to show containedness between such graph classes.

In Chapter 3 we analyzed methods for symbol elimination in combinations and extensions of theories. On the one hand, we established a result for quantifier elimination in combinations of disjoint theories and we showed that virtual substitution can be used to apply quantifier elimination in the combination of the theory of an infinite set and the theory of real closed fields. On the other hand, we presented an algorithm for property-directed symbol elimination in local theory extensions and proposed an improvement.

The invariant strengthening algorithm proposed in Chapter 4 uses property-directed symbol elimination to iteratively strenghten an initial property to an inductive invariant. We proved correctness of the algorithm under certain assumptions, but also analyzed situations in which these assumptions can be relaxed. We showed that termination can be guaranteed in some situations, but not in general. However, in some cases the choice of the parameters can have an effect on the termination, as it was shown on an example. Several refinements of the algorithm were presented and the applicability of the algorithm and its refinements was illustrated on examples. The implementation of the algorithm was described briefly. In the conclusion of Chapter 4 we discussed benefits of our approach compared to other approaches.

In Chapter 5 we proposed an algorithm for computing a certain kind of interpolants in the description logics $\mathcal{EL}$ and $\mathcal{EL}^+$, which serve as high-level explanations for subsumption relations that occur as a result of combining two ontologies or extending one ontology with another. The approach is based on the reduction to the uniform word problem in classes of semilattices with monotone operators and the computation of $\leq$-interpolating terms using resolution. In the presence of role inclusions, hierarchical reasoning is also necessary. As it may impose so-called mixed instances, a separation of those, also using $\leq$-interpolation, can be necessary. To reduce the number of axioms to consider, we use unsatisfiable core computation. An implementation and three illustrative examples were presented.

Certain types of geometric graph classes as well as simple transformations yielding symmetric graph classes were investigated in Chapter 6. Those classes are defined by exclusion, inclusion and transfer predicates which describe geometric conditions based on a distance function. For checking containedness of graph classes we first used a form of second-order quantifier elimination to eliminate the edge predicate E. In a second step we then could

use hirearchical resoning and property-directed symbol elimination to either prove containedness or to obtain geometric conditions on parameters such that containedness is guaranteed.

The applications shown in this thesis have in common that the problems are described using complex theories, for which efficient resoning is in general difficult. By reducing the problems in a suitable way, for instance using hierarchical reasoning to make a reduction to the base theory, these problems become easier to solve. We can not only check satisfiability of the reduced formulae, the parametric nature of many problems also allows us to use property-directed symbol elimination to derive conditions on parameters which guarantee unsatisfiability.

In conclusion, it was shown in this thesis that combining different methods for symbol elimination, i.e. property-directed as well as general symbol elimination, and hierarchical reasoning, in particular for theory extensions satisfying a locality property, is a useful approach to solve interesting problems in a wide range of application areas.

Bibliography

[1] Mohammad Abdulaziz, Kurt Mehlhorn, and Tobias Nipkow: *Trustworthy Graph Algorithms (Invited Talk).* In Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen (editors): *Proc. 44th Int. Symposium on Mathematical Foundations of Computer Science (MFCS 2019)*, volume 138 of *LIPIcs*, pages 1:1–1:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[2] Francesco Alberti, Roberto Bruttomesso, Silvio Ghilardi, Silvio Ranise, and Natasha Sharygina: *An extension of lazy abstraction with interpolation for programs with arrays.* Formal Methods in System Design, 45(1):63–109, 2014.

[3] Christian Alrabbaa, Franz Baader, Stefan Borgwardt, Patrick Koopmann, and Alisa Kovtunova: *Finding Small Proofs for Description Logic Entailments: Theory and Practice.* In Elvira Albert and Laura Kovács (editors): *LPAR 2020: 23rd International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Alicante, Spain, May 22-27, 2020*, volume 73 of *EPiC Series in Computing*, pages 32–67. EasyChair, 2020. `https://easychair.org/publications/volume/LPAR23`.

[4] Franz Baader: *Terminological Cycles in a Description Logic with Existential Restrictions.* In Georg Gottlob and Toby Walsh (editors): *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003*, pages 325–330. Morgan Kaufmann, 2003. `http://ijcai.org/Proceedings/03/Papers/048.pdf`.

[5] Franz Baader, Sebastian Brandt, and Carsten Lutz: *Pushing the $\mathcal{EL}$ Envelope.* In Leslie Pack Kaelbling and Alessandro Saffiotti (editors): *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30 - August 5, 2005*, pages 364–369. Professional Book Center, 2005.

[6] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider (editors): *The Description Logic Handbook: Theory, Implementation, and Applications.* Cambridge University Press, 2003, ISBN 0-521-78176-0.

[7] Franz Baader and Bernhard Hollunder: *Embedding Defaults into Terminological Knowledge Representation Formalisms.* J. Autom. Reasoning, 14(1):149–180, 1995. `https://doi.org/10.1007/BF00883932`.

[8] Franz Baader, Carsten Lutz, and Boontawee Suntisrivaraporn: *Efficient Reasoning in $\mathcal{EL}^+$.* In Bijan Parsia, Ulrike Sattler, and David Toman (editors): *Proceedings of the 2006 International Workshop on Description Logics (DL2006), Windermere, Lake District, UK, May 30 - June 1, 2006*, volume 189 of *CEUR Workshop Proceedings.* CEUR-WS.org, 2006. `http://ceur-ws.org/Vol-189/submission_8.pdf`.

[9] Franz Baader, Carsten Lutz, and Boontawee Suntisrivaraporn: *Is tractable reasoning in extensions of the description logic $\mathcal{EL}$ useful in practice?* Journal of Logic,

Language and Information, 2007. Special Issue on Method for Modality (M4M), 2007.

[10] Franz Baader, Rafael Peñaloza, and Boontawee Suntisrivaraporn: *Pinpointing in the Description Logic $\mathcal{EL}$.* In Diego Calvanese, Enrico Franconi, Volker Haarslev, Domenico Lembo, Boris Motik, Anni-Yasmin Turhan, and Sergio Tessaris (editors): *Proceedings of the 2007 International Workshop on Description Logics (DL2007), Brixen-Bressanone, near Bozen-Bolzano, Italy, 8-10 June, 2007*, volume 250 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007. `http://ceur-ws.org/Vol-250/paper_16.pdf`.

[11] Franz Baader, Rafael Peñaloza, and Boontawee Suntisrivaraporn: *Pinpointing in the Description Logic $\mathcal{EL}^+$.* In Joachim Hertzberg, Michael Beetz, and Roman Englert (editors): *KI 2007: Advances in Artificial Intelligence, 30th Annual German Conference on AI, KI 2007, Osnabrück, Germany, September 10-13, 2007, Proceedings*, volume 4667 of *Lecture Notes in Computer Science*, pages 52–67. Springer, 2007. `https://doi.org/10.1007/978-3-540-74565-5_7`.

[12] Franz Baader and Boontawee Suntisrivaraporn: *Debugging SNOMED CT Using Axiom Pinpointing in the Description Logic $\mathcal{EL}^+$.* In Ronald Cornet and Kent A. Spackman (editors): *Proceedings of the Third International Conference on Knowledge Representation in Medicine, Phoenix, Arizona, USA, May 31st - June 2nd, 2008*, volume 410 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008. `http://ceur-ws.org/Vol-410/Paper01.pdf`.

[13] Leo Bachmair and Harald Ganzinger: *Resolution Theorem Proving.* In John Alan Robinson and Andrei Voronkov (editors): *Handbook of Automated Reasoning (in 2 volumes)*, pages 19–99. Elsevier and MIT Press, 2001. `https://doi.org/10.1016/b978-044450813-3/50004-7`.

[14] Leo Bachmair, Harald Ganzinger, and Uwe Waldmann: *Refutational Theorem Proving for Hierarchic First-Order Theories.* Appl. Algebra Eng. Commun. Comput., 5:193–212, 1994.

[15] Lali Barrière, Pierre Fraigniaud, Lata Narayanan, and Jaroslav Opatrny: *Robust position-based routing in wireless ad hoc networks with irregular transmission ranges.* Wireless Communications and Mobile Computing, 3(2):141–153, Mar 2003, ISSN 15308669.

[16] Dirk Beyer, Thomas A. Henzinger, Rupak Majumdar, and Andrey Rybalchenko: *Invariant Synthesis for Combined Theories.* In Byron Cook and Andreas Podelski (editors): *Verification, Model Checking, and Abstract Interpretation, 8th International Conference, VMCAI 2007, Nice, France, January 14-16, 2007, Proceedings*, volume 4349 of *Lecture Notes in Computer Science*, pages 378–394. Springer, 2007.

[17] Lucas Böltz and Hannes Frey: *Automatically Testing Containedness between Geometric Graph Classes defined by Inclusion, Exclusion, and Transfer Axioms under Simple Transformations.* Inf., 13(12):578, 2022. `https://doi.org/10.3390/info13120578`.

[18] Lucas Böltz, Hannes Frey, Dennis Peuter, and Viorica Sofronie-Stokkermans: *On Testing Containedness Between Geometric Graph Classes using Second-order Quantifier Elimination and Hierarchical Reasoning (Short Paper).* In Renate A. Schmidt,

Christoph Wernhard, and Yizheng Zhao (editors): *Proceedings of the Second Workshop on Second-Order Quantifier Elimination and Related Topics (SOQE 2021) associated with the 18th International Conference on Principles of Knowledge Representation and Reasoning (KR 2021), Online Event, November 4, 2021*, volume 3009 of *CEUR Workshop Proceedings*, pages 37–45. CEUR-WS.org, 2021. https://ceur-ws.org/Vol-3009/short1.pdf.

[19] Alexander Borgida, Enrico Franconi, and Ian Horrocks: *Explaining $\mathcal{ALC}$ Subsumption.* In Werner Horn (editor): *ECAI 2000, Proceedings of the 14th European Conference on Artificial Intelligence, Berlin, Germany, August 20-25, 2000*, pages 209–213. IOS Press, 2000.

[20] Prosenjit Bose, Pat Morin, Ivan Stojmenović, and Jorge Urrutia: *Routing with Guaranteed Delivery in Ad Hoc Wireless Networks.* Wireless Networks, 7(6):609–616, nov 2001.

[21] Aaron R. Bradley: *IC3 and beyond: Incremental, Inductive Verification.* In P. Madhusudan and Sanjit A. Seshia (editors): *Computer Aided Verification - 24th International Conference, CAV 2012, Berkeley, CA, USA, July 7-13, 2012 Proceedings*, volume 7358 of *Lecture Notes in Computer Science*, page 4. Springer, 2012.

[22] Aaron R. Bradley and Zohar Manna: *The calculus of computation - decision procedures with applications to verification.* Springer, 2007. https://doi.org/10.1007/978-3-540-74113-8.

[23] Aaron R. Bradley and Zohar Manna: *Property-directed incremental invariant generation.* Formal Asp. Comput., 20(4-5):379–405, 2008.

[24] Aaron R. Bradley, Zohar Manna, and Henny B. Sipma: *What's Decidable About Arrays?* In E. Allen Emerson and Kedar S. Namjoshi (editors): *Verification, Model Checking, and Abstract Interpretation, 7th International Conference, VMCAI 2006, Charleston, SC, USA, January 8-10, 2006, Proceedings*, volume 3855 of *Lecture Notes in Computer Science*, pages 427–442. Springer, 2006.

[25] Christopher W. Brown and James H. Davenport: *The complexity of quantifier elimination and cylindrical algebraic decomposition.* In Dongming Wang (editor): *Symbolic and Algebraic Computation, International Symposium, ISSAC 2007, Waterloo, Ontario, Canada, July 28 - August 1, 2007, Proceedings*, pages 54–60. ACM, 2007. https://doi.org/10.1145/1277548.1277557.

[26] Roberto Bruttomesso, Silvio Ghilardi, and Silvio Ranise: *Quantifier-free interpolation in combinations of equality interpolating theories.* ACM Trans. Comput. Log., 15(1):5:1–5:34, 2014.

[27] Diego Calvanese, Silvio Ghilardi, Alessandro Gianola, Marco Montali, and Andrey Rivkin: *Model Completeness, Uniform Interpolants and Superposition Calculus.* J. Autom. Reason., 65(7):941–969, 2021. https://doi.org/10.1007/s10817-021-09596-x.

[28] Diego Calvanese, Silvio Ghilardi, Alessandro Gianola, Marco Montali, and Andrey Rivkin: *Combination of Uniform Interpolants via Beth Definability.* J. Autom. Reason., 66(3):409–435, 2022. https://doi.org/10.1007/s10817-022-09627-1.

[29] George E. Collins: *Quantifier Elimination for the Elementary Theory of Real Closed Fields by Cylindrical Algebraic Decomposition.* Second GI Conf. Automata Theory and Formal Languages, pages 134–183, 1975.

[30] Sylvain Conchon, Amit Goel, Sava Krstic, Alain Mebsout, and Fatiha Zaïdi: *Cubicle: A Parallel SMT-Based Model Checker for Parameterized Systems - Tool Paper.* In P. Madhusudan and Sanjit A. Seshia (editors): *Computer Aided Verification - 24th International Conference, CAV 2012, Berkeley, CA, USA, July 7-13, 2012 Proceedings*, volume 7358 of *Lecture Notes in Computer Science*, pages 718–724. Springer, 2012.

[31] Bruno Courcelle: *On the Expression of Monadic Second-Order Graph Properties Without Quantifications Over Sets of Edges (Extended Abstract).* In *Proceedings of the Fifth Annual Symposium on Logic in Computer Science (LICS '90), Philadelphia, Pennsylvania, USA, June 4-7, 1990*, pages 190–196. IEEE Computer Society, 1990. https://doi.org/10.1109/LICS.1990.113745.

[32] Bruno Courcelle: *The Monadic Second-order Logic of Graphs VI: On Several Representations of Graphs by Relational Structures.* Discret. Appl. Math., 63(2):199–200, 1995. https://doi.org/10.1016/0166-218X(95)00006-D.

[33] Bruno Courcelle: *The Expression of Graph Properties and Graph Transformations in Monadic Second-Order Logic.* In Grzegorz Rozenberg (editor): *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*, pages 313–400. World Scientific, 1997, ISBN 9810228848.

[34] Bruno Courcelle: *The monadic second-order logic of graphs XVI : Canonical graph decompositions.* Log. Methods Comput. Sci., 2(2), 2006. https://doi.org/10.2168/LMCS-2(2:2)2006.

[35] Bruno Courcelle: *Monadic Second-Order Logic for Graphs: Algorithmic and Language Theoretical Applications.* In Adrian-Horia Dediu, Armand-Mihai Ionescu, and Carlos Martín-Vide (editors): *Language and Automata Theory and Applications, Third International Conference, LATA 2009, Tarragona, Spain, April 2-8, 2009. Proceedings*, volume 5457 of *Lecture Notes in Computer Science*, pages 19–22. Springer, 2009. https://doi.org/10.1007/978-3-642-00982-2_2.

[36] William Craig: *Three Uses of the Herbrand-Gentzen Theorem in Relating Model Theory and Proof Theory.* J. Symb. Log., 22(3):269–285, 1957. https://doi.org/10.2307/2963594.

[37] James H. Davenport and Joos Heintz: *Real Quantifier Elimination is Doubly Exponential.* J. Symb. Comput., 5(1/2):29–35, 1988. https://doi.org/10.1016/S0747-7171(88)80004-X.

[38] Isil Dillig, Thomas Dillig, Boyang Li, and Kenneth L. McMillan: *Inductive invariant generation via abductive inference.* In Antony L. Hosking, Patrick Th. Eugster, and Cristina V. Lopes (editors): *Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications, OOPSLA 2013, part of SPLASH 2013, Indianapolis, IN, USA, October 26-31, 2013*, pages 443–456. ACM, 2013.

[39] Christian Doczkal and Damien Pous: *Graph Theory in Coq: Minors, Treewidth, and Isomorphisms*. J. Autom. Reason., 64(5):795–825, 2020. https://doi.org/10.1007/s10817-020-09543-2.

[40] Andreas Dolzmann: *Algorithmic strategies for applicable real quantifier elimination*. PhD thesis, University of Passau, Germany, 2000. http://elib.ub.uni-passau.de/opus/volltexte/2001/6/index.html.

[41] Andreas Dolzmann and Thomas Sturm: *Redlog: Computer Algebra Meets Computer Logic*. ACM SIGSAM Bulletin, 31(2):2–9, 1997.

[42] Andreas Dolzmann, Thomas Sturm, and Volker Weispfenning: *A New Approach for Automatic Theorem Proving in Real Geometry*. J. Autom. Reason., 21(3):357–380, 1998. https://doi.org/10.1023/A:1006031329384.

[43] Johannes Faber, Swen Jacobs, and Viorica Sofronie-Stokkermans: *Verifying CSP-OZ-DC Specifications with Complex Data Types and Timing Parameters*. In Jim Davies and Jeremy Gibbons (editors): *Integrated Formal Methods, 6th International Conference, IFM 2007, Oxford, UK, July 2-5, 2007, Proceedings*, volume 4591, pages 233–252. Springer, 2007.

[44] Stephan Falke and Deepak Kapur: *When Is a Formula a Loop Invariant?* In Narciso Martí-Oliet, Peter Csaba Ölveczky, and Carolyn L. Talcott (editors): *Logic, Rewriting, and Concurrency - Essays dedicated to José Meseguer on the Occasion of His 65th Birthday*, volume 9200 of *Lecture Notes in Computer Science*, pages 264–286. Springer, 2015.

[45] Melvin Fitting: *First-Order Logic and Automated Theorem Proving, Second Edition*. Graduate Texts in Computer Science. Springer, 1996, ISBN 978-1-4612-7515-2. https://doi.org/10.1007/978-1-4612-2360-3.

[46] Marie Fortin, Boris Konev, and Frank Wolter: *Interpolants and Explicit Definitions in Extensions of the Description Logic EL*. In Gabriele Kern-Isberner, Gerhard Lakemeyer, and Thomas Meyer (editors): *Proceedings of the 19th International Conference on Principles of Knowledge Representation and Reasoning, KR 2022, Haifa, Israel. July 31 - August 5, 2022*, 2022. https://proceedings.kr.org/2022/16/.

[47] Dov M. Gabbay and Hans Jürgen Ohlbach: *Quantifier Elimination in Second-Order Predicate Logic*. In Bernhard Nebel, Charles Rich, and William R. Swartout (editors): *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning (KR'92). Cambridge, MA, USA, October 25-29, 1992*, pages 425–435. Morgan Kaufmann, 1992.

[48] Harald Ganzinger: *Relating Semantic and Proof-Theoretic Concepts for Polynominal Time Decidability of Uniform Word Problems*. In *16th Annual IEEE Symposium on Logic in Computer Science, Boston, Massachusetts, USA, June 16-19, 2001, Proceedings*, pages 81–90. IEEE Computer Society, 2001. https://doi.org/10.1109/LICS.2001.932485.

[49] Harald Ganzinger, Viorica Sofronie-Stokkermans, and Uwe Waldmann: *Modular Proof Systems for Partial Functions with Weak Equality*. In David A. Basin and Michaël Rusinowitch (editors): *Automated Reasoning - Second International Joint Conference, IJCAR 2004, Cork, Ireland, July 4-8, 2004, Proceedings*, volume 3097 of *Lecture Notes in Computer Science*, pages 168–182. Springer, 2004.

[50] Harald Ganzinger, Viorica Sofronie-Stokkermans, and Uwe Waldmann: *Modular proof systems for partial functions with Evans equality.* Inf. Comput., 204(10):1453–1492, 2006.

[51] Silvio Ghilardi: *Model-Theoretic Methods in Combined Constraint Satisfiability.* J. Autom. Reason., 33(3-4):221–249, 2004. https://doi.org/10.1007/s10817-004-6241-5.

[52] Silvio Ghilardi, Alessandro Gianola, and Deepak Kapur: *Uniform Interpolants in EUF: Algorithms using DAG-representations.* Log. Methods Comput. Sci., 18(2), 2022. https://doi.org/10.46298/lmcs-18(2:2)2022.

[53] Silvio Ghilardi and Silvio Ranise: *Backward Reachability of Array-based Systems by SMT solving: Termination and Invariant Synthesis.* Logical Methods in Computer Science, 6(4), 2010.

[54] Robert Givan and David A. McAllester: *New Results on Local Inference Relations.* In Bernhard Nebel, Charles Rich, and William R. Swartout (editors): *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning (KR'92). Cambridge, MA, USA, October 25-29, 1992*, pages 403–412. Morgan Kaufmann, 1992.

[55] Robert Givan and David A. McAllester: *Polynomial-time Computation via Local Inference Relations.* CoRR, cs.LO/0007020, 2000. https://arxiv.org/abs/cs/0007020.

[56] Bernhard Gleiss, Laura Kovács, and Simon Robillard: *Loop Analysis by Quantification over Iterations.* In Gilles Barthe, Geoff Sutcliffe, and Margus Veanes (editors): *LPAR-22. 22nd International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Awassa, Ethiopia, 16-21 November 2018*, volume 57 of *EPiC Series in Computing*, pages 381–399. EasyChair, 2018.

[57] Arie Gurfinkel, Sharon Shoham, and Yakir Vizel: *Quantifiers on Demand.* In Shuvendu K. Lahiri and Chao Wang (editors): *Automated Technology for Verification and Analysis - 16th International Symposium, ATVA 2018, Los Angeles, CA, USA, October 7-10, 2018, Proceedings*, volume 11138 of *Lecture Notes in Computer Science*, pages 248–266. Springer, 2018.

[58] Reiko Heckel, Leen Lambers, and Maryam Ghaffari Saadat: *Analysis of Graph Transformation Systems: Native vs Translation-based Techniques.* In Rachid Echahed and Detlef Plump (editors): *Proceedings Tenth International Workshop on Graph Computation Models, GCM@STAF 2019, Eindhoven, The Netherlands, 17th July 2019*, volume 309 of *EPTCS*, pages 1–22, 2019. https://doi.org/10.4204/EPTCS.309.1.

[59] Lawrence J. Henschen and Larry Wos: *Unit Refutations and Horn Sets.* J. ACM, 21(4):590–605, 1974. https://doi.org/10.1145/321850.321857.

[60] Krystof Hoder, Laura Kovács, and Andrei Voronkov: *Interpolation and Symbol Elimination in Vampire.* In Jürgen Giesl and Reiner Hähnle (editors): *Automated Reasoning, 5th International Joint Conference, IJCAR 2010, Edinburgh, UK, July 16-19, 2010. Proceedings*, volume 6173 of *Lecture Notes in Computer Science*, pages 188–195. Springer, 2010.

[61] Wilfrid Hodges: *Model theory*, volume 42 of *Encyclopedia of mathematics and its applications.* Cambridge University Press, 1993, ISBN 978-0-521-30442-9.

[62] Matthias Horbach and Viorica Sofronie-Stokkermans: *Obtaining Finite Local Theory Axiomatizations via Saturation.* In Pascal Fontaine, Christophe Ringeissen, and Renate A. Schmidt (editors): *Frontiers of Combining Systems - 9th International Symposium, FroCoS 2013, Nancy, France, September 18-20, 2013. Proceedings*, volume 8152 of *Lecture Notes in Computer Science*, pages 198–213. Springer, 2013.

[63] Carsten Ihlemann, Swen Jacobs, and Viorica Sofronie-Stokkermans: *On Local Reasoning in Verification.* In C. R. Ramakrishnan and Jakob Rehof (editors): *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, volume 4963 of *Lecture Notes in Computer Science*, pages 265–281. Springer, 2008.

[64] Carsten Ihlemann and Viorica Sofronie-Stokkermans: *On Hierarchical Reasoning in Combinations of Theories.* In Jürgen Giesl and Reiner Hähnle (editors): *Automated Reasoning, 5th International Joint Conference, IJCAR 2010, Edinburgh, UK, July 16-19, 2010. Proceedings*, volume 6173 of *Lecture Notes in Computer Science*, pages 30–45. Springer, 2010.

[65] Carsten Ihlemann and Viorica Sofronie-Stokkermans: *System Description: H-PILoT (Version 1.9).* CoRR, abs/1009.0673, 2010. `http://arxiv.org/abs/1009.0673`.

[66] Swen Jacobs and Viktor Kuncak: *Towards Complete Reasoning about Axiomatic Specifications.* In Ranjit Jhala and David A. Schmidt (editors): *Verification, Model Checking, and Abstract Interpretation - 12th International Conference, VMCAI 2011, Austin, TX, USA, January 23-25, 2011. Proceedings*, volume 6538 of *Lecture Notes in Computer Science*, pages 278–293. Springer, 2011.

[67] Deepak Kapur: *A Quantifier-Elimination Based Heuristic for Automatically Generating Inductive Assertions for Programs.* J. Systems Science & Complexity, 19(3):307–330, 2006.

[68] Deepak Kapur, Rupak Majumdar, and Calogero G. Zarba: *Interpolation for data structures.* In Michal Young and Premkumar T. Devanbu (editors): *Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2006, Portland, Oregon, USA, November 5-11, 2006*, pages 105–116. ACM, 2006.

[69] Aleksandr Karbyshev, Nikolaj Bjørner, Shachar Itzhaky, Noam Rinetzky, and Sharon Shoham: *Property-Directed Inference of Universal Invariants or Proving Their Absence.* J. ACM, 64(1):7:1–7:33, 2017.

[70] Brad Karp and H. T. Kung: *GPSR: Greedy Perimeter Stateless Routing for Wireless Networks.* In *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking*, pages 243–254, 2000.

[71] Boris Konev, Dirk Walther, and Frank Wolter: *Forgetting and Uniform Interpolation in Large-Scale Description Logic Terminologies.* In Craig Boutilier (editor): *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, pages 830–835, 2009. `http://ijcai.org/Proceedings/09/Papers/142.pdf`.

[72] Laura Kovács and Andrei Voronkov: *Finding Loop Invariants for Programs over Arrays Using a Theorem Prover.* In Marsha Chechik and Martin Wirsing (editors): *Fundamental Approaches to Software Engineering, 12th International Conference, FASE 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings*, volume 5503 of *Lecture Notes in Computer Science*, pages 470–485. Springer, 2009.

[73] Laura Kovács and Andrei Voronkov: *Interpolation and Symbol Elimination.* In Renate A. Schmidt (editor): *Automated Deduction - CADE-22, 22nd International Conference on Automated Deduction, Montreal, Canada, August 2-7, 2009. Proceedings*, volume 5663 of *Lecture Notes in Computer Science*, pages 199–213. Springer, 2009.

[74] Fabian Kuhn, Roger Wattenhofer, and Aaron Zollinger: *Ad hoc networks beyond unit disk graphs.* Wireless Networks, 14(5):715–729, Oct 2008, ISSN 1022-0038.

[75] Carsten Lutz, Inanç Seylan, and Frank Wolter: *An Automata-Theoretic Approach to Uniform Interpolation and Approximation in the Description Logic $\mathcal{EL}$.* In Gerhard Brewka, Thomas Eiter, and Sheila A. McIlraith (editors): *Principles of Knowledge Representation and Reasoning: Proceedings of the Thirteenth International Conference, KR 2012, Rome, Italy, June 10-14, 2012.* AAAI Press, 2012. http://www.aaai.org/ocs/index.php/KR/KR12/paper/view/4511.

[76] Carsten Lutz, Inanç Seylan, and Frank Wolter: *The Data Complexity of Ontology-Mediated Queries with Closed Predicates.* Log. Methods Comput. Sci., 15(3), 2019. https://doi.org/10.23638/LMCS-15(3:23)2019.

[77] Philipp Marohn: *Verifikation und Constraint-Generierung in parametrischen Systemen.* Bachelor's thesis, Universität Koblenz-Landau, Deutschland, 2021.

[78] Philipp Marohn and Viorica Sofronie-Stokkermans: *SEH-PILoT: A System for Property-Directed Symbol Elimination - Work in Progress (Short Paper).* In Renate A. Schmidt, Christoph Wernhard, and Yizheng Zhao (editors): *Proceedings of the Second Workshop on Second-Order Quantifier Elimination and Related Topics (SOQE 2021) associated with the 18th International Conference on Principles of Knowledge Representation and Reasoning (KR 2021), Online Event, November 4, 2021*, volume 3009 of *CEUR Workshop Proceedings*, pages 75–82. CEUR-WS.org, 2021. http://ceur-ws.org/Vol-3009/short2.pdf.

[79] Leonardo Mendonça de Moura and Nikolaj S. Bjørner: *Z3: An Efficient SMT Solver.* In C. R. Ramakrishnan and Jakob Rehof (editors): *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer, 2008. https://doi.org/10.1007/978-3-540-78800-3_24.

[80] Oded Padon, Neil Immerman, Sharon Shoham, Aleksandr Karbyshev, and Mooly Sagiv: *Decidability of inferring inductive invariants.* In Rastislav Bodík and Rupak Majumdar (editors): *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, pages 217–231. ACM, 2016.

[81] Dennis Peuter and Viorica Sofronie-Stokkermans: *On Inductive Verification and Synthesis.* In Christoph Benzmüller, Xavier Parent, and Alexander Steen (ed-

itors): *Selected Student Contributions and Workshop Papers of LuxLogAI 2018*, volume 10 of *Kalpa Publications in Computing*, pages 1–8. EasyChair, 2018. http://www.easychair.org/publications/paper/C1mW.

[82] Dennis Peuter and Viorica Sofronie-Stokkermans: *On Invariant Synthesis for Parametric Systems.* In Pascal Fontaine (editor): *Automated Deduction - CADE 27 - 27th International Conference on Automated Deduction, Natal, Brazil, August 27-30, 2019, Proceedings*, volume 11716 of *Lecture Notes in Computer Science*, pages 385–405. Springer, 2019. https://doi.org/10.1007/978-3-030-29436-6_23.

[83] Dennis Peuter and Viorica Sofronie-Stokkermans: *Finding High-Level Explanations for Subsumption w.r.t. Combinations of CBoxes in $\mathcal{EL}$ and $\mathcal{EL}^+$.* In Stefan Borgwardt and Thomas Meyer (editors): *Proceedings of the 33rd International Workshop on Description Logics (DL 2020) co-located with the 17th International Conference on Principles of Knowledge Representation and Reasoning (KR 2020), Online Event [Rhodes, Greece], September 12th to 14th, 2020*, volume 2663 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2020. http://ceur-ws.org/Vol-2663/paper-18.pdf.

[84] Dennis Peuter and Viorica Sofronie-Stokkermans: *Symbol Elimination and Applications to Parametric Entailment Problems.* In Boris Konev and Giles Reger (editors): *Frontiers of Combining Systems - 13th International Symposium, FroCoS 2021, Birmingham, UK, September 8-10, 2021, Proceedings*, volume 12941 of *Lecture Notes in Computer Science*, pages 43–62. Springer, 2021. https://doi.org/10.1007/978-3-030-86205-3_3.

[85] Dennis Peuter and Viorica Sofronie-Stokkermans: *Symbol Elimination and Applications to Parametric Entailment Problems (Abstract).* In Renate A. Schmidt, Christoph Wernhard, and Yizheng Zhao (editors): *Proceedings of the Second Workshop on Second-Order Quantifier Elimination and Related Topics (SOQE 2021) associated with the 18th International Conference on Principles of Knowledge Representation and Reasoning (KR 2021), Online Event, November 4, 2021*, volume 3009 of *CEUR Workshop Proceedings*, pages 83–91. CEUR-WS.org, 2021. https://ceur-ws.org/Vol-3009/abstract3.pdf.

[86] Dennis Peuter, Viorica Sofronie-Stokkermans, and Sebastian Thunert: *On P-Interpolation in Local Theory Extensions and Applications to the Study of Interpolation in the Description Logics $\mathcal{EL}$, $\mathcal{EL}^+$.* In Brigitte Pientka and Cesare Tinelli (editors): *Automated Deduction - CADE 29 - 29th International Conference on Automated Deduction, Rome, Italy, July 1-4, 2023, Proceedings*, volume 14132 of *Lecture Notes in Computer Science*, pages 419–437. Springer, 2023. https://doi.org/10.1007/978-3-031-38499-8_24.

[87] Andrey Rybalchenko and Viorica Sofronie-Stokkermans: *Constraint solving for interpolation.* J. Symb. Comput., 45(11):1212–1233, 2010. https://doi.org/10.1016/j.jsc.2010.06.005.

[88] Stefan Schlobach and Ronald Cornet: *Non-Standard Reasoning Services for the Debugging of Description Logic Terminologies.* In Georg Gottlob and Toby Walsh (editors): *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003*, pages 355–362. Morgan Kaufmann, 2003. http://ijcai.org/Proceedings/03/Papers/053.pdf.

[89] Viorica Sofronie-Stokkermans: *Decision Procedures for Verification, Slides from January 9, 2023, Pages 30-31.* Lecture. Universität Koblenz.

[90] Viorica Sofronie-Stokkermans: *Hierarchic Reasoning in Local Theory Extensions.* In Robert Nieuwenhuis (editor): *Automated Deduction - CADE-20, 20th International Conference on Automated Deduction, Tallinn, Estonia, July 22-27, 2005, Proceedings*, volume 3632 of *Lecture Notes in Computer Science*, pages 219–234. Springer, 2005.

[91] Viorica Sofronie-Stokkermans: *Interpolation in Local Theory Extensions.* In Ulrich Furbach and Natarajan Shankar (editors): *Automated Reasoning, Third International Joint Conference, IJCAR 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*, volume 4130 of *Lecture Notes in Computer Science*, pages 235–250. Springer, 2006.

[92] Viorica Sofronie-Stokkermans: *Automated theorem proving by resolution in non-classical logics.* Ann. Math. Artif. Intell., 49(1-4):221–252, 2007. `https://doi.org/10.1007/s10472-007-9051-8`.

[93] Viorica Sofronie-Stokkermans: *Interpolation in Local Theory Extensions.* Logical Methods in Computer Science, 4(4), 2008.

[94] Viorica Sofronie-Stokkermans: *Locality and Subsumption Testing in $\mathcal{EL}$ and Some of its Extensions.* In Franz Baader, Carsten Lutz, and Boris Motik (editors): *Proceedings of the 21st International Workshop on Description Logics (DL2008), Dresden, Germany, May 13-16, 2008*, CEUR Workshop Proceedings 353. CEUR-WS.org, 2008. `http://ceur-ws.org/Vol-353/Sofronie-Stokkermans.pdf`.

[95] Viorica Sofronie-Stokkermans: *Locality and subsumption testing in $\mathcal{EL}$ and some of its extensions.* In Carlos Areces and Robert Goldblatt (editors): *Advances in Modal Logic 7, papers from the seventh conference on "Advances in Modal Logic"*, pages 315–339. College Publications, 2008. `http://www.aiml.net/volumes/volume7/Sofronie-Stokkermans.pdf`.

[96] Viorica Sofronie-Stokkermans: *Hierarchical Reasoning for the Verification of Parametric Systems.* In Jürgen Giesl and Reiner Hähnle (editors): *Automated Reasoning, 5th International Joint Conference, IJCAR 2010, Edinburgh, UK, July 16-19, 2010. Proceedings*, volume 6173 of *Lecture Notes in Computer Science*, pages 171–187. Springer, 2010.

[97] Viorica Sofronie-Stokkermans: *Hierarchical Reasoning and Model Generation for the Verification of Parametric Hybrid Systems.* In Maria Paola Bonacina (editor): *Automated Deduction - CADE-24 - 24th International Conference on Automated Deduction, Lake Placid, NY, USA, June 9-14, 2013. Proceedings*, volume 7898 of *Lecture Notes in Computer Science*, pages 360–376. Springer, 2013.

[98] Viorica Sofronie-Stokkermans: *On Interpolation and Symbol Elimination in Theory Extensions.* In Nicola Olivetti and Ashish Tiwari (editors): *Automated Reasoning - 8th International Joint Conference, IJCAR 2016, Coimbra, Portugal, June 27 - July 2, 2016, Proceedings*, volume 9706 of *Lecture Notes in Computer Science*, pages 273–289. Springer, 2016.

[99] Viorica Sofronie-Stokkermans: *Representation Theorems and Locality for Subsumption Testing and Interpolation in the Description Logics $\mathcal{EL}$, $\mathcal{EL}^+$ and their Extensions*

with n*-ary Roles and Numerical Domains*. Fundam. Informaticae, 156(3-4):361–411, 2017. https://doi.org/10.3233/FI-2017-1612.

[100] Viorica Sofronie-Stokkermans: *On Interpolation and Symbol Elimination in Theory Extensions*. Logical Methods in Computer Science, Volume 14, Issue 3, September 2018. https://lmcs.episciences.org/4848.

[101] Viorica Sofronie-Stokkermans and Carsten Ihlemann: *Automated Reasoning in Some Local Extensions of Ordered Structures*. In *37th International Symposium on Multiple-Valued Logic, ISMVL 2007, 13-16 May 2007, Oslo, Norway*, page 1. IEEE Computer Society, 2007. https://doi.org/10.1109/ISMVL.2007.10.

[102] Kent A. Spackman: *Normal forms for description logic expressions of clinical concepts in SNOMED RT*. In *AMIA 2001, American Medical Informatics Association Annual Symposium, Washington, DC, USA, November 3-7, 2001*. AMIA, 2001. http://knowledge.amia.org/amia-55142-a2001a-1.597057/t-001-1.599654/f-001-1.599655/a-126-1.599761/a-127-1.599758.

[103] Kent A. Spackman, Keith E. Campbell, and Roger A. Côté: *SNOMED RT: a reference terminology for health care*. In *AMIA 1997, American Medical Informatics Association Annual Symposium, Nashville, TN, USA, October 25-29, 1997*. AMIA, 1997. http://knowledge.amia.org/amia-55142-a1997a-1.585351/t-001-1.587519/f-001-1.587520/a-127-1.587635/a-128-1.587632.

[104] Thomas Sturm: *Selected Topics in Automated Reasoning*. Lecture, Juni 2011. Max-Planck-Institut für Informatik, Saarbrücken.

[105] Thomas Sturm: *Thirty Years of Virtual Substitution: Foundations, Techniques, Applications*. In Manuel Kauers, Alexey Ovchinnikov, and Éric Schost (editors): *Proceedings of the 2018 ACM on International Symposium on Symbolic and Algebraic Computation, ISSAC 2018, New York, NY, USA, July 16-19, 2018*, pages 11–16. ACM, 2018. https://doi.org/10.1145/3208976.3209030.

[106] Boontawee Suntisrivaraporn: *Polynomial time reasoning support for design and maintenance of large-scale biomedical ontologies*. PhD thesis, Dresden University of Technology, Germany, 2009. http://hsss.slub-dresden.de/deds-access/hsss.urlmapping.MappingServlet?id=1233830966436-5928.

[107] Alfred Tarski: *A Decision Method for Elementary Algebra and Geometry*. RAND Corporation, 1948.

[108] Sebastian Thunert: *Automatization of Computing High-Level Explanations for Subsumptions in $\mathcal{EL}$ and $\mathcal{EL}^+$*. Master's thesis, Universität Koblenz-Landau, Deutschland, 2021.

[109] Marco Voigt: *Decidable fragments of first-order logic and of first-order linear arithmetic with uninterpreted predicates*. PhD thesis, Saarland University, Saarbrücken, Germany, 2019. https://publikationen.sulb.uni-saarland.de/handle/20.500.11880/27767.

[110] Christoph Weidenbach, Dilyana Dimova, Arnaud Fietzke, Rohit Kumar, Martin Suda, and Patrick Wischnewski: *SPASS Version 3.5*. In Renate A. Schmidt (editor): *Automated Deduction - CADE-22, 22nd International Conference on Automated Deduction, Montreal, Canada, August 2-7, 2009. Proceedings*, volume 5663

of *Lecture Notes in Computer Science*, pages 140–145. Springer, 2009. https://doi.org/10.1007/978-3-642-02959-2_10.

[111] Volker Weispfenning: *The Complexity of Linear Problems in Fields.* Journal of Symbolic Computation, (5):3–27, 1988.

[112] Volker Weispfenning: *Quantifier Elimination for Real Algebra - the Quadratic Case and Beyond.* Appl. Algebra Eng. Commun. Comput., 8(2):85–101, 1997. https://doi.org/10.1007/s002000050055.

[113] Greta Yorsh and Madanlal Musuvathi: *A Combination Method for Generating Interpolants.* In Robert Nieuwenhuis (editor): *Automated Deduction - CADE-20, 20th International Conference on Automated Deduction, Tallinn, Estonia, July 22-27, 2005, Proceedings*, volume 3632 of *Lecture Notes in Computer Science*, pages 353–368. Springer, 2005.

Curriculum Vitae

Personal Data

Name:	Dennis Peuter
Address:	Universität Koblenz, Universitätsstraße 1, 56070 Koblenz
E-Mail:	dpeuter@uni-koblenz.de
Date of Birth:	20.07.1987
Citizenship:	German
Homepage:	https://userpages.uni-koblenz.de/∼dpeuter/

Academic Positions

since Feb 2017	Research and teaching assistant at Universität Koblenz, Institute for Computer Science, working group „Formal Methods and Theoretical Computer Science“ led by Prof. Dr. Viorica Sofronie-Stokkermans

Education

2009 – 2016	Bachelor and Master of Education with subjects computer science and mathematics, Universität Koblenz-Landau, Koblenz
1998 – 2007	Abitur, Integrierte Gesamtschule Kastellaun

Teaching

since SS 2017	Instructor for Exercises in "Grundlagen der theoretischen Informatik"
since WS 2017/2018	Instructor for Exercises in "Vertiefung Theoretische Informatik"
since SS 2018	Assistant for Seminar "Decision Procedures and Applications"

- D. Peuter and V. Sofronie-Stokkermans: On Inductive Verification and Synthesis. In Selected Student Contributions and Workshop Papers of LuxLogAI 2018, volume 10 of Kalpa Publications in Computing, pages 1-8. EasyChair, 2018.

- D. Peuter and V. Sofronie-Stokkermans: On Invariant Synthesis for Parametric Systems. In Proceedings of the 27th International Conference on Automated Deduction (CADE-27), volume 11716 of Lecture Notes in Computer Science, pages 385-405. Springer, 2019.

- D. Peuter and V. Sofronie-Stokkermans: Finding High-Level Explanations for Subsumption w.r.t. Combinations of CBoxes in $\mathcal{EL}$ and $\mathcal{EL}^+$. In Proceedings of the 33rd International Workshop on Description Logic (DL 2020) co-located with the 17th International Conference on Principles of Knowledge Representation and Reasoning (KR 2020), volume 2663 of CEUR Workshop Proceedings. CEUR-WS.org, 2020.

- L. Böltz, H. Frey, D. Peuter, and V. Sofronie-Stokkermans: On Testing Containedness Between Geometric Graph Classes using Second-order Quantifier Elimination and Hierarchical Reasoning (Short Paper). In: Proceedings of the Second Workshop on Second-Order Quantifier Elimination and Related Topics (SOQE 2021) associated with the 18th International Conference on Principles of Knowledge Representation and Reasoning (KR 2021), volume 3009 of CEUR Workshop Proceedings, pages 37-45. CEUR-WS.org, 2021.

- D. Peuter and V. Sofronie-Stokkermans: Symbol Elimination and Applications to Parametric Entailment Problems (Abstract). In Proceedings of the Second Workshop on Second-Order Quantifier Elimination and Related Topics (SOQE 2021) associated with the 18th International Conference on Principles of Knowledge Representation and Reasoning (KR 2021), 2021, volume 3009 of CEUR Workshop Proceedings, pages 83-91. CEUR-WS.org, 2021.

- D. Peuter and V. Sofronie-Stokkermans: Symbol Elimination and Applications to Parametric Entailment Problems. In Proceedings of the 13th International Symposium on Frontiers of Combining Systems (FroCoS 2021), volume 12941 of Lecture Notes in Computer Science, pages 43-62. Springer, 2021.

- D. Peuter, V. Sofronie-Stokkermans, and S. Thunert: On P-Interpolation in Local Theory Extensions and Applications to the Study of Interpolation in the Description Logics $\mathcal{EL}$, $\mathcal{EL}^+$. In: Proceedings of the 29th International Conference on Automated Deduction (CADE-29), volume 14132 of Lecture Notes in Computer Science, pages 419-437. Springer, 2023.