Zeitbewertete Prädikat/Transitions-Netze zur Modellierung, Simulation und Analyse sicherheitskritischer Echtzeitsysteme

Dissertation

Katharina Hupf

Campus Koblenz
Fachbereich 4: Informatik
Institut Softwaretechnik

03. November 2008

für meinen Vater

Summary

Within this thesis *time evaluated predicate/transition nets (t-pr/t-nets)* have been developed for the purpose to model, simulate and verify complex real-time systems. Therefore, t-pr/t-nets integrate concepts to model timing constraints and can be analysed by the means of structural analysis such as the calculation of s- and t-invariants as well as the identification of traps and co-traps. The applicability of t-pr/t-nets to model, simulate and verify complex systems in the domain of safety-critical real-time systems is proven by the *Earliest-Deadline-First-Protocol (EDF)* and the *Priority-Inheritance-Protocol (PIP)*. Therefore, the EDF and PIP are modeled by means of t-pr/t-nets. The resulting t-pr/t-nets are verified using structural analysis methods. Due to the enormous complexity and the applicability of structural analysis methods for the verification of the EDF and PIP, it can be shown that t-pr/t-nets are appropriate to model, simulate and verify complex systems in the field of safety-critical real-time systems.

Keywords: Petri nets, time Petri nets, real-time systems, modelling, simulation, verification, s-invariants, t-invariants, traps, co-traps

Zusammenfassung

Im Rahmen dieser Arbeit wurden die *zeitbewerteten Prädikat/Transitions-Netze (Z-Pr/T-Netze)* zur Modellierung, Simulation und Verifikation sicherheitskritischer Echtzeitsysteme entwickelt. Z-Pr/T-Netze integrieren Konzepte zur Modellierung temporärer Zusammenhänge und sind darüber hinaus mittels der Berechnung von S- und T-Invarianten sowie der Identifikation von Traps und Co-Traps strukturell analysierbar. Die Eignung von Z-Pr/T-Netzen zur Modellierung, Simulation und Verifikation komplexer Systeme aus dem Anwendungsbereich sicherheitskritischer Echtzeitsysteme wird anhand des *Earliest-Deadline-First-Protokolls (EDF)* und des *Priority-Inheritance-Protokolls (PIP)* belegt. Es erfolgt daher eine Modellierung des EDF und PIP mittels Z-Pr/T-Netzen sowie eine Verifikation für das EDF und PIP basierend auf der strukturellen Analyse der korrespondierenden Z-Pr/T-Netze. Die Anwendbarkeit struktureller Analyseverfahren zur Verifikation des EDF und PIP in Verbindung mit der nicht zu unterschätzenden Komplexität eben dieser belegen die Anwendbarkeit von Z-Pr/T-Netzen zur Modellierung, Simulation und Verifikation komplexer Systeme aus dem Anwendungsbereich sicherheitskritischer Echtzeitsysteme.

Stichwörter: Petrinetze, Zeit-Petrinetze, Echtzeitsysteme, Modellierung, Simulation, Verifikation, S-Invarianten, T-Invarianten, Traps, Co-Traps

Vorwort

Die vorliegende Arbeit entstand in der Arbeitgruppe von Prof. Dr. Kurt Lautenbach an der Universität Koblenz-Landau in dem Zeitraum von Dezember 2004 bis November 2008. Dabei erhielt ich von Juni 2005 bis Mai 2007 ein Promotionsstipendium der Landesgraduiertenförderung. Diese finanzielle Unterstützung war eine große Hilfe während der Promotion. Das Gelingen dieser Arbeit ist jedoch in erster Linie auf die Unterstützung und Hilfe von Personen zurück zu führen, denen ich an dieser Stellen danken möchte.
Zu allererst möchte ich mich bei Prof. Dr. Kurt Lautenbach für die hervorragende Betreuung dieser Arbeit bedanken. Mit ihm stand mir jederzeit ein Ansprechpartner zur Seite, der mich bei der Erstellung dieser Arbeit und allen dabei auftauchenden Problemen unterstützt hat. Die zahlreichen Diskussionen und Anregungen bilden den Grundstein dieser Arbeit.
Des Weiteren möchte ich Prof. Dr. Dieter Zöbel für die Bereitschaft zur Erstellung des Zweitgutachtens sowie für die vielen wertvollen Kommentare danken, die in hohem Maße zur Abrundung dieser Arbeit beigetragen haben.
Mein besonderer Dank gilt Dr. Stephan Philippi, welcher dieses Promotionsvorhaben auf den Weg gebracht und von Anfang an unterstützt hat.
Dank gilt auch allen Mitgliedern der Arbeitsgruppe „Petrinetze“ für die harmonische und gute Zusammenarbeit. Insbesondere Alexander Pinl möchte ich für seine Bereitschaft danken, mir mit Rat und Tat zur Seite zu stehen.
Christina Göhring danke ich dafür, dass sie ihre knapp bemessene Zeit zum Korrekturlesen dieser Arbeit geopfert hat.
Von ganzem Herzen danke ich meinem Freund Holger. Seine Unterstützung, seine motivierenden Worte und sein Rückhalt haben maßgeblich zum Gelingen dieser Arbeit beigetragen.
Zu guter Letzt möchte ich mich bei meiner Familie bedanken, ohne deren bedingungslose Unterstützung und Liebe diese Arbeit nicht möglich gewesen wäre. Danke!

Koblenz, 03.November 2008 Katharina Hupf

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Computersysteme werden immer mehr in Bereichen eingesetzt, in denen ein Versagen nicht nur wirtschaftliche Verluste bedeutet, sondern auch Menschenleben gefährden kann, wie z.B. bei Fahrassistenzsystemen in modernen Fahrzeugen oder auch bei computergestützten Operationsassistenzsystemen im Bereich der Medizin. Als Konsequenz erlangt die Verifikation von Software wachsende Bedeutung. Die Verifikation von Software bezeichnet den mathematischen Beweis, dass ein Programm der vorgegebenen Spezifikation entspricht.

Gerade in Hinblick auf sicherheitskritische Echtzeitsysteme spielt die Verifikation eine bedeutende Rolle. Dabei seien mit *Echtzeitsystemen* diejenigen Computersysteme bezeichnet, welche die Berechnung von Ergebnissen innerhalb gegebener Fristen garantieren [4],[6],[48]. Die Verifikation von Echtzeitsystemen beinhaltet somit einen temporären Aspekt. Für ein Echtzeitsystem muss daher sowohl die Korrektheit der Ergebnisse als auch die fristgerechte Bereitstellung eben dieser verifiziert werden.

Der Einsatz von Petrinetzen zur Verifikation im Anwendungsbereich der Echtzeitsysteme ist aus vielerlei Gründen naheliegend. *Petrinetze* [27] sind eine Modellierungssprache, welche sich sowohl in der Wissenschaft als auch in der Industrie als Mittel zur Simulation, Analyse und Diagnose von Modellen etabliert haben. Ein nicht zu unterschätzender Vorteil von Petrinetzen ist die graphische Visualisierbarkeit und damit einhergehend eine bessere Verständlichkeit der Modelle auch für Experten aus anderen Anwendungsdomänen. Dies ist ein Vorzug, der Petrinetze deutlich von anderen formalen Sprachen abgrenzt. So verfügt beispielsweise ein Petrinetz-Modell eine höhere Verständlichkeit als ein Modell mittels Gleichungssystemen auf Basis von Differentialgleichungen. Des Weiteren erlaubt die formale Basis der Petrinetze eine strukturelle Analyse der Modelle. Die strukturelle Analyse von Petrinetzen bezeichnet dabei eine Herleitung von Netzeigenschaften ausschließlich anhand der Struktur des zu analysierenden Petrinetzes. Die mittels struktureller Analyse extrahierten Strukturkomponenten können zur Verifikation von Systemeigenschaften genutzt werden. Insbesondere die unmittelbare Anwendbarkeit struktureller Analysemethoden qualifizieren Petrinetze zur Verifikation von Systemeigenschaften sicherheitskritischer Echtzeitsysteme.

Ein *Petrinetz* ist ein gerichteter bipartiter Graph, bestehend aus Stellen und Transitionen, welche über Kanten miteinander verbunden sind. Dabei dürfen immer nur Knoten unterschiedlichen Typs miteinander verbunden sein. Die Stellen können mit sogenannten Marken markiert sein, welche den (verteilten) Systemzustand repräsentieren. Die Dynamik des Netzes wird durch die Transitionen modelliert. Dabei variieren die verschiedenen Netzklassen in der Beschriftung der einzelnen Elemente des Netzes. Unabhängig davon jedoch werden Stellen graphisch durch Kreise, Transitionen durch Quadrate, Kanten durch Pfeile und Marken durch kleine ausgefüllte Kreise dargestellt [1],[27],[28],[34],[36],[41].

Durch Einführung individueller Marken sind aus den von C.A. Petri im Jahr 1962 entwickelten Petrinetzen [27] die sogenannten *höheren Petrinetze* hervor gegangen. Darunter zählen die *Prädikat/Transitions-Netze* [7],[8],[9] und die *Coloured Petri Nets* [12],[13] zu ihren bekanntesten Vertretern. Sie erlauben Marken mit beliebig komplexen Datenstrukturen und stellen im Gegensatz zu den einfachen Petrinetzen in vielerlei Hinsicht eine Verbesserung der Handhabbarkeit und Modellierungsmächtigkeit dar.

Um einen Einsatz von Petrinetzen zur Modellierung, Simulation und Verifikation sicherheitskritischer Echtzeitsysteme zu ermöglichen, muss ein Konzept zur Repräsentation von Zeit in Petrinetze integriert werden. Es existiert bereits eine Vielzahl von Ansätzen, Konzepte zur Modellierung von Zeit in Petrinetze zu integrieren. Der Mehrheit dieser Ansätze liegt jedoch ein Zeitmodell zu Grunde, welches Zeit als eine monoton steigende Funktion abbildet z.B. [14],[21],[30],[31], [42],[44],[45],[46]. Dies bedingt jedoch, dass der Zustandsraum der Modelle unendlich wird und eine strukturelle Analyse der Modelle somit nicht möglich ist. Zahlreiche Ansätze integrieren zwar eine Verifikation auf Basis einer Analyse des Erreichbarkeitsgraphen, dies ist jedoch bei Anwendungen realem Komplexitätsumfangs auf Grund der Explosion des Zustandsraumes nicht praktikabel. Somit bleibt bei einer Vielzahl der zeiterweiterten Ansätze das Potenzial von Petrinetzen ungenutzt, die Modelle unmittelbar strukturell analysieren und verifizieren zu können. Gerade in Hinblick auf die fatalen Konsequenzen, welche fehlerhafte Softwaresysteme mit sich ziehen können, rückt jedoch eine praktikable Lösung für die Verifikation von Echtzeitsystemen immer mehr in den Mittelpunkt des Interesses.

Im Rahmen dieser Arbeit wurde daher die Klasse der *zeitbewerteten Prädikat/Transitions-Netze (Z-Pr/T-Netze)* entwickelt. Z-Pr/T-Netze integrieren Konzepte zur Modellierung temporärer Zusammenhänge und erlauben somit die Modellierung und Simulation von zeitbewerteten Systemen. Die mittels Z-Pr/T-Netzen modellierten Systeme sind darüber hinaus strukturell analysierbar und können durch die unmittelbare strukturelle Analyse der gegebenen Modelle

zur Verifikation von Systemeigenschaften genutzt werden. Z-Pr/T-Netze erlauben somit die Modellierung, Simulation und Verifikation zeitbehafteter Systeme und stellen somit einen vielversprechenden Ansatz zur Modellierung, Simulation und Verifikation im Anwendungsbereich sicherheitskritischer Echtzeitsysteme dar.

Die nachfolgende Arbeit gliedert sich wie folgt: Zunächst werden in Kapitel 2 Konzepte zur Modellierung von Zeit in Petrinetzen vorgestellt sowie die bekanntesten Vertreter zeiterweiterter Petrinetze insbesondere in Hinblick auf die Anwendbarkeit struktureller Analyseverfahren eingehend beleuchtet.

Anschließend erfolgt in Kapitel 3 eine Einführung in die Klasse der *summenbeschrifteten Prädikat/Transitions-Netze (S-Pr/T-Netze)*. Die Wertebereiche der Variablen werden in S-Pr/T-Netzen durch die Anwendung der Modulo-Operationen $\oplus$ und $\ominus$ beschränkt. S-Pr/T-Netze besitzen daher einen endlichen Zustandsraum und sind mittels struktureller Analyseverfahren wie z.B. der Berechnung von S- und T-Invarianten analysierbar. Aus diesem Grund bilden S-Pr/T-Netze die Grundlage für die im Rahmen dieser Arbeit neu entwickelte Klasse der Z-Pr/T-Netze. Die Definition grundlegender Begriffe sowie eine Definition der S-Pr/T-Netze erfolgt in Kapitel 3.1. Die Dynamik in S-Pr/T-Netzen wird in Kapitel 3.2 erläutert. In Kapitel 3.3 werden die strukturellen Analysemethoden in S-Pr/T-Netzen erörtert. Dabei wird die Berechnung von S- und T-Invarianten in Kapitel 3.3.1 und die Identifikation von Traps und Co-Traps in Kapitel 3.3.2 beleuchtet.

Im Anschluss daran erfolgt in Kapitel 4 eine detaillierte Einführung der im Rahmen dieser Arbeit entwickelten Klasse der *zeitbewerteten Prädikat/Transitions-Netze (Z-Pr/T-Netze)*. Ebenso wie in S-Pr/T-Netzen werden die Wertebereiche der Variablen in Z-Pr/T-Netzen durch Anwendung der Modulo-Operationen $\oplus$ und $\ominus$ beschränkt. Z-Pr/T-Netze besitzen somit ebenfalls einen endlichen Zustandsraum und sind strukturell analysierbar. Darüber hinaus ist die semantische Interpretation der Variablen fest vorgegeben. Des Weiteren garantiert eine modifizierte Transitionsregel die korrekte Abbildung der zeitlichen Abfolge von Zustandsübergängen. Die Definition der Z-Pr/T-Netze erfolgt in Kapitel 4.1 und die Dynamik in Z-Pr/T-Netzen wird in Kapitel 4.2 erläutert. Anschließend wird die strukturelle Analyse von Z-Pr/T-Netzen in Kapitel 4.3 thematisiert. Dabei wird die strukturelle Analyse von Z-Pr/T-Netzen mittels der Berechnung von S- und T-Invarianten sowie der Identifikation von Traps und Co-Traps beschrieben. Anschließend werden in Kapitel 4.4 Techniken erläutert, welche bei der Verifikation mittels struktureller Analyse von Z-Pr/T-Netzen Anwendung finden. Das Zusammenspiel dieser Techniken und der vorgestellten, strukturellen Analyseverfahren wird abschließend anhand von Beispielen verdeutlicht.

In Kapitel 5 erfolgt die Verifikation prioritätsbasierter, dynamischer Planungsverfahren mittels struktureller Analyse der zugehörigen Z-Pr/T-Netze. Dies erfolgt exemplarisch für das *Earliest-Deadline-First-Protokoll (EDF)* in Kapitel 5.1 und das *Priority-Inheritance-Protokoll (PIP)* in Kapitel 5.2. Anhand der genannten Planungsverfahren wird die Eignung von Z-Pr/T-Netzen zur Modellierung, Simulation und Analyse komplexer, zeitbehafteter Systeme nachgewiesen und die Anwendbarkeit von Z-Pr/T-Netzen zur Verifikation zeitbehafteter Systeme aus dem Anwendungsbereich der Echtzeitsysteme unter Beweis gestellt.

Abschließend werden in Kapitel 6 die Ergebnisse dieser Arbeit zusammengefasst und ein Ausblick auf weiterführende Arbeiten gegeben.

2 Zeitkonzepte für Petrinetze

Es existiert bereits eine Vielzahl von Ansätzen, Konzepte zur Modellierung von Zeit mit Petrinetzen zu verschmelzen. Ziel einer solchen Synthese von temporären Konzepten und Petrinetzen ist die Erlangung eines Formalismus zur Modellierung, Simulation und Verifikation zeitbehafteter Systeme. Dies ist insbesondere in Hinblick auf die Verifikation sicherheitskritischer Echtzeitsysteme von Interesse.

Um einen Einsatz von Petrinetzen zur Modellierung, Simulation und Verifikation sicherheitskritischer Echtzeitsysteme zu ermöglichen, muss ein Konzept zur Repräsentation von Zeit in Petrinetze integriert werden.

Dabei kann unterschieden werden, welche Elemente eines Petrinetzes um ein Konzept zur Modellierung von Zeit erweitert werden. Grundsätzlich können die *Stellen*, die *Transitionen*, die *Kanten* oder die *Marken* eines Petrinetzes um ein Konzept zur Repräsentation von Zeit angereichert werden [26].

Im Folgenden werden die bekanntesten Vertreter zeiterweiterter Petrinetze vorgestellt und ihre Anwendbarkeit hinsichtlich der Modellierung, Simulation und Verifikation sicherheitskritischer Echtzeitsysteme untersucht.

2.1 Petrinetze mit zeiterweiterten Transitionen

Neben den *Timed Petri Nets (TdPNs)* [33] zählen die *Time Petri Nets (TPNs)* [24] zu den bekanntesten Vertretern zeiterweiterter Petrinetze. In beiden Fällen erfolgt eine Assoziation temporärer Bedingungen mit den Transitionen eines Petrinetzes.

2.1.1 Time Petri Nets (TPNs)

In *Time Petri Nets (TPNs)* [24] ist jeder Transition t ein Intervall $[Eft(t), Lft(t)]$ zugeordnet, welches den frühesten und spätesten Zeitpunkt für das Feuern der Transition t spezifiziert. Ist für eine Transition t nach ihrer Aktivierung der Zeitpunkt Lft(t) erreicht, so muss diese feuern, es sei denn ihre Aktivierung wurde durch Feuern einer anderen Transition aufgehoben.

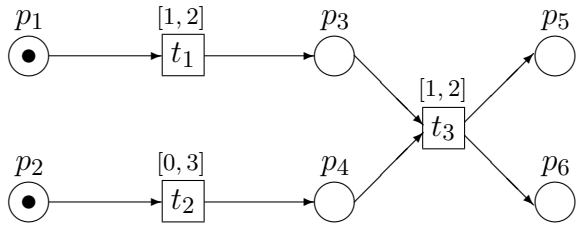


Abbildung 2.1: Beispiel für ein Time Petri Net (vgl. [26])

Definition 1 (Time Petri Net (TPN))
Ein Time Petri Net (TPN) ist ein Tupel G=(P,T,F,m^0,Eft,Lft), wenn gilt

- *P ist eine endliche Menge von Stellen*
- *T ist eine endliche Menge von Transitionen*
- *F: $(P \times T) \cup (T \times P) \rightarrow \mathbb{N}$ ist eine Flussrelation*
- *m^0 ist eine Anfangsmarkierung*
- *Eft:$T \rightarrow \mathbb{N}$ und Lft:$T \rightarrow \mathbb{N}$ sind Funktionen, welche jeder Transition $t \in T$ eine earliest firing time und eine latest firing time zuordnen, wobei gilt Eft(t) $\leq$ Lft(t) für alle $t \in T$.* □

Leveson et al. [19] schlagen eine Verifikation von TPNs auf Basis einer Analyse des zugehörigen Erreichbarkeitsgraphen vor. Ein Erreichbarkeitsgraph zeigt alle Systemzustände, die ein Petrinetz ausgehend von einer Anfangsmarkierung annehmen kann. Eine Analyse des Erreichbarkeitsgraphen ist jedoch ein Ansatz, welcher bei Anwendungen realem Komplexitätsumfangs auf Grund der Explosion des Zustandsraumes nicht praktikabel ist. Es existieren zwar Ansätze, die Komplexität der Analyse von TPNs zu reduzieren [3], dennoch bleibt dabei das Potenzial von Petrinetzen ungenutzt, die Modelle unmittelbar strukturell analysieren und verifzieren zu können.

2.1.2 Timed Petri Nets (TdPNs)

Analog zu den TPNs erfolgt in den *Timed Petri Nets (TdPNs)* [33] eine Assoziation temporärer Bedingungen mit den Transitionen eines Petrinetzes. Jedoch werden den Transitionen keine Schaltintervalle, sondern Schaltdauern zugeordnet. Eine Transition t feuert unmittelbar nach Aktivierung. Transitionen in TdPNs feuern jedoch nicht zeitlos, d.h. während des Schaltvorgangs vergeht Zeit.

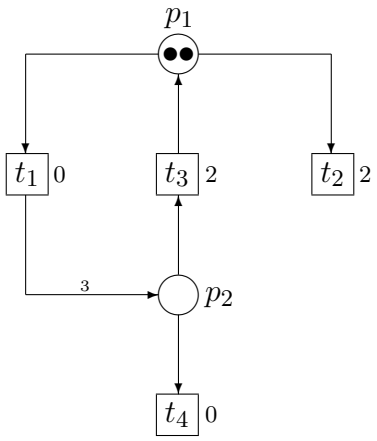


Abbildung 2.2: Beispiel für ein Timed Petri Net (vgl. [44])

Definition 2 (Timed Petri Net (TdPN))
Ein Tupel G=(P,T,F,m^0,D) ist ein Timed Petri Net, wenn gilt

- *P ist eine endliche Menge von Stellen*
- *T ist eine endliche Menge von Transitionen*
- *F: $(P \times T) \cup (T \times P) \rightarrow \mathbb{N}$ ist eine Flussrelation*
- *m^0 ist eine Anfangsmarkierung*
- *D:$T \rightarrow \mathbb{Q}_0^+$ ist eine Funktion, welche jeder Transition $t \in T$ eine Schaltdauer zuordnet.* □

In TdPNs feuern Transitionen nicht zeitlos. Das hat jedoch den gravierenden Nachteil, dass nicht mehr zwangsläufig alle Marken zu jedem Zeitpunkt sichtbar sind. Dies schränkt die Verständlichkeit der Modelle erheblich ein und stellt somit eine potenzielle Fehlerquelle bei der Modellierung komplexer Systeme dar.

Sei zur Verdeutlichung das in Abbildung 2.2 dargestellte TdPN gegeben. Feuert in dem gegebenen TdPN die Transitionen t_3, so konsumiert diese eine Marke von der Stelle p_2, welche für eine Dauer von zwei Zeiteinheiten nicht sichtbar ist (analog für die Transition t_2).

Auch für TdPNs basiert die Verifikation gemäss [31],[32] auf einer Analyse des korrespondierenden Erreichbarkeitsgraphen und ist somit auf Grund der Explosion des Zustandsraums bei Anwendungen höherer Komplexität nicht praktikabel.

2.1.3 Timing Constraint Petri Nets (TCPNs)

Die Transitionen in *Timing Constraint Petri Nets (TCPNs)* [42] feuern analog zu den Transitionen in TdPNs nicht zeitlos. D.h. während des Feuerns einer Transition vergeht Zeit. Des Weiteren können sowohl die Stellen als auch die Transitionen eines TCPNs mit einem Intervall annotiert sein, welches eine untere und eine obere Zeitgrenze spezifiziert.

Definition 3 (Timing Constraint Petri Net (TCPN))
Ein Tupel G=(P,T,F,C,D,m^0) ist ein Timing Constraint Petri Net, wenn gilt

- P *ist eine endliche Menge von Stellen*
- T *ist eine endliche Menge von Transitionen*
- F: $(P \times T) \cup (T \times P) \rightarrow \mathbb{N}$ *ist eine Flussrelation*
- C *ist eine Funktion, welche einer Stelle oder einer Transition des TCPNs ein Intervall [min(pt),max(pt)] zuordnet, wobei pt entweder eine Stelle oder eine Transition ist*
- D:$T \rightarrow \mathbb{Q}_0^+$ *ordnet jeder Transition $t \in T$ eine Schaltdauer zu*
- m^0 *ist eine Anfangsmarkierung* □

Eine Transition ist in einem TCPN aktiviert, wenn jede ihrer Eingangsstellen hinreichend markiert ist. Wird eine Transition t zum Zeitpunkt t_0 aktiviert, so ist die Transition t in dem Intervall $[t_0+\min(t),t_0+\max(t)]$ feuerbar. Eine feuerbare Transition kann schalten, jedoch ist das Schalten einer feuerbaren Transition nicht zwingend. D.h. eine feuerbare Transition schaltet oder eben auch nicht. Die Abbildung 2.3 zeigt ein Beispiel für ein TCPN.
Auch für TCPNs basiert die Verifikation auf der Analyse des zugehörigen Erreichbarkeitsgraphen [42] und ist somit auf Grund der damit einhergehenden Komplexität nicht praktikabel. TCPNs eignen sich daher nicht für die Verifikation komplexer, zeitbewerteter Systeme.

2.1.4 Priority Duration Petri Nets (PDPNs)

Priority Duration Petri Nets (PDPNs) [44] sind eng verwandt mit den TdPNs. Analog zu TdPNs schalten Transitionen in PDPNs nicht zeitlos. Mit den Transitionen in PDPNs sind demnach Schaltdauern assoziiert. Des Weiteren sind den Transitionen eines PDPNs statische Prioritäten zugeordnet.

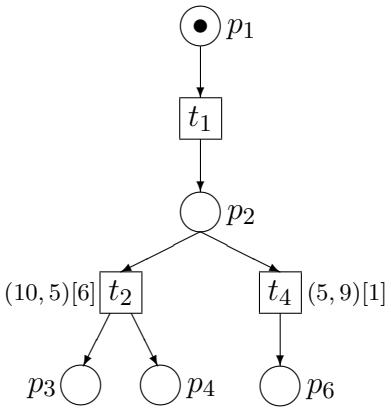


Abbildung 2.3: Beispiel für ein Timing Constraint Petri Net (vgl. [42])

Definition 4 (Priority Duration Petri Net (PDPN))
Ein Tupel G=(P,T,F,m^0,D,Θ) ist ein Priority Duration Petri Net, wenn gilt

- P *ist eine endliche Menge von Stellen*
- T *ist eine endliche Menge von Transitionen*
- F: $(P \times T) \cup (T \times P) \rightarrow \mathbb{N}$ *ist eine Flussrelation*
- m^0 *ist eine Anfangsmarkierung*
- D:$T \rightarrow \mathbb{Q}_0^+$ *ordnet jeder Transition eine Schaltdauer zu*
- Θ:$T \rightarrow \mathbb{N}$ *ordnet jeder Transition eine Priorität zu* □

Die Abbildung 2.4 zeigt ein PDPN (vgl. Abb.2.2). In dem gegebenen Beispiel feuern initial die Transitionen t_1 und t_2. Die Transition t_1 feuert zeitlos und markiert die Stelle p_2 mit drei Marken, so dass die Transition t_3 einmal und die Transition t_4 zweimal feuern kann. Erst im Anschluss daran wird die Uhr inkrementiert.
Analog zu den TdPNs feuern Transitionen in PDPNs nicht zeitlos. Die Modellierung zeitkonsumierender Transitionen hat zur Konsequenz, dass nicht mehr zwangsläufig alle Tupel zu jedem Zeitpunkt sichtbar sind. Die Modellierung zeitkonsumierender Transitionen schränkt die Verständlichkeit der Modelle somit erheblich ein.
Eine Verifikation von PDPNs erfolgt nach Werner et al. [44] auf Basis einer Analyse des zugehörigen Erreichbarkeitsgraphen und stellt somit auf Grund der damit verbundenen Komplexität keinen praktikablen Ansatz zur Analyse komplexer Modelle dar.

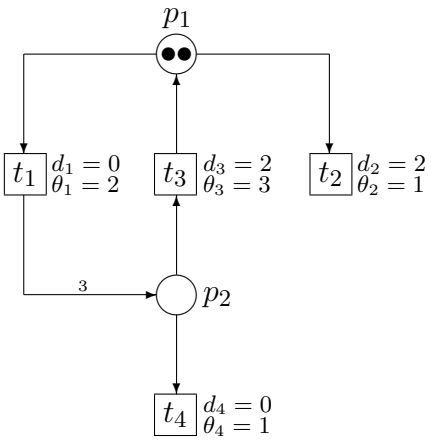


Abbildung 2.4: Beispiel für ein Priority Duration Petri Net [44]

2.2 Petrinetze mit zeiterweiterten Kanten

Ebenso wie die Transitionen können die Kanten eines Petrinetzes um ein Konzept zur Modellierung von Zeit erweitert werden. Zeitgrenzen an den eingehenden Kanten einer Transition spezifizieren, wann die Marken auf den Eingangsstellen der Transition verfügbar werden. Zeitgrenzen an den ausgehenden Kanten einer Transition spezifizieren, wann die Marken auf den Ausgangsstellen der Transition produziert werden. In [16] wurde gezeigt, dass beide Konzepte äquivalent sind und dass darüber hinaus TPNs und TdPNs als Spezialisierungen dieses generellen Konzepts angesehen werden können.

Zu den Petrinetzen mit zeiterweiterten Kanten zählen u.a. die *Arc Timed Petri Nets (ATPN)* [10] sowie [40],[43]. Die in [10] vorgeschlagene Methode zur Verifikation von ATPNs beruht auf der Analyse des zugehörigen Zustandsgraphen, welcher alle Zustände eines ATPNs repräsentiert. Analog zu der Erreichbarkeitsanalyse in TPNs und TdPNs ist eine solche Vorgehensweise jedoch auf Grund der Explosion des Zustandsraums bei Modellen größerer Komplexität nicht praktikabel.

2.3 Petrinetze mit zeiterweiterten Marken

In den *Timestamp Petri Nets (TSPN)* [11] sind die Marken eines Petrinetzes um ein Konzept zur Modellierung von Zeit angereichert. Jeder Marke in einem TSPN ist ein *Zeitstempel* zugeordnet, welcher den Zeitpunkt spezifiziert, zu dem die Marke auf der Stelle produziert wurde. Des Weiteren sind mit den Eingangskanten von Transitionen Intervalle assoziiert, welche spezifizieren, wann

eine Kante in Relation zu dem Zeitstempel der Marken auf den Eingangsstellen einer Transition *durchlässig* (engl. *permeable*) wird.

Definition 5 (Timestamp Petri Net)
Ein Tupel G=(P,T,F,λ_I) ist ein Timestamp Petri Net, wenn gilt

- *P ist eine endliche Menge von Stellen*
- *T ist eine endliche Menge von Transitionen*
- *F: $(P \times T) \cup (T \times P) \rightarrow \mathbb{N}$ ist eine Flussrelation*
- *λ_I ist eine Funktion, welche jeder Eingangskante einer Transition ein Intervall [lb,ub] zuweist, wobei lb die untere Grenze und ub die obere Grenze des Intervalles spezifizieren mit $lb \leq ub$.*

Die Abbildung 2.5 zeigt ein TSPN. Die Marke auf der Stelle p_1 hat den Zeitstempel 56. Somit ist die Kante $L(p_1, t_3)$ von 56 bis 60 durchlässig. Die Marke auf der Stelle p_2 hat den Zeitstempel 54. Daraus folgt, dass die Kante $L(p_2, t_3)$ von 57 an durchlässig ist. Die Transition t_3 kann somit von 57 bis 60 feuern. Eine Transition in einem TSPN kann nicht feuern, wenn ihre Eingangskanten nicht gleichzeitig durchlässig sind.

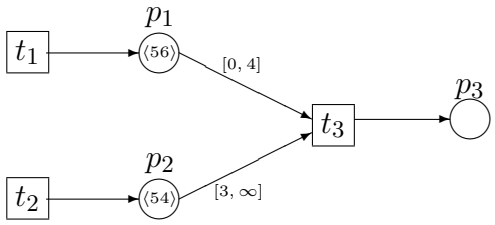


Abbildung 2.5: Beispiel für ein Timestamp Petri Net [38]

2.4 Fazit

Es existiert eine Vielzahl von Ansätzen, Konzepte zur Modellierung von Zeit in Petrinetze zu integrieren. In diesem Kapitel wurden verschiedene Lösungen

zur Integration temporärer Konzepte in Petrinetze vorgestellt. Allen genannten Ansätzen zeiterweiterter Petrinetze ist jedoch gemein, dass sie zwar eine Modellierung und Simulation zeitbewerteter Systeme ermöglichen, jedoch einer Möglichkeit entbehren, die gegebenen Modelle strukturell zu analysieren. Zwar integrieren zahlreiche Ansätze eine Verifikation auf Basis einer Analyse des Erreichbarkeitsgraphen, jedoch ist eine Erreichbarkeitsanalyse bei der Verifikation zeitbewerteter Modelle von realem Komplexitätsumfang auf Grund der Zustandsraumexplosion nicht praktikabel. Das Potenzial von Petrinetzen, die Modelle unmittelbar strukturell analysieren und verifizieren zu können, bleibt somit bei der Mehrheit der zeiterweiterten Petrinetze ungenutzt. Im Rahmen dieser Arbeit wurde daher mit den *zeitbewerteten Prädikat/Transitionsnetzen* eine Petrinetzklasse entwickelt, welche neben der Modellierung und Simulation auch eine Verifikation auf Basis einer strukturellen Analyse der Modelle erlaubt.

3 Summenbeschriftete Prädikat/Transitions-Netze

Im diesem Kapitel werden die *summenbeschrifteten Prädikat/Transitions-Netze (S-Pr/T-Netze)* eingeführt. Diese gehen aus den von K. Lautenbach in [17] eingeführten *Simple Marked-graph-like Predicate Transition Nets* hervor. Durch Anwendung der Modulo-Operationen $\oplus$ und $\ominus$ werden in S-Pr/T-Netzen die Wertebereiche der Variablen beschränkt. S-Pr/T-Netze sind somit endlich und strukturell analysierbar. Da S-Pr/T-Netze über eine hinreichende Modellierungsmächtigkeit für die Darstellung temporärer Zusammenhänge verfügen und weiterhin strukturell analysierbar sind, bilden sie die Grundlage für die im Rahmen dieser Arbeit neu entwickelte Klasse der *zeitbewerteten Prädikat/Transitions-Netze (Z-Pr/T-Netze).*
In diesem Kapitel erfolgt zunächst in Kapitel 3.1 eine Definition der Klasse der S-Pr/T-Netze. Anschließend wird in Kapitel 3.2 das dynamische Verhalten präsentiert und an einem Beispiel detailliert erläutert. Abschließend werden in Kapitel 3.3 die verschiedenen Methoden zur strukturellen Analyse von S-Pr/T-Netzen erörtert. Dazu zählen die strukturelle Analyse mittels S- und T-Invarianten sowie die Identifikation von Traps und Co-Traps. Dabei stützt sich dieses Kapitel in weiten Teilen auf die Arbeiten von K. Lautenbach [17] und V. Kindler [15].

3.1 Einführung in S-Pr/T-Netze

Definition 6 (Multimenge)
Sei A eine endliche Menge $A = \{a_1, a_2, ..., a_n\}$.
Eine Abbildung $m : A \longrightarrow \mathbb{Z}$ heißt Multimenge über A.
Eine Abbildung $m_+ : A \longrightarrow \mathbb{N}_0$ heißt nicht-negative Multimenge über A .

MS(A) sei die Menge aller Multimengen über A, $MS_+(A)$ die Menge aller nicht-negativen Multimengen über A.

□

Multimengen werden als formale Summen wie folgt geschrieben:
$m = m(a_1) \cdot a_1 + m(a_2) \cdot a_2 + ... + m(a_n) \cdot a_n$

Definition 7 *Sei $A = \{a_1, ..., a_n\}$ eine endliche Menge und seien m, m_1 und m_2 Multimengen über A. Dann definieren wir:*

$$\begin{aligned} m = m_1 + m_2 &\iff \forall a \in A : m(a) = m_1(a) + m_2(a) \\ m = m_1 - m_2 &\iff \forall a \in A : m(a) = m_1(a) - m_2(a) \\ m_1 \leq m_2 &\iff \forall a \in A : m_1(a) \leq m_2(a) \\ m_1 \geq m_2 &\iff \forall a \in A : m_1(a) \geq m_2(a) \end{aligned}$$

□

Beispiel 1
Die Besonderheit von Multimengen gegenüber „gewöhnlichen“ Mengen besteht darin, dass einzelne Elemente in einer Multimenge mehrfach vorkommen können. Seien $m_1 = \{a, c, c\}$ und $m_2 = \{a, a, b, c, c\}$ Multimengen. So können diese ebenso als formale Summen geschrieben werden:

$$m_1 = 1 \cdot a + 0 \cdot b + 2 \cdot c \qquad \textit{und} \qquad m_2 = 2 \cdot a + 1 \cdot b + 2 \cdot c$$

Sowohl m_1 als auch m_2 sind nicht-negative Multimengen. Die Summe und Differenz aus m_1 und m_2 berechnen sich wie folgt:

$$\begin{aligned} m_1 + m_2 &= (1+2) \cdot a + (0+1) \cdot b + (2+2) \cdot c = 3a + b + 4c \\ m_1 - m_2 &= (1-2) \cdot a + (0-1) \cdot b + (2-2) \cdot c = -a - b + 0c \end{aligned}$$

Des weiteren gilt $m_2 \geq m_1$, da $m_2(a) = 2 \geq 1 = m_1(a)$, $m_2(b) = 1 \geq 0 = m_1(b)$ und $m_2(c) = 2 \geq 2 = m_1(c)$.

□

Definition 8
Sei $n \in \mathbb{N}$ und X eine nichtleere, endliche Menge von Variablen mit Wertebereich $\mathbb{Z}$. Dann definieren wir:

$$\begin{aligned} < N > &:= \{< 0 >, < 1 >, ..., < n-1 >\} \\ < X \oplus N > &:= \{< x \oplus h > \mid x \in X \quad \textit{und} \quad h \in \{0, 1, ..., n-1\}\} \end{aligned}$$

□

Definition 9 (S-Pr/T-Netz)
Sei X eine nichtleere, endliche Menge von Variablen, $n \in \mathbb{N}$ und $< X \oplus N >$ wie in Definition 8 gegeben. Dann heißt $G = (\ P,\ T,\ F,\ L,\ n\ )$ summenbeschriftetes Prädikat/Transitions-Netz, wenn gilt:

- P *ist eine endliche Menge von Stellen*
- T *ist eine endliche Menge von Transitionen*
- F *ist eine Flussrelation, für die gilt:*
  - $F = (P \times T) \cup (T \times P)$
  - $P \cap T = \emptyset$
  - $P \cup T \neq \emptyset$
- $L : F \longrightarrow \ < X \oplus N >$
- n *ist der Modulus des Netzes* □

Anmerkung:
Durch die Symbole „<>“ wird in S-Pr/T-Netzen das anonyme Token repräsentiert.

3.2 Dynamik in S-Pr/T-Netzen

Im Folgenden wird nun die Dynamik von S-Pr/T-Netzen eingehend erläutert.

Definition 10 (Vor- und Nachbereich)
Sei $p \in P$ eine Stelle und $t \in T$ eine Transition. Dann seien

$$\begin{aligned} {}^\bullet p &:= \{\ t \ \mid\ (t,p) \in F\ \} \\ p^\bullet &:= \{\ t \ \mid\ (p,t) \in F\ \} \\ {}^\bullet t &:= \{\ p \ \mid\ (p,t) \in F\ \} \\ t^\bullet &:= \{\ p \ \mid\ (t,p) \in F\ \} \end{aligned}$$

Vorbereich und Nachbereich von p bzw. t. □

Definition 11 (S- und T-Vektor)
Sei G = (P,T,F,L,n) ein S-Pr/T-Netz und seien die Menge der Stellen und die Menge der Transitionen beliebig, aber fest geordnet, so dass diese als (endliche) Indexmenge benutzt werden können.
Ein mit P indizierter Vektor der Länge | P | heißt S-Vektor und ein mit T indizierter Vektor der Länge | T | heißt T-Vektor.

Im Folgenden werden ausschließlich S- bzw. T-Vektoren mit Multimengen über <N> als Einträgen betrachtet. Ein S- bzw. T-Vektor hat dann die Form:

$$S : P \longrightarrow MS(<N>) \qquad bzw. \qquad U : T \longrightarrow MS(<N>)$$

Ein S- bzw. T-Vektor heißt nicht-negativ, wenn alle Einträge nicht-negative Multimengen sind.

Ein S- bzw. T-Vektor heißt konstant, wenn alle Einträge aus Multimengen über Konstanten bestehen. Anderenfalls heißt der S- bzw. T-Vektor parametrisiert.

□

Beispiel 2
Sei $G = (P, T, F, L, n)$ ein S-Pr/T-Netz mit $P = \{p_1, p_2, p_3\}$, $T = \{t_1, t_2\}$ und Modulus $n = 2$.

$$S_1 = \begin{bmatrix} a\cdot < x > + b\cdot < x \oplus 1 > \\ c\cdot < x > + d\cdot < x \oplus 1 > \\ e\cdot < x > + f\cdot < x \oplus 1 > \end{bmatrix} \qquad {S_1}' = \begin{bmatrix} a\cdot < 0 > + b\cdot < 1 > \\ c\cdot < 0 > + d\cdot < 1 > \\ e\cdot < 0 > + f\cdot < 1 > \end{bmatrix}$$

Dann ist S_1 ein mit der Variablen x parametrisierter S-Vektor, wohingegen ${S_1}'$ ein konstanter S-Vektor ist. Wenn $a, b, c, d, e, f \in \mathbb{N}_0$ gilt, dann sind S_1 und ${S_1}'$ darüber hinaus nicht-negative S-Vektoren.

$$U_1 = \begin{bmatrix} a\cdot < x > + b\cdot < x \oplus 1 > \\ c\cdot < x > + d\cdot < x \oplus 1 > \end{bmatrix} \qquad {U_1}' = \begin{bmatrix} a\cdot < 0 > + b\cdot < 1 > \\ c\cdot < 0 > + d\cdot < 1 > \end{bmatrix}$$

Analog ist U_1 ein mit der Variablen x parametrisierter T-Vektor und ${U_1}'$ ein konstanter T-Vektor. U_1 und ${U_1}'$ sind nicht-negative T-Vektoren, wenn $a, b, c, d \in \mathbb{N}_0$ gilt.

□

Definition 12 (Markierung)
Sei G=(P,T,F,L,n) ein S-Pr/T-Netz. Eine Markierung von G ist eine Abbildung $M : P \longrightarrow MS_+(< N >)$.

□

Definition 13 (Inzidenzmatrix)
Sei G = (P,T,F,L,n) ein S-Pr/T-Netz. Die Inzidenzmatrix $[G(p,t)]$ von G ist wie folgt definiert:

$$G(p,t) = \begin{cases} L(t,p) - L(p,t), & falls\ (t,p) \in F \cap (T \times P)\ und\ (p,t) \in F \cap (P \times T) \\ -L(p,t), & falls\ (p,t) \in F \cap (P \times T)\ und\ (t,p) \notin F \\ L(t,p), & falls\ (p,t) \notin F\ und\ (t,p) \in F \cap (T \times P) \\ \emptyset, & sonst \end{cases}$$ □

Dabei repräsentieren die Spalten der Inzidenzmatrix die Transitionen des S-Pr/T-Netzes, wohingegen die Zeilen der Inzidenzmatrix die Stellen des S-Pr/T-Netzes repräsentieren. Im Folgenden dienen N(p,-) bzw. N(-,t) als Abkürzung für die Zeilen bzw. Spalten der Inzidenzmatrix.

Definition 14 (aktiviert)
Sei M eine Markierung, $t \in T$, $p \in P$, $k \in \mathbb{Z}$ und $X = \{x\}$. Eine Transition t ist unter M aktiviert für $< x >=< k >$ gdw.

$$\forall p \in {}^\bullet t: \quad M(p) \geq L(p,t),$$

wobei das x aus L(p,t) durch das k aus M(p) substituiert ist. □

D.h. eine Transition ist aktiviert, wenn alle Stellen im Vorbereich der Transition gemäss der Kantengewichte markiert sind.

Definition 15 (Schalten, Folgemarkierung)
Mit obigen Bezeichnern gilt: Falls t unter M markiert ist, kann t schalten (bzw. feuern) für $< x >=< k >$ und dadurch in eine Folgemarkierung M' überführen mit

$$M'(p) = \begin{cases} M(p) - L(p,t) & falls\ \ p \in {}^\bullet t\ und\ p \notin t^\bullet \\ M(p) + L(t,p) & falls\ \ p \notin {}^\bullet t\ und\ p \in t^\bullet \\ M(p) - L(p,t) + L(t,p) & falls\ \ p \in {}^\bullet t \cap t^\bullet \\ M(p) & falls\ \ p \in P \setminus ({}^\bullet t \cap t^\bullet) \end{cases},$$

wobei das x aus L(p,t) bzw. L(t,p) durch das k aus M(p) substituiert ist. □

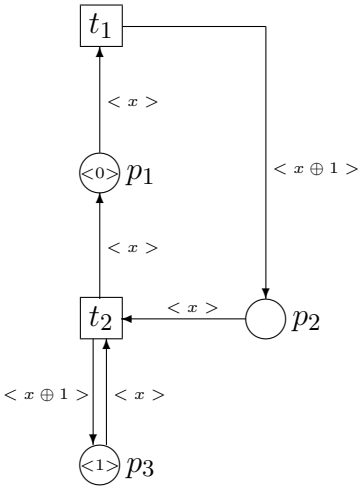


Abbildung 3.1: S-Pr/T-Netz G_1 [17]

Beispiel 3
Die Dynamik in S-Pr/T-Netzen wird nun anhand des in Abbildung 3.1 gegebenen S-Pr/T-Netzes G_1 verdeutlicht. Dabei sei der Modulus des S-Pr/T-Netzes G_1 n=2.

Die Anfangsmarkierung M_0 ist

$$M_0 = \begin{bmatrix} 1\cdot < 0 > +0\cdot < 1 > \\ 0\cdot < 0 > +0\cdot < 1 > \\ 0\cdot < 0 > +1\cdot < 1 > \end{bmatrix} = \begin{bmatrix} < 0 > \\ \emptyset \\ < 1 > \end{bmatrix}$$

Die Inzidenzmatrix von G_1 lautet

$$G_1(\text{p,t}) = \begin{array}{c|c|c} & t_1 & t_2 \\ \hline p_1 & - < x > & < x > \\ p_2 & < x \oplus 1 > & - < x > \\ p_3 & \emptyset & - < x > + < x \oplus 1 > \end{array}$$

Demnach repräsentiert die nachfolgende Schaltsequenz die Dynamik in dem S-Pr/T-Netz G_1:

$$\begin{bmatrix} < 0 > \\ \emptyset \\ < 1 > \end{bmatrix} \xrightarrow{t_1} \begin{bmatrix} \emptyset \\ < 1 > \\ < 1 > \end{bmatrix} \xrightarrow{t_2} \begin{bmatrix} < 1 > \\ \emptyset \\ < 0 > \end{bmatrix} \xrightarrow{t_1} \begin{bmatrix} \emptyset \\ < 0 > \\ < 0 > \end{bmatrix}$$

$$\xrightarrow{t_2} \begin{bmatrix} <0> \\ \emptyset \\ <1> \end{bmatrix}$$

□

3.3 Strukturelle Analyse in S-Pr/T-Netzen

Ein Vorteil von Petrinetzen als Modellierungssprache ist – neben ihrer graphischen Visualisierbarkeit – ihre formale Basis, welche eine strukturelle Analyse der Modelle erlaubt. Die so extrahierten, strukturellen Eigenschaften erlauben Rückschlüsse auf das Systemverhalten, also auf Aspekte der Dynamik des Systems, und können zur Verifikation von Systemeigenschaften genutzt werden.
Im Folgenden wird die strukturelle Analyse von Z-Pr/T-Netzen mittels S- und T-Invarianten sowie Traps und Co-Traps vorgestellt und anhand von Beispielen verdeutlicht.

3.3.1 S- und T-Invarianten in S-Pr/T-Netzen

S- und T-Invarianten sind strukturelle Komponenten, welche zur Verifikation von Systemeigenschaften verwendet werden können. Im Folgenden werden zunächst S-Invarianten und anschließend T-Invarianten definiert und anhand eines Beispieles erläutert.

S-Invarianten
S-Invarianten sind strukturelle Komponenten, für die gilt, dass die gewichtete Anzahl der Marken auf den Stellen der S-Invarianten konstant ist.
In Stellen/Transitions-Netzen können S-Invarianten als Lösung homogener Gleichungssysteme berechnet werden. Da in S-Pr/T-Netzen jedoch auf Tupeln über Multimengen gerechnet wird, muss zunächst eine entsprechende Operation eingeführt werden, welche das Rechnen auf Multimengen erlaubt. Dies erfolgt mit der Einführung des S-Produktes.
Das S-Produkt beschreibt – analog zu dem Produkt zwischen ganzen Zahlen – ein Produkt zwischen Multimengen. Mittels dieser Operation ist es möglich, das Auftreten von Tupeln in Multimengen zu erfassen und Tupel in Multimengen auf Gleichheit zu überprüfen.

Definition 16 (S-Produkt)
Seien $x \in X$, $z \in X \cup \mathbb{Z}$, $h, k \in \mathbb{Z}$ *und* $n \in \mathbb{N}$. *Dann sei das S-Produkt wie folgt definiert:*

$$< h > \bullet < k >= \begin{cases} 1, & falls \quad h \equiv k (mod\ n) \\ 0, & sonst \end{cases}$$

Seien $V_1 = (m_1, m_2, ..., m_r)^T$ *und* $V_2 = (n_1, n_2, ..., n_r)^T$ *zwei Vektoren gleicher Länge r mit Multimengen über konstanten und variablen 1-Tupeln als Einträgen. Seien die Multimengen* m_l *und* n_k *mit* $1 \leq k, l \leq r$ *über die endliche Menge A mit* $|A| = n$ *definiert. Dann ist das S-Produkt von* V_1 *und* V_2 *definiert als:*

$$V_1^T \bullet V_2 \;:=\; \sum_{l=1}^{r} \sum_{i=1}^{n} \sum_{j=1}^{n} m_l(a_i) \cdot n_l(b_j) \cdot (a_i \bullet b_j)$$

Mit obigen Bezeichnern soll gelten:

$$\begin{aligned} < h > \bullet \emptyset &= 0 \\ < x \oplus h > \bullet \emptyset &= 0 \\ \emptyset \bullet \emptyset &= 0 \\ <> \bullet <> &= 1 \\ <> \bullet < z > &= 0 \end{aligned}$$

□

Anmerkung:
Das S-Produkt ist kommutativ und distributiv bezüglich der formalen Summen für Multimengen. Des Weiteren gilt mit obigen Bezeichnern und $y \in X$:

$$< x \oplus h > \bullet < y \oplus k > \;=\; < x \oplus (h \pm z) > \bullet < y \oplus (k \pm z) >$$

$$< x \oplus h > \bullet < x \oplus k > \;=\; \begin{cases} 1 & falls \;\; h \equiv k (mod\ n) \\ 0 & sonst \end{cases}$$

Beispiel 4
Sei $x \in X$ und der Modulus n=2, dann ist:

$$\begin{aligned} < x > \bullet < x \oplus 1 > &= 0 \\ < x > \bullet < x \oplus 2 > &= < x > \bullet < x > \;=\; 1 \end{aligned}$$

Seien $x, y \in X$, V_1,V_2 zwei Vektoren gleicher Länge und der Modulus n=2 mit:

$$V_1 = \begin{bmatrix} 1 \cdot < x > + 2 \cdot < x \oplus 1 > \\ 3 \cdot < x > + 4 \cdot < x \oplus 1 > \end{bmatrix} \qquad V_2 = \begin{bmatrix} 4 \cdot < y > + 3 \cdot < y \oplus 1 > \\ 2 \cdot < y > + 1 \cdot < y \oplus 1 >) \end{bmatrix}$$

Dann berechnet sich das S-Produkt der Vektoren V_1 und V_2 wie folgt:

$$\begin{aligned} V_1^T \bullet V_2 \;=\;& 1\cdot 4\cdot(<x>\bullet<y>) \;+\; 1\cdot 3\cdot(<x>\bullet<y\oplus 1>) \\ &+ 2\cdot 4\cdot(<x\oplus 1>\bullet<y>) + 2\cdot 3\cdot(<x\oplus 1>\bullet<y\oplus 1>) \\ &+ 3\cdot 2\cdot(<x>\bullet<y>) \;+\; 3\cdot 1\cdot(<x>\bullet<y\oplus 1>) \\ &+ 4\cdot 2\cdot(<x\oplus 1>\bullet<y>) + 4\cdot 1\cdot(<x\oplus 1>\bullet<y\oplus 1>) \\ \\ =\;& 20\cdot(<x>\bullet<y>) \;+\; 22\cdot(<x>\bullet<y\oplus 1>) \\ \\ =\;& \begin{cases} 20 & falls \;\; x=y \\ 22 & sonst \end{cases} \end{aligned}$$

□

Definition 17 (markiert)
Sei S ein S-Vektor und M eine Markierung. Die durch S spezifizierte Stellenmenge heißt markiert, wenn gilt: $S^T \bullet M > 0$.

Anderenfalls heißt die durch S gegebene Stellenmenge unmarkiert. □

Definition 18 (S-Invariante)
Sei G=(P,T,F,L,n) ein S-Pr/T-Netz. Ein S-Vektor S heißt S-Invariante von G, wenn gilt:

$$S^T \bullet [G(p,t)] = 0^T \;.$$

Eine S-Invariante S heißt konstant, wenn alle ihre Einträge aus Multimengen über Konstanten bestehen. Anderenfalls heißt sie parametrisiert. □

Beispiel 5
Sei erneut das in Abbildung 3.1 dargestellte S-Pr/T-Netz G_1 mit nachfolgender Inzidenzmatrix gegeben:

G_1(p,t) =

	t_1	t_2
p_1	$-<x>$	$<x>$
p_2	$<x\oplus 1>$	$-<x>$
p_3	$\emptyset$	$-<x>+<x\oplus 1>$

Sei des Weiteren ein Stellenvektor S wie folgt gegeben:

$$S = \begin{bmatrix} 1\cdot<y> \;+\; 0\cdot<y\oplus 1> \\ 0\cdot<y> \;+\; 1\cdot<y\oplus 1> \\ 1\cdot<y> \;+\; 0\cdot<y\oplus 1> \end{bmatrix} = \begin{bmatrix} <y> \\ <y\oplus 1> \\ <y> \end{bmatrix}$$

Ist der S-Vektor S eine S-Invariante für das S-Pr/T-Netz G_1, so muss gelten: $S^T \bullet [G_1(p,t)] = 0^T$.

$$
\begin{aligned}
S^T \bullet G_1(-,t_1) &= < y > \bullet(- < x >) + < y \oplus 1 > \bullet < x \oplus 1 > + < y > \bullet \emptyset \\
&= \begin{cases} -1+1+0=0 & \text{für } x = y \\ -0+0+0=0 & \textit{sonst} \end{cases} \\
S^T \bullet G_1(-,t_2) &= < y > \bullet < x > + < y \oplus 1 > \bullet(- < x >) \\
&\qquad + < y > \bullet(- < x > + < x \oplus 1 >) \\
&= \begin{cases} 1-0-1+0=0 & \text{für } x = y \\ 0-1-0+1=0 & \textit{sonst} \end{cases}
\end{aligned}
$$

Der S-Vektor S ist somit eine S-Invariante für das in Abbildung 3.1 dargestellte S-Pr/T-Netz G_1. Es gilt, dass die gewichtete Anzahl der Tupel auf den Stellen einer S-Invarianten konstant ist.
Dabei ist der S-Vektor S eine mit Variable y parametrisierte S-Invariante, wohingegen die folgenden S-Vektoren konstante S-Invarianten für das S-Pr/T-Netz G_1 sind:

$$S_1 = \begin{bmatrix} < 0 > \\ < 1 > \\ < 0 > \end{bmatrix} \qquad S_2 = \begin{bmatrix} < 1 > \\ < 0 > \\ < 1 > \end{bmatrix}$$ □

T-Invarianten
Analog zu der Berechnung von S-Invarianten lässt sich die Berechnung von T-Invarianten in Stellen/Transitions-Netzen auf das Lösen homogener Gleichungssysteme zurückführen. Diese Vorgehensweise lässt sich ohne Einführung einer weiteren Operation, welche die korrespondierende Operation auf Multimengen repräsentiert, nicht unmittelbar auf S-Pr/T-Netze übertragen. Es wird daher zunächst das T-Produkt als grundlegende Operation eingeführt, welche das Berechnen von T-Invarianten in S-Pr/T-Netzen erlaubt.
Das T-Produkt liefert eine Notation für die Substitution von Variablen in Tupeln.

Definition 19 (T-Produkt)
Sei $x \in X$, $h \in \mathbb{Z}$, $z \in X \cup \mathbb{Z}$. *Dann sei das T-Produkt wie folgt definiert:*

$$
\begin{aligned}
< x \oplus h > \circ < z > &:= < z \oplus h > \\
< h > \circ < z > &:= < h >
\end{aligned}
$$

$$\begin{aligned} < z > \circ \emptyset &:= \emptyset \\ \emptyset \circ < z > &:= \emptyset \\ <> \circ < z > &:= <> \\ < z > \circ <> &:= < z > \end{aligned}$$

Seien $V_1 = (m_1, m_2, ..., m_r)^T$ *und* $V_2 = (n_1, n_2, ..., n_r)^T$ *zwei Vektoren gleicher Länge r mit Multimengen über konstanten und variablen 1-Tupeln als Einträgen. Seien die Multimengen* m_l *und* n_k *mit* $1 \leq k, l \leq r$ *über die endliche Menge A mit* $|A| = n$ *definiert.*

Dann ist das T-Produkt von V_1 *und* V_2 *definiert als:*

$$V_1^T \circ V_2 := \sum_{l=1}^{r} \sum_{i=1}^{n} \sum_{j=1}^{n} m_l(a_i) \cdot n_l(b_j) \cdot (a_i \circ b_j)$$

□

Das T-Produkt dient der Substitution von Variablen in Tupeln in Verbindung mit dem Feuern von Transitionen und ist distributiv bezüglich der formalen Summen für Multimengen.

Beispiel 6
Seien $x, y \in X$, V_1,V_2 zwei Vektoren gleicher Länge und der Modulus n=2 mit:

$$V_1 = \begin{bmatrix} 1\cdot < x > +2\cdot < x \oplus 1 > \\ 3\cdot < x > +4\cdot < x \oplus 1 > \end{bmatrix} \qquad V_2 = \begin{bmatrix} 4\cdot < y > +3\cdot < y \oplus 1 > \\ 2\cdot < y > +1\cdot < y \oplus 1 >) \end{bmatrix}$$

Dann berechnet sich das T-Produkt der Vektoren V_1 und V_2 wie folgt:

$$\begin{aligned} V_1^T \circ V_2 = \quad & 1 \cdot 4 \cdot (< x > \circ < y >) + 1 \cdot 3 \cdot (< x > \circ < y \oplus 1 >) \\ & + 2 \cdot 4 \cdot (< x \oplus 1 > \circ < y >) + 2 \cdot 3 \cdot (< x \oplus 1 > \circ < y \oplus 1 >) \\ & + 3 \cdot 2 \cdot (< x > \circ < y >) + 3 \cdot 1 \cdot (< x > \circ < y \oplus 1 >) \\ & + 4 \cdot 2 \cdot (< x \oplus 1 > \circ < y >) + 4 \cdot 1 \cdot (< x \oplus 1 > \circ < y \oplus 1 >) \\ \\ = \quad & 4 \cdot (< x > \circ < y >) + 3 \cdot (< x \oplus 1 > \circ < y >) \\ & + 8 \cdot (< x \oplus 1 > \circ < y >) + 6 \cdot (< x > \circ < y >) \\ & + 6 \cdot (< x > \circ < y >) + 3 \cdot (< x \oplus 1 > \circ < y >) \\ & + 8 \cdot (< x \oplus 1 > \circ < y >) + 4 \cdot (< x > \circ < y >) \end{aligned}$$

$$= \quad 20 \cdot (< x > \circ < y >) \; + \; 22 \cdot (< x \oplus 1 > \circ < y >)$$

$$= \quad 20 \cdot < y > \; + \; 22 \cdot < y \oplus 1 >$$

Dabei gilt unter Berücksichtigung der Modulo-Operation mit Modulus n=2 für die obige Berechnung:

$$\begin{aligned} < x > \circ < y \oplus 1 > &= < x \oplus 1 > \circ < y > \\ < x > \circ < y > &= < x \oplus 1 > \circ < y \oplus 1 > \end{aligned}$$

□

Definition 20 (T-Invariante)
Sei G=(P,T,F,L,n) ein S-Pr/T-Netz. Ein T-Vektor U heißt T-Invariante von G, wenn gilt:

$$[G(p,t)] \circ U = 0 \; .$$

Eine T-Invariante heißt konstant, wenn alle Einträge aus Multimengen über Konstanten bestehen. Anderenfalls heißt die T-Invariante parametrisiert. □

Beispiel 7
Sei erneut das in Abbildung 3.1 dargestellte S-Pr/T-Netz G_1 sowie folgender T-Vektor gegeben:

$$U = \begin{bmatrix} 1 \cdot < y > + 1 \cdot < y \oplus 1 > \\ 1 \cdot < y > + 1 \cdot < y \oplus 1 > \end{bmatrix} = \begin{bmatrix} < y > + < y \oplus 1 > \\ < y > + < y \oplus 1 > \end{bmatrix}$$

Ist der T-Vektor U eine T-Invariante für das S-Pr/T-Netz G_1, so muss gelten: $[G_1(p,t)] \circ U = 0$.

$$\begin{aligned} G_1(p_1,-) \circ U &= - < x > \circ (< y > + < y \oplus 1 >) + < x > \circ (< y > + < y \oplus 1 >) \\ &= 0 \end{aligned}$$

$$\begin{aligned} G_1(p_2,-) \circ U &= < x \oplus 1 > \circ (< y > + < y \oplus 1 >) - < x > \circ (< y > + < y \oplus 1 >) \\ &= < x > \circ (< y \oplus 1 > + < y >) - < x > \circ (< y > + < y \oplus 1 >) \\ &= 0 \end{aligned}$$

$$\begin{aligned} G_1(p_3,-) \circ U &= (- < x > + < x \oplus 1 >) \circ (< y > + < y \oplus 1 >) \\ &= - < x > \circ (< y > + < y \oplus 1 >) + < x \oplus 1 > \circ (< y > + < y \oplus 1 >) \\ &= - < x > \circ (< y > + < y \oplus 1 >) + < x > \circ (< y \oplus 1 > + < y >) \\ &= 0 \end{aligned}$$

Hierbei ist zu beachten, dass unter Berücksichtigung der Modulo-Operation mit Modulus $n = 2$ folgendes gilt:

$$\begin{aligned} < x \oplus 1 > \circ < y > &= < x > \circ < y \oplus 1 > \\ < x \oplus 1 > \circ < y \oplus 1 > &= < x > \circ < y > \end{aligned}$$

Der T-Vektor U ist somit eine T-Invariante für das in Abbildung 3.1 dargestellte S-Pr/T-Netz G_1. Eine T-Invariante spezifiziert eine Schaltsequenz zur Reproduktion einer Markierung.
In dem gegebenen Beispiel ist die Anfangsmarkierung M_0 durch die T-Invariante reproduzierbar, indem sowohl die Transition t_1 als auch die Transition t_2 jeweils einmal für ein Tupel $< y >$ und einmal für ein Tupel $< y \oplus 1 >$ feuern.
Der T-Vektor U ist eine mit Variable y parametrisierte T-Invariante für das S-Pr/T-Netz G_1, wohingegen der folgende T-Vektor eine konstante T-Invariante für das S-Pr/T-Netz G_1 ist:

$$U_1 = \begin{bmatrix} < 0 > + < 1 > \\ < 0 > + < 1 > \end{bmatrix}$$

□

3.3.2 Traps und Co-Traps in S-Pr/T-Netzen

Neben S- und T-Invarianten bieten Traps und Co-Traps eine weitere Möglichkeit der strukturellen Analyse von Petrinetzen.

Co-Traps
In einem S/T-Netz ist eine Stellenmenge ein Co-Trap, wenn jede Transition, die eine Marke auf einer Stelle des Co-Traps produziert, auch eine Marke von einer Stelle des Co-Traps konsumiert. In S-Pr/T-Netzen sind Co-Traps wie folgt definiert:

Definition 21 (Co-Trap)
Sei $G = (P, T, F, L, n)$ ein S-Pr/T-Netz und sei des Weiteren $C : P \rightarrow MS_+(<N>)$ ein S-Vektor mit Einträgen aus Multimengen über <N>. Der S-Vektor C ist ein Co-Trap für das S-Pr/T-Netz G, wenn gilt: Jede Transition t, welche eine Stelle $p \in C$ mit einem Tupel gemäss der Kantenanschrift $L(t, p)$ markiert, konsumiert auch ein Tupel gemäss der Kantenanschrift $L(p', t)$ von einer Stelle $p' \in C$.

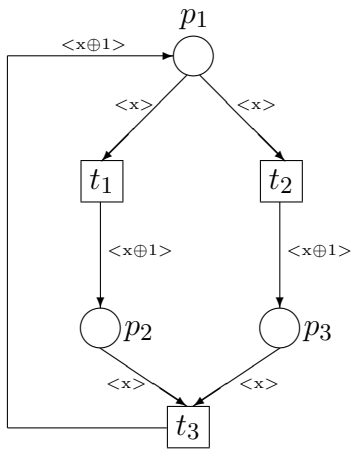


Abbildung 3.2: S-Pr/T-Netz G_2

Ein Co-Trap heißt konstant, wenn alle seine Einträge aus Multimengen über Konstanten bestehen. Anderenfalls heißt der Co-Trap parametrisiert.

Ein Co-Trap C heißt unter einer Markierung M faktisch markiert, wenn gilt: $C^T \bullet M > 0$. Anderenfalls heißt der Co-Trap C faktisch nicht markiert unter M.

□

Eine Stellenmenge ist in einem S-Pr/T-Netz ein Co-Trap, wenn jede Transition, welche eine Stelle des Co-Traps mit einem Tupel markiert, auch wieder ein entsprechendes Tupel gemäss der Kantenanschriften von einer Stelle des Co-Traps konsumiert. Ein Co-Trap heißt in einem S-Pr/T-Netz faktisch nicht markiert, wenn keine Stelle des Co-Traps derart markiert ist, so dass eine Transition in dem S-Pr/T-Netz schalten und den Co-Trap faktisch wieder markieren kann. D.h. die Stellen eines faktisch nicht markierten Co-Traps können durchaus markiert sein, jedoch ausschließlich mit Variablenbelegungen, welche keine erneute faktische Markierung des Co-Traps erlauben. Ein faktisch nicht markierter Co-Trap kann somit auch nicht wieder faktisch markiert werden. Sei zur Verdeutlichung das nachfolgende Beispiel gegeben.

Beispiel 8
Sei das in Abbildung 3.2 dargestellte S-Pr/T-Netz mit Modulus $n = 2$ gegeben. Dann sind die folgenden S-Vektoren Co-Traps für das S-Pr/T-Netz G_2:

$$C_1 = \begin{bmatrix} < x > \\ < x \oplus 1 > \\ \emptyset \end{bmatrix} \qquad C_2 = \begin{bmatrix} < x > \\ \emptyset \\ < x \oplus 1 > \end{bmatrix} \qquad C_3 = \begin{bmatrix} < x > \\ < x \oplus 1 > \\ < x \oplus 1 > \end{bmatrix}$$

Der S-Vektor C_4 ist jedoch kein Co-Trap für das S-Pr/T-Netz G_2:

$$C_4 = \begin{bmatrix} < x \oplus 1 > \\ < x \oplus 1 > \\ \emptyset \end{bmatrix}$$

Demnach markiert die Transition t_3 die Stelle $p_1 \in C_4$ mit einem Tupel $< x \oplus 1 >$ und konsumiert ein Tupel $< x \oplus 1 >$ von der Stelle $p_2 \in C_4$. Dies widerspricht jedoch den gegebenen Kantenanschriften. Der S-Vektor C_4 ist somit kein Co-Trap für das S-Pr/T-Netz G_2 (analog für die Transition t_1).

Die S-Vektoren C_1, C_2 und C_3 sind mit der Variablen x parametrisierte Co-Traps für das S-Pr/T-Netz G_2.

Der S-Vektor C_5 hingegen ist für das S-Pr/T-Netz G_2 ein zu C_1 korrelierender, konstanter Co-Trap. Seien C_5 und die Markierung des Netzes mit M_1 wie folgt gegeben:

$$C_5 = \begin{bmatrix} < 0 > \\ < 1 > \\ \emptyset \end{bmatrix} \qquad M_1 = \begin{bmatrix} < 1 > \\ \emptyset \\ \emptyset \end{bmatrix}$$

Der Co-Trap C_5 ist unter der Markierung M_1 faktisch nicht markiert, da gilt:

$${C_5}^T \bullet M_1 = [< 0 >, < 1 >, \emptyset] \bullet \begin{bmatrix} < 1 > \\ \emptyset \\ \emptyset \end{bmatrix} = 0$$

Der mit C_5 gegebene Co-Trap ist unter der Markierung M_1 somit derart markiert, dass keine der Transitionen durch ihr Feuern eine Markierung produzieren kann, so dass der Co-Trap wieder faktisch als markiert gilt. □

Traps

In S/T-Netzen ist eine Stellenmenge ein Trap, wenn jede Transition, die von einer Stelle des Traps eine Marke konsumiert, auch wieder eine Marke auf einer Stelle des Traps produziert. In S-Pr/T-Netzen sind Traps wie folgt definiert:

Definition 22 (Trap)
Sei $G = (P, T, F, L, n)$ ein S-Pr/T-Netz und sei $D : P \rightarrow MS_+(<N>)$ ein S-Vektor mit Einträgen aus Multimengen über $<N>$. Der S-Vektor D ist ein Trap für das S-Pr/T-Netz G, wenn gilt: Jede Transition t, welche eine Tupel gemäss der Kantenanschrift $L(p,t)$ von einer Stelle $p \in D$ konsumiert, produziert auch wieder ein Tupel gemäss der Kantenanschrift $L(t,p')$ auf einer Stelle $p' \in D$.

Ein Trap D heißt konstant, wenn alle seine Einträge aus Multimengen über Konstanten bestehen. Anderenfalls heißt der Trap D parametrisiert.

Ein Trap D heißt unter einer Markierung M faktisch markiert, wenn gilt: $D^T \bullet M > 0$. Anderenfalls heißt der Trap D unter der Markierung M faktisch nicht markiert. □

Eine Stellenmenge ist in einem S-Pr/T-Netz ein Trap, wenn jede Transition, die ein Tupel von einer Stelle des Traps konsumiert, auch wieder ein entsprechendes Tupel gemäss der Kantenanschriften auf einer Stelle des Traps produziert. Somit kann ein einmal faktisch markierter Trap nicht wieder faktisch leer werden. Sei zur Verdeutlichung das nachfolgende Beispiel gegeben.

Beispiel 9
Für das in Abbildung 3.2 mit Modulus $n = 2$ dargestellte S-Pr/T-Netz G_2 ist der folgende S-Vektor ein mit der Variablen x parametrisierter Trap:

$$D_1 = \begin{bmatrix} < x > \\ < x \oplus 1 > \\ < x \oplus 1 > \end{bmatrix}$$

Der S-Vektor D_2 ist jedoch kein Trap für das S-Pr/T-Netz G_2:

$$D_2 = \begin{bmatrix} < x > \\ < x > \\ < x > \end{bmatrix}$$

Demnach konsumiert die Transition t_1 ein Tupel $< x >$ von der Stelle $p_1 \in D_2$ und markiert die Stelle $p_2 \in D_2$ mit einem Tupel $< x >$. Dies widerspricht jedoch den gegebenen Kantenanschriften. Der S-Vektor D_2 ist somit kein Trap für das S-Pr/T-Netz G_2 (analog für die Transitionen t_2 und t_3).

Sei nun mit D_3 ein zu D_1 korrelierender, konstanter Trap für das S-Pr/T-Netz G_2 gegeben und sei die Markierung des S-Pr/T-Netzes mit M_2 wie folgt spezifiziert:

$$D_3 = \begin{bmatrix} <0> \\ <1> \\ <1> \end{bmatrix} \qquad M_2 = \begin{bmatrix} <0> \\ <1> \\ <1> \end{bmatrix}$$

Der gegebene Trap D_3 gilt unter der Markierung M_2 als faktisch markiert, da Folgendes erfüllt ist:

$${D_3}^T \bullet M_2 = [<0>, <1>, <1>] \bullet \begin{bmatrix} <0> \\ <1> \\ <1> \end{bmatrix} = 3 > 0$$

Auf Grund der Charakteristika von Traps kann die durch D_3 spezifizierte Stellenmenge daher nicht mehr leer werden. □

3.4 Fazit

In diesem Kapitel wurde die Klasse der S-Pr/T-Netze eingehend beleuchtet. Durch Anwendung der Modulo-Operationen $\oplus$ und $\ominus$ werden in S-Pr/T-Netzen die Wertebereiche der Variablen beschränkt. S-Pr/T-Netze besitzen somit einen endlichen Zustandsraum. S-Pr/T-Netze sind daher trotz ihrer Modellierungsmächtigkeit weiterhin mittels der Berechnung von S- und T-Invarianten sowie der Identifikation von Traps und Co-Traps strukturell analysierbar. Auf Grund dieser Vorzüge von S-Pr/T-Netzen bilden diese die Grundlage für die im Rahmen dieser Arbeit entwickelte Klasse der *zeitbewerteten Prädikat/Transitions-Netze*.

4 Zeitbewertete Prädikat/Transitions-Netze

Im Rahmen dieser Arbeit wurde die Klasse der *zeitbewerteten Prädikat/Transitions-Netze*, oder auch kurz *Z-Pr/T-Netze*, entwickelt. Z-Pr/T-Netze erlauben die Modellierung, Simulation und strukturelle Analyse zeitbewerteter Systeme und sind insbesondere in Hinblick auf den Anwendungsbereich sicherheitskritischer Echtzeitsysteme von hohem Interesse.

Grundlage für die im Rahmen dieser Arbeit neu entwickelte Klasse der Z-Pr/T-Netze bilden dabei die in Kapitel 3 vorgestellten S-Pr/T-Netze. Dies begründet sich insbesondere darin, dass die Wertebereiche der Variablen in S-Pr/T-Netzen durch Anwendung der Modulo-Operatoren $\oplus$ und $\ominus$ beschränkt werden und S-Pr/T-Netze somit endlich sind. S-Pr/T-Netze sind daher trotz ihrer Modellierungsmächtigkeit weiterhin strukturell analysierbar. Auf Grund dessen geht die Klasse der Z-Pr/T-Netze strukturell aus den S-Pr/T-Netzen hervor.

Die Modellierung zeitbewerteter Systeme bedingt die Einbindung eines Konzeptes zu Modellierung von Zeit. Neben den damit einhergehenden strukturellen Modifikationen erfordert die Modellierung und Simulation zeitbewerteter Systeme darüber hinaus eine modifizierte Dynamik. Das Eintreten von Zustandsübergängen in zeitbewerteten Systemen ist nicht nur von dem aktuellen Systemzustand, sondern auch von der aktuellen Systemzeit abhängig. Dies wird in Z-Pr/T-Netzen mittels einer restriktiveren Transitionsregel realisiert. Die strukturellen Modifikationen sowie die daraus resultierende Netzklasse der Z-Pr/T-Netze werden zunächst in Kapitel 4.1 detailliert erläutert. Anschließend wird in Kapitel 4.2 eine modifizierte Schaltregel eingeführt und die veränderte Dynamik anhand eines Beispieles verdeutlicht.

Hauptaugenmerk bei der Entwicklung der Z-Pr/T-Netze liegt dabei auf der Anwendbarkeit struktureller Analysemethoden. Im Gegensatz zu einer Vielzahl anderer Ansätze zeiterweiterter Petrinetze [14],[21],[30],[31],[42],[44],[45],[46] sind Z-Pr/T-Netze strukturell analysierbar, so dass eine unmittelbare strukturelle Analyse der Modelle zur Verifikation zeitbewerteter Systeme genutzt werden kann. Die strukturellen Analyseverfahren wie die Berechnung von S- und T-Invarianten sowie die Identifikation von Traps und Co-Traps in Z-Pr/T-Netzen werden in

Kapitel 4.3 erörtert. Anschließend werden in Kapitel 4.4 Techniken vorgestellt, welche bei der Verifikation auf Basis einer strukturellen Analyse von Z-Pr/T-Netzen Anwendung finden. Das Zusammenspiel struktureller Analyseverfahren und dieser Techniken wird anschließend in Kapitel 4.5 anhand von Beispielen erläutert.

4.1 Einführung in Z-Pr/T-Netze

Bezeichne die *Systemzeit* die Zeit in dem zu modellierenden System und die *Modellzeit* die Darstellung der Systemzeit in dem Modell. Entscheidend bei der Modellierung zeitbewerteter Systeme ist somit eine adäquate Darstellung der Systemzeit in dem Modell.

Grundsätzlich kann bei der Modellierung zeiterweiterter Petrinetze unterschieden werden, ob Transitionen zeitlos oder zeitkonsumierend schalten. Eine Modellierung zeitkonsumierender Transitionen wie z.B. bei den Timed Petri Nets [33] (siehe Kap. 2.1.2) hat jedoch zum Nachteil, dass in der graphischen Darstellung der Petrinetze nicht mehr zwangsläufig alle Tupel zu jedem Zeitpunkt sichtbar sind. Dies wiederum bedeutet eine Einbusse bezüglich der Verständlichkeit der Modelle und stellt somit eine potenzielle Fehlerquelle bei der Modellierung komplexer Systeme dar. Aus diesem Grund schalten Transitionen in Z-Pr/T-Netzen zeitlos, wohingegen Zeit vergeht, während ein Tupel auf einer Stelle verweilt.

Eine Realisierung eines derartigen Zeitkonzeptes erfolgt in Z-Pr/T-Netzen mittels der Modellierung *lokaler Modellzeiten*, welche ihrerseits die *globale Modellzeit* implizieren. Jedes (nicht anonyme) Tupel in einem Z-Pr/T-Netz beinhaltet die Variable j. Die Variable j spezifiziert den Zeitpunkt bzgl. der globalen Modellzeit, bis zu welchem das Tupel auf der jeweiligen Stelle verweilen muss. Erst wenn dieser Zeitpunkt erreicht ist, wird das Tupel für Transitionen wieder verfügbar. In Abhängigkeit der Variablen j wird die lokale Modellzeit für jede Stelle definiert. Ist eine Stelle markiert, so ist die lokale Modellzeit dieser Stelle das Minimum aller j-Werte ihrer Markierung. Ist eine Stelle nicht markiert, so ist ihre lokale Modellzeit nicht definiert. Das Minimum aller lokalen Modellzeiten entspricht der globalen Modellzeit. Demnach kann eine Transition t nur dann aktiviert sein, wenn alle Tupel auf den Eingangsstellen der Transition t zu der aktuellen, globalen Modellzeit verfügbar sind. D.h. die lokalen Modellzeiten aller Eingangsstellen der Transition t müssen mit der globalen Modellzeit übereinstimmen.

Petrinetze sind verteilte Systeme. D.h. der verteilte Systemzustand wird durch die Markierung aller Stellen repräsentiert. Die hier vorgenommene Unterscheidung zwischen einer lokalen und einer globalen Modellzeit hat zum Vorteil, dass zum einen die globale Modellzeit nicht explizit modelliert werden muss und zum

anderen die lokal vorliegenden Modellzeiten nicht konsistent gehalten werden müssen. Die Komplexität der Modelle kann somit erheblich reduziert werden.
Die Transitionen in Z-Pr/T-Netzen schalten zeitlos. Zustandsübergänge können in dem zu modellierenden System jedoch sowohl zeitlos als auch zeitkonsumierend sein. Seien im Folgenden solche zeitlosen Zustandsübergänge mit *Events* und solche zeitkonsumierenden Zustandsübergänge mit *Aktionen* bezeichnet. Sowohl Events als auch Aktionen müssen in Z-Pr/T-Netzen mittels zeitlos-schaltender Transitionen modelliert werden. Es existieren daher zwei temporäre Größen, welche in die Klasse der Z-Pr/T-Netze integriert werden – die lokale Modellzeit und die Dauer einer Aktion. Z-Pr/T-Netze sind wie folgt definiert:

Definition 23 (Z-Pr/T-Netz)
Seien I, J, Π *nichtleere, endliche Mengen von Variablen und* $n_i, n_j \in \mathbb{N}$. *Dann heißt ein Tupel* $G = (P, T, F, L, n_i, n_j)$ *zeitbewertetes Prädikat/Transitions-Netz, wenn gilt:*

- P *ist eine endliche Menge von Stellen*
- T *ist eine endliche Menge von Transitionen*
- F *ist eine Flussrelation, für die gilt:*
  - $F = (P \times T) \cup (T \times P)$
  - $P \cap T = \emptyset$
  - $P \cup T \neq \emptyset$
- $L : F \rightarrow < I \oplus N_i, J \oplus N_j, \Pi >$ *ordnet jeder Kante ein Tripel zu, wobei*
  - $i \in I$ *die Dauer der zu modellierenden Aktion repräsentiert.*
  - $j \in J$ *die lokale Modellzeit repräsentiert.*
  - $\pi \in \Pi$ *die Priorität des Tupels repräsentiert.*
- n_i *ist der Modulus der Variablen i mit* $n_i = max\,\{Dauer\ aller\ Aktionen\}+1$
- n_j *ist der Modulus der Variablen j* □

Den Kanten in Z-Pr/T-Netzen sind n-stellige Tupel mit $n \geq 3$ zugeordnet. Die Tupel umfassen die Variablen i, j und π, wobei deren semantische Interpretation fest vorgegeben ist. Die Tupel können jedoch in Abhängigkeit des zu modellierenden Systems um weitere Variablen ergänzt werden. Für Beispiele diesbezüglich sei hier auf Kapitel 5 verwiesen.

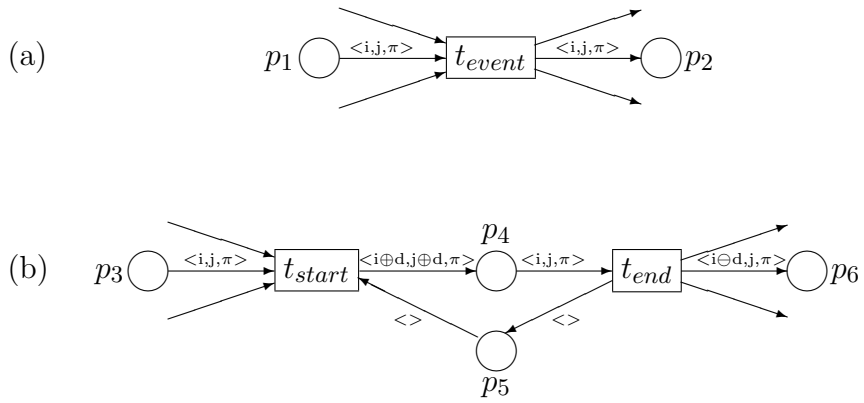


Abbildung 4.1: Events und Aktionen in Z-Pr/T-Netzen

Des Weiteren werden die Wertebereiche der Variablen i und j durch Anwendung der Modulo-Operationen $\oplus$ und $\ominus$ beschränkt. Z-Pr/T-Netze sind endlich und strukturell analysierbar (siehe Kap. 4.3).

Modellierung von Events und Aktionen in Z-Pr/T-Netzen

In Z-Pr/T-Netzen feuern Transitionen zeitlos. Demzufolge müssen sowohl Events als auch Aktionen mittels zeitlos-schaltender Transitionen modelliert werden.

Events bezeichnen zeitlose Zustandsübergänge. Demzufolge lassen sich Events unmittelbar durch Transitionen in Z-Pr/T-Netzen abbilden (siehe Abb. 4.1(a)). Da Events zeitlos sind, haben Transitionen, welche Events modellieren, weder Einfluss auf die Variable i noch die Variable j.

Aktionen bezeichnen hingegen zeitkonsumierende Zustandsübergänge. Sie lassen sich mittels zeitlos-schaltender Transitionen wie in Abbildung 4.1(b) modellieren. Das Feuern der Transitionen t_{start} bzw. t_{end} markiert den Beginn bzw. das Ende der zu modellierenden Aktion, wohingegen das Tupel für die Dauer der zu modellierenden Aktion auf der Stelle p_4 verweilt. Feuert die Transition t_{start}, so (modulo-)inkrementiert sie den Wert der Variablen j um die Dauer der zu modellierenden Aktion. Für die Dauer der zu modellierenden Aktion verweilt das Tupel auf der Stelle p_4. Ist der Zeitpunkt zur Beendigung der zu modellierenden Aktion erreicht, so feuert die Transition t_{end}. Des Weiteren wird der Wert der Variablen i beim Feuern der Transition t_{start} um die Dauer der zu modellierenden Aktion (modulo-)inkrementiert und beim Feuern der Transition t_{end} wieder um die Dauer der zu modellierenden Aktion (modulo-)dekrementiert. Das hat zur Konsequenz, dass die Variable i ausschließlich auf der Stelle p_4 einen Wert ungleich Null besitzt.

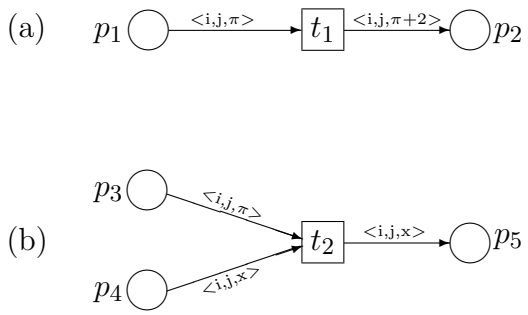


Abbildung 4.2: Manipulation von Prioritäten mittels (a) arithmetischer Operatoren und (b) mittels Wertepropagation

Modellierung von Prioritäten in Z-Pr/T-Netzen

Die Variable π repräsentiert die *Priorität* eines Tupels. Die Modellierung von Prioritäten gemäss Definition 23 erlaubt im Sinne der Modellierung von Echtzeitsystemen sowohl die Repräsentation dynamischer als auch statischer Prioritäten. Sei eine Repräsentation von Prozessen durch eindeutig identifizierbare Tupel gegeben. Die Modellierung dynamischer Prioritäten kann dann mittels Anwendung der arithmetischen Operatoren $+/-$ auf die Variable π (siehe Abb. 4.2 (a)) oder mittels Wertepropagation (siehe Abb. 4.2 (b)) erfolgen. Ist in einem Z-Pr/T-Netz keine Form der Manipulation der Variablen π gegeben, so modelliert das gegebene Z-Pr/T-Netz statische Prioritäten.

Oftmals erfolgt in dem zu modellierenden System eine Assoziation statischer Prioritäten mit den Events und Aktionen des Systems. D.h. die Tupel besitzen alle identische und invariante Prioritäten. Jedoch sind den Transitionen statische Prioritäten zugeordnet. Ist in diesem Fall eine Transition aktiviert, so ist sie dies immer für die mit ihr assoziierten Priorität. Um eine Modellierung von Prioritäten, welche mit den Transitionen eines Z-Pr/T-Netzes assoziiert sind, mit einer Modellierung gemäss Definition 23 in Einklang zu bringen, muss gewährleistet werden, dass die Eingangsstellen der Transitionen entsprechend ihrer Prioritäten markiert sind. Sei dazu beispielsweise eine Transition t_{event} gegeben, welche im Falle einer Aktiviertheit immer mit Priorität $\pi = 3$ aktiviert ist. Dementsprechend müssen die Eingangsstellen der Transition t_{event} immer mit einem Wert $\pi = 3$ markiert werden (siehe Abb. 4.3(a)). Zur Reduzierung der Komplexität der Modelle erfolgt oftmals eine Modellierung ohne die Variable π in den Kantenanschriften, wohingegen die Transitionen mit der mit ihnen assoziierten Priorität annotiert werden (siehe Abb. 4.3(b)).

Im nachfolgenden Kapitel 4.2 wird nun die Dynamik in Z-Pr/T-Netzen erläutert.

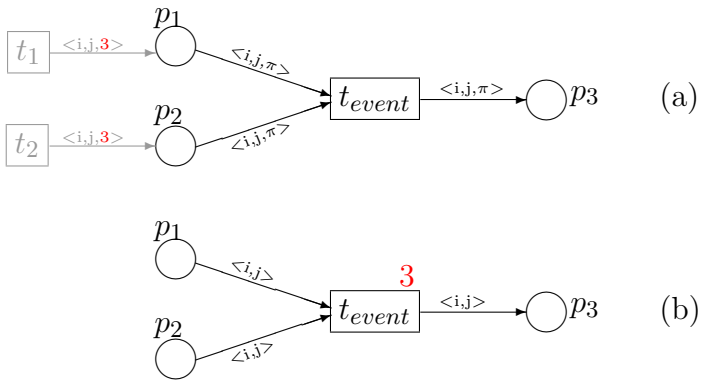


Abbildung 4.3: Modellierung transitionsassoziierter Prioritäten

4.2 Dynamik in Z-Pr/T-Netzen

Ziel dieses Ansatzes ist die Anwendung von Z-Pr/T-Netzen zur Modellierung, Simulation und strukturellen Analyse im Anwendungsbereich sicherheitskritischer Echtzeitsysteme. Seien die Zustände des zu modellierenden Systems mit *Systemzuständen* bezeichnet. Das Eintreten von Zustandsübergängen hängt in zeitbewerteten Systemen nicht nur vom aktuellen Systemzustand, sondern auch von der aktuellen Systemzeit ab. Dementsprechend muss die zeitliche Abfolge von Zustandsübergängen in Z-Pr/T-Netzen korrekt abgebildet werden. D.h. jedem Systemzustand muss genau ein Zustand in dem Z-Pr/T-Netz entsprechen. Dies erfolgt in Z-Pr/T-Netzen mittels Definition einer modifizierten Transitionsregel.

Eine Restriktion der Dynamik mittels modifizierter Schaltregel hat im Gegensatz zu einer expliziten Modellierung eines Mechanismus, welcher eine Taktung aller Zustandsübergänge vornimmt, den Vorteil, dass die Komplexität der Modelle erheblich reduziert wird.

Seien im Weiteren S- und T-Vektoren in Z-Pr/T-Netzen wie in Definition 11 gegeben und sei des Weiteren in Z-Pr/T-Netzen Vor- und Nachbereich einer Transition bzw. Stelle wie in Definition 10 gegeben. Die Dynamik in Z-Pr/T-Netzen wird dann durch die nachfolgende Definition spezifiziert.

Definition 24 (aktiviert)
Sei $U = \{t_1, t_2, ..., t_n\}$ eine Menge von Transitionen. Eine Transition $t_k \in U$ ist im herkömmlichen Sinne aktiviert, wenn gilt:

$$\forall p \in {}^\bullet t_k : \quad M(p) =< i_k, j_k, \pi_k > \quad \wedge \quad M(p) \geq L(p, t_k)$$

Sei $U' = \{t_k, t_l\}$ mit $U' \subseteq U$ und bezeichne NIL die Menge aller Warteaktionen. Die Transition t_k ist dann aktiviert, wenn eine der folgenden Bedingungen gilt:

1. $j_k < j_l$ $\forall 1 \leq k, l \leq n$
2. $j_k = j_l \quad \wedge \quad \pi_k > \pi_l$ $\forall 1 \leq k, l \leq n$
3. $j_k = j_l \quad \wedge \quad \pi_k = \pi_l \quad \wedge \quad i_k > i_l$ $\forall 1 \leq k, l \leq n$
4. $j_k = j_l \quad \wedge \quad \pi_k = \pi_l \quad \wedge \quad i_k = i_l$

   $\wedge \quad t_l \in NIL$ $\forall 1 \leq k, l \leq n$ □

Sei eine Menge von Transitionen $U = \{t_1, t_2, ..., t_n\}$ gegeben, welche im herkömmlichen Sinne aktiviert sind. Eine Transition $t_k \in U$ ist in einem Z-Pr/T-Netz genau dann aktiviert, wenn zusätzlich eine der vier obigen Schaltregeln erfüllt ist. Dabei finden die folgenden vier Prinzipien Anwendung:

Prinzip 1:
Es sind zunächst immer diejenigen Transitionen aktiviert, deren Eingangsstellen mit dem niedrigsten Wert für die Variable j markiert sind.

Die Werte der Variablen j spezifizieren die lokale Modellzeit einer Stelle und geben an, wann ein Tupel in Bezug auf die globale Modellzeit für Transitionen wieder verfügbar ist. Somit kann eine Transition in Z-Pr/T-Netzen nur dann aktiviert sein, wenn alle Tupel auf den Eingangsstellen der Transition zu der aktuellen, globalen Modellzeit verfügbar sind. D.h. die lokalen Modellzeiten aller Eingangsstellen der Transition müssen mit der globalen Modellzeit übereinstimmen. Eine derartige Schaltregel hat zur Konsequenz, dass die Abweichungen der lokalen Modellzeiten von der globalen Modellzeit so minimal wie möglich gehalten werden.

Prinzip 2:
Existieren mehrere Transitionen, deren Eingangsstellen mit dem niedrigsten Wert für die Variable j markiert sind, dann sind von diesen Transitionen diejenigen Transitionen aktiviert, deren Eingangsstellen mit dem höchsten Wert für die Variable π markiert sind. Eine Überprüfung der Prioritäten erfolgt nur dann, wenn die obige Prämisse erfüllt ist, da anderenfalls Prinzip 1 Anwendung findet.

Sind zu einem Zeitpunkt bzgl. der globalen Modellzeit mehrere Transitionen im herkömmlichen Sinne aktiviert, so feuern zunächst diejenigen Transitionen, welche für die höchste Priorität aktiviert sind.

Prinzip 3:
Existieren mehrere Transitionen, deren Eingangsstellen mit dem niedrigsten Wert für die Variable j und dem höchsten Wert für die Variable π markiert sind, dann sind von diesen Transitionen diejenigen Transitionen aktiviert, deren Eingangsstellen mit dem größten Wert für die Variable i markiert sind. Eine Überprüfung der Variablen i erfolgt nur bei Erfüllung der oben genannten Prämisse, da anderenfalls Prinzip 1 oder Prinzip 2 Anwendung finden.

Die Variable i wird ausschließlich von Transitionen manipuliert, welche Aktionen modellieren. Ist eine Transition t im herkömmlichen Sinne für einen Wert $i \neq 0$ aktiviert, so handelt es sich bei der Transition t um eine Transition, welche das Ende einer Aktion signalisiert. Anderenfalls modelliert die Transition t ein Event oder den Beginn einer Aktion. Demnach erhalten zunächst diejenigen Transitionen Vorrang, welche die am weitesten fortgeschrittenen Aktionen modellieren.

Prinzip 4:
Existieren mehrere Transitionen, deren Eingangsstellen mit dem niedrigsten Wert für die Variable j und den höchsten Werten für die Variablen π und i markiert sind, so haben von diesen Transitionen diejenigen Transitionen Vorrang, welche keine Warteaktionen modellieren. Ist die Prämisse nicht erfüllt, dann findet eines der obigen Prinzipien Anwendung.

Das Modellieren von Warteaktionen dient dem Inkrementieren der lokalen Modellzeiten. Transitionen, welche Events oder Aktionen modellieren, haben daher Vorrang vor denjenigen Transitionen, welche Warteaktionen modellieren.

Die obigen vier Prinzipien gelten auch für Transitionen, welche Warteaktionen modellieren. Existieren mehrere Transitionen, deren Eingangsstellen mit dem niedrigsten Wert für die Variable j und den höchsten Werten für die Variablen π und i markiert sind und modellieren diese Transitionen darüber hinaus Warteaktionen, so kann es bei Anwendung von Prinzip 4 jedoch zu einem Konflikt zwischen diesen Transitionen kommen. Ebenso kann es unter Anwendung von Prinzip 4 zu einem Konflikt kommen, wenn mindestens zwei Transitionen existieren, deren Eingangsstellen mit dem niedrigsten Wert für die Variable j und den höchsten Werten für die Variablen π und i markiert sind, und diese Transitionen keine Warteaktionen modellieren.

Definition 25 (Schalten, Folgemarkierung)
Sei M eine Markierung, $t \in T$, $p \in P$, $k, l, m \in \mathbb{Z}$ und i,j,π Variablen. Falls t unter M aktiviert ist, kann t schalten (bzw. feuern) für $< i, j, \pi >=< k, l, m >$ und dadurch in eine Folgemarkierung M' überführen mit

$$M'(p) = \begin{cases} M(p) - L(p,t) & falls \;\; p \in {}^\bullet t \text{ und } p \notin t^\bullet \\ M(p) + L(t,p) & falls \;\; p \in t^\bullet \text{ und } p \notin {}^\bullet t \\ M(p) - L(p,t) + L(t,p) & falls \;\; p \in {}^\bullet t \cap t^\bullet \\ M(p) & falls \;\; p \in P \setminus ({}^\bullet t \cap t^\bullet) \end{cases},$$

wobei das i, j und π aus L(p,t) bzw. L(t,p) durch das k,l und m aus M(p) substituiert sind. □

Die Dynamik in Z-Pr/T-Netzen wird anhand des nachfolgenden Beispiels verdeutlicht (vgl. [32]).

Beispiel 10
Sei das in Abbildung 4.4 dargestellte Z-Pr/T-Netz G_3 gegeben (vgl. [32]).
Der Modulus n_i der Variablen i in dem Z-Pr/T-Netz G_3 ist $n_i = 3$. Die Variable i wird somit auf einem Wertebereich $[0, 1, 2]$ abgebildet. Der Modulus n_j der Variablen j sei beliebig, aber hinreichend groß gewählt.
Des Weiteren ist durch das zugrundeliegende Beipiel in [32] vorgegeben, dass alle Tupel die gleiche, statische Priorität besitzen. D.h. die Werte der Variablen π sind für alle Tupel identisch und invariant. Die Werte der Variablen π haben somit in dem gegebenen Beispiel keinen Einfluss auf die Schaltreihenfolge der Transitionen. Daher wurde das Z-Pr/T-Netz G_3 ohne die Variable π modelliert und die Komplexität des Z-Pr/T-Netzes somit reduziert (Technik I, Kap. 4.4).
Die folgende Schaltsequenz repräsentiert dann einen Pfad im Erreichbarkeitsgraphen des Z-Pr/T-Netzes G_3:

$$\begin{bmatrix} 4\cdot < 0,0 > \\ \emptyset \\ \emptyset \\ <> \\ \emptyset \\ \emptyset \\ <> \\ <> \\ \emptyset \\ \emptyset \\ <> \\ \emptyset \\ \emptyset \\ \emptyset \end{bmatrix} \xrightarrow{t_{1.0}} \begin{bmatrix} 2\cdot < 0,0 > \\ \emptyset \\ \emptyset \\ \emptyset \\ < 2,2 > \\ \emptyset \\ <> \\ <> \\ \emptyset \\ \emptyset \\ <> \\ \emptyset \\ \emptyset \\ \emptyset \end{bmatrix} \xrightarrow{t_{2.0}} \begin{bmatrix} < 0,0 > \\ \emptyset \\ \emptyset \\ \emptyset \\ < 2,2 > \\ \emptyset \\ <> \\ \emptyset \\ < 1,1 > \\ \emptyset \\ <> \\ \emptyset \\ \emptyset \\ \emptyset \end{bmatrix} \xrightarrow{nil_{1.0}} \begin{bmatrix} \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \\ < 2,2 > \\ \emptyset \\ <> \\ \emptyset \\ < 1,1 > \\ \emptyset \\ <> \\ < 1,1 > \\ \emptyset \\ \emptyset \end{bmatrix}$$

$$\xrightarrow{nil_{1.1}} \begin{bmatrix} <0,1> \\ \emptyset \\ \emptyset \\ \emptyset \\ <2,2> \\ \emptyset \\ <> \\ \emptyset \\ <1,1> \\ \emptyset \\ <> \\ \emptyset \\ \emptyset \\ \emptyset \end{bmatrix} \xrightarrow{t_{2.1}} \begin{bmatrix} <0,1> \\ \emptyset \\ <0,1> \\ \emptyset \\ <2,2> \\ \emptyset \\ <> \\ <> \\ \emptyset \\ \emptyset \\ <> \\ \emptyset \\ \emptyset \\ \emptyset \end{bmatrix} \xrightarrow{t_{4.0}} \begin{bmatrix} <0,1> \\ \emptyset \\ \emptyset \\ \emptyset \\ <2,2> \\ \emptyset \\ <> \\ <> \\ \emptyset \\ <2,3> \\ \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \end{bmatrix} \xrightarrow{t_{2.0}} \begin{bmatrix} \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \\ <2,2> \\ \emptyset \\ <> \\ \emptyset \\ <1,2> \\ <2,3> \\ \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \end{bmatrix}$$

$$\xrightarrow{t_{1.2}} \begin{bmatrix} \emptyset \\ <0,2> \\ \emptyset \\ <> \\ \emptyset \\ \emptyset \\ <> \\ \emptyset \\ <1,2> \\ <2,3> \\ \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \end{bmatrix} \xrightarrow{t_{2.1}} \begin{bmatrix} \emptyset \\ <0,2> \\ <0,2> \\ <> \\ \emptyset \\ \emptyset \\ <> \\ <> \\ \emptyset \\ <2,3> \\ \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \end{bmatrix} \xrightarrow{t_{3.0}} \begin{bmatrix} \emptyset \\ \emptyset \\ <0,2> \\ <> \\ \emptyset \\ <2,4> \\ \emptyset \\ <> \\ \emptyset \\ <2,3> \\ \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \end{bmatrix} \xrightarrow{\cdots}$$

Die obige Schaltsequenz verdeutlicht die modifizierte Dynamik in Z-Pr/T-Netzen in Hinblick auf die modifizierte Schaltregel. Z-Pr/T-Netze eignen sich somit nicht nur zur Modellierung, sondern auch zur Simulation zeitbewerteter Systeme. □

Im nachfolgenden Kapitel werden nun Verfahren zur strukturellen Analyse von Z-Pr/T-Netzen vorgestellt.

4.3 Strukturelle Analyse in Z-Pr/T-Netzen

Ein großer Vorteil von Petrinetzen ist – neben ihrer graphischen Visualisierbarkeit – ihre formale Basis, welche eine strukturelle Analyse der Modelle erlaubt. Die mittels struktureller Analyse ermittelten Strukturkomponenten können unmittelbar zur Verifikation von Systemeigenschaften genutzt werden. Die Synthese von temporären Konzepten und von Petrinetzen stellt daher einen vielverspre-

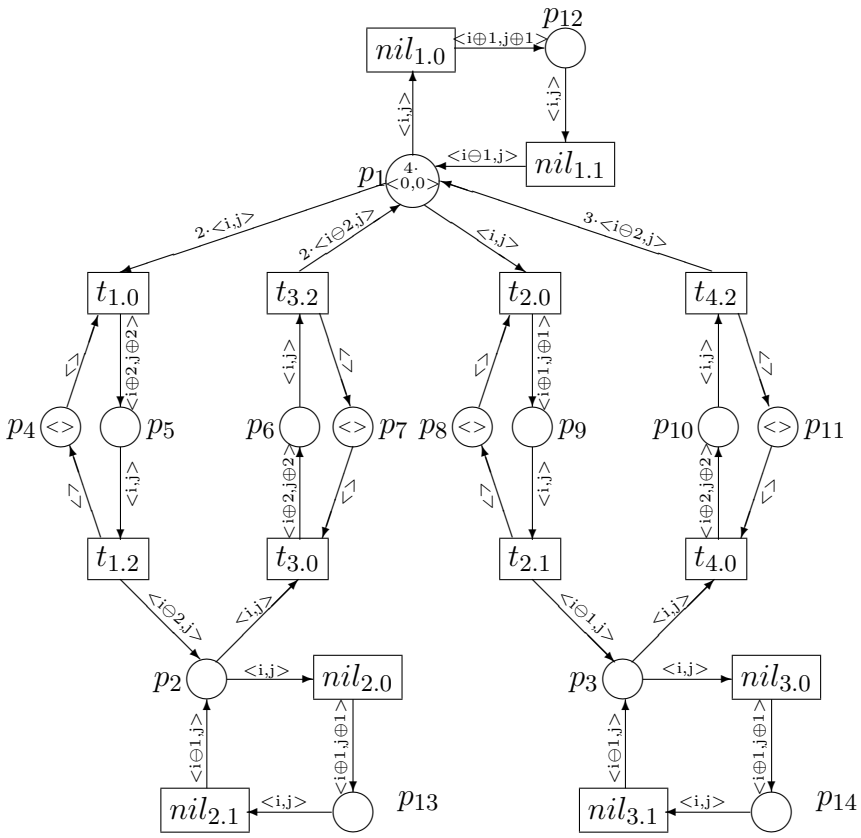


Abbildung 4.4: Z-Pr/T-Netz G_3 (vgl. [32])

chenden Ansatz dar für die Entwicklung eines Formalismus zur Modellierung, Simulation, vor allem aber zur Verifikation in der Anwendungsdomäne sicherheitskritischer Echtzeitsysteme.
Im Rahmen dieser Arbeit wurde daher die Klasse der Z-Pr/T-Netze entwickelt, welche temporäre Konzepte in Petrinetze integriert und neben der Modellierung und Simulation auch die Verifikation der Modelle auf Basis einer strukturellen Analyse der Modelle ermöglicht. Die hier vorgestellten Verfahren der strukturellen Analyse von Z-Pr/T-Netzen sind die Berechnung von S- und T-Invarianten (siehe Kap. 4.3.1) sowie die Identifikation von Traps und Co-Traps (siehe Kap. 4.3.2).
Die Berechnung von S- und T-Invarianten sowie die Identifikation von Traps und Co-Traps in Z-Pr/T-Netzen erfolgt analog zu der Berechnung von S- und T-Invarianten sowie der Identifikation von Traps und Co-Traps in S-Pr/T-Netzen. Jedoch wird in Z-Pr/T-Netzen auf Multimengen über konstanten und variablen Tripeln gerechnet und nicht wie in S-Pr/T-Netzen auf Multimengen über konstanten und variablen einstelligen Tupeln.

4.3.1 S- und T-Invarianten in Z-Pr/T-Netzen

Sei die Inzidenzmatrix in Z-Pr/T-Netzen analog zu der Inzidenzmatrix in S-Pr/T-Netzen definiert (siehe Def. 13). Dann sind S- und T-Invarianten in Z-Pr/T-Netzen wie folgt definiert:

Definition 26 (S-Invariante)
Sei $G = (P, T, F, L, n_i, n_j)$ ein Z-Pr/T-Netz. Ein S-Vektor S heißt S-Invariante von G, wenn gilt:

$$S^T \bullet [G(p,t)] = 0^T \ .$$

Eine S-Invariante S heißt konstant, wenn alle ihre Einträge aus Multimengen über Konstanten bestehen. Anderenfalls heißt sie parametrisiert. □

Auch in Z-Pr/T-Netzen gilt für S-Invarianten, dass die gewichtete Anzahl der Tupel auf den Stellen der S-Invarianten konstant ist. Die strukturelle Analyse von Z-Pr/T-Netzen mittels der Berechnung von S-Invarianten kann z.B. zum Nachweis der Nicht-Erreichbarkeit einer Markierung genutzt werden (siehe Bsp. 12).

Definition 27 (T-Invariante)
Sei $G = (P, T, F, L, n_i, n_j)$ ein Z-Pr/T-Netz. Ein T-Vektor U heißt T-Invariante von G, wenn gilt:

$$[G(p,t)] \circ U = 0 \ .$$

Eine T-Invariante heißt konstant, wenn alle Einträge aus Multimengen über Konstanten bestehen. Anderenfalls heißt die T-Invariante parametrisiert. □

Auch in Z-Pr/T-Netzen spezifizieren T-Invarianten Schaltsequenzen zur Reproduktion einer Markierung. Mittels T-Invarianten kann in Z-Pr/T-Netzen z.B. das Nicht-Feuern einer Transition nachgewiesen werden (siehe Bsp. 11).

4.3.2 Traps und Co-Traps in Z-Pr/T-Netzen

Auch Traps und Co-Traps sind in Z-Pr/T-Netzen analog zu Traps und Co-Traps in S-Pr/T-Netzen definiert. Jedoch ist ein Trap bzw. ein Co-Trap in einem Z-Pr/T-Netz ein S-Vektor mit Multimengen über konstanten und variablen Tripeln als Einträgen und nicht wie in S-Pr/T-Netzen ein S-Vektor mit Multimengen über konstanten und variablen einstelligen Tupeln als Einträgen (vgl. Def. 21 und Def. 22).

Definition 28 (Co-Trap)
Sei $G = (P, T, F, L, n_i, n_j)$ ein Z-Pr/T-Netz und sei des Weiteren $C : P \rightarrow MS_+(<N>)$ ein S-Vektor mit Einträgen aus Multimengen über <N>.
Der S-Vektor C ist ein Co-Trap für das Z-Pr/T-Netz G, wenn gilt: Jede Transition t, welche eine Stelle $p \in C$ mit einem Tupel gemäss der Kantenanschrift L(t,p) markiert, konsumiert auch ein Tupel gemäss der Kantenanschrift L(p',t) von der Stelle $p' \in C$.

Ein Co-Trap heißt konstant, wenn alle seine Einträge aus Multimengen über Konstanten bestehen. Anderenfalls heißt der Co-Trap parametrisiert.

Ein Co-Trap C ist unter einer Markierung M faktisch markiert, wenn gilt: $C^T \bullet M > 0$. Anderenfalls heißt der Co-Trap C faktisch nicht markiert unter M. □

Definition 29 (Trap)
Sei $G = (P, T, F, L, n_i, n_j)$ ein Z-Pr/T-Netz und sei $D : P \rightarrow MS_+(<N>)$ ein S-Vektor mit Einträgen aus Multimengen über <N>.
Der S-Vektor D ist ein Trap für das Z-Pr/T-Netz G, wenn gilt: Jede Transition t, welche eine Tupel gemäss der Kantenanschrift L(p,t) von einer Stelle $p \in D$ konsumiert, produziert auch wieder ein Tupel gemäss der Kantenanschrift L(t,p') auf einer Stelle $p' \in D$.

Ein Trap D heißt konstant, wenn alle seine Einträge aus Multimengen über Konstanten bestehen. Anderenfalls heißt der Trap D parametrisiert.

Ein Trap D heißt unter einer Markierung M faktisch markiert, wenn gilt: $D^T \bullet M > 0$. Anderenfalls heißt der Trap D unter der Markierung M faktisch nicht markiert. □

Mit Hilfe der Identifikation faktisch nicht markierter Co-Traps kann den Ursachen für das Nicht-Feuern von Transitionen nachgegangen werden. Die Identifikation faktisch nicht markierter Co-Traps ist bei Problemstellungen bzgl. der Nichterreichbarkeit von Markierungen durch Verklemmungen erfolgsversprechend (siehe Kap. 5.1 und Kap. 5.2).

4.4 Techniken bei der strukturellen Analyse in Z-Pr/T-Netzen

In der Praxis hat das Arbeiten mit Z-Pr/T-Netzen bei der Verifikation in zeitbewerteten Systemen gezeigt, dass eine unmittelbare strukturelle Analyse der Modelle nicht zwangsläufig zum Erfolg führt.

Sei beispielsweise die Nicht-Erreichbarkeit einer Markierung zu beweisen, wobei die Erreichbarkeit eben dieser Markierung durch die zugrundeliegende Transitionsregel unterbunden wird. Aspekte der Dynamik sind jedoch strukturell nicht analysierbar. Die Resultate einer unmittelbaren strukturellen Analyse des Z-Pr/T-Netzes können sich in diesem Fall somit als nicht aussagekräftig bezüglich des zu führenden Beweises herausstellen.

Im Folgenden werden daher Techniken vorgestellt, welche sich bei der Verifikation in zeitbewerteten Systemen als sachdienlich erwiesen haben. Führt eine direkte strukturelle Analyse eines Z-Pr/T-Netzes nicht unmittelbar zum Erfolg, so kann eine Transformation des zu analysierenden Z-Pr/T-Netzes gemäss dieser Techniken dazu führen, dass die strukturelle Analyse des resultierenden Z-Pr/T-Netzes die gewünschten Ergebnisse erzielt.

4.4.1 Technik I: Simulative Inspektion

Die modifizierte Schaltregel in Z-Pr/T-Netzen erschwert das analytische Beweisen, erleichtert jedoch das simulative Inspizieren der Modelle.

Mittels *simulativer Inspektion* wird das Schaltverhalten von Transitionen unter der in Z-Pr/T-Netzen geltenden Schaltregel untersucht. Kann eine Transition ausgehend von einer gegebenen Anfangsmarkierung nie schalten, so kann das gegebene Z-Pr/T-Netz durch Eliminierung eben dieser Transition und aller von ihr abhängigen Netzteile stark vereinfacht werden. Dadurch wird nicht nur die Komplexität des zu analysierenden Z-Pr/T-Netzes, sondern auch des zu führenden Beweises reduziert (siehe Bsp. 11 und 12).

Regel 1: (Eliminieren von NIL-Transitionen)
Sei die Stelle p die Eingangsstelle einer NIL-Transition nil und besitze die Stelle p des Weiteren mindestens eine Ausgangstransition t, welche ein Event modelliert. Dann kann die Transition nil mit allen von ihr abhängigen Netzteile eliminiert werden.

Dies begründet sich darin, dass bei einer Markierung der Stelle p sowohl die Transition *nil* als auch die Transition *t* im herkömmlichen Sinne aktiviert sind. Gemäss des vierten Prinzipes der Schaltregel erhält jedoch immer die Transition *t* Vorrang vor der Transition *nil*. Da die Transition *t* darüber hinaus zeitlos schaltet, kann die Transition *nil* auch bei einer mehrfachen Markierung der Stelle p nie schalten. Die Transition *nil* und alle von ihr abhängigen Netzteile können somit eliminiert werden.
Besitze die Eingangsstelle p einer NIL-Transition *nil* eine Transition t im Nachbereich, welche eine Aktion und kein Event modelliert. Eine Eliminierung der

Transition *nil* ist auch dann u.U. möglich. Jedoch lassen sich in diesem Fall keine allgemeingültigen Aussagen darüber treffen, da ein Feuern der Transition *nil* nicht grundsätzlich ausgeschlossen werden kann.

Regel 2: (Eliminieren von Variablen in den Kantenanschriften)
In Z-Pr/T-Netzen sind die Kantenanschriften als Tripel der Form $< i, j, \pi >$ gegeben. Die Werte dieser Variablen bestimmen gemäss der vier Prinzipien der Transitionsregel die Schaltreihenfolge der Transitionen. Findet in einem Z-Pr/T-Netz eines der vier Prinzipien der Schaltregel keine Anwendung, so kann das Z-Pr/T-Netz ohne die entsprechende Variable in den Kantenanschriften dargestellt werden.

Sei beispielsweise die Priorität aller Tupel identisch und invariant. Das zweite Prinzip der Schaltregel findet somit in dem gegebenen Fall nie Anwendung. Das Z-Pr/T-Netz kann daher ohne die Variable π in den Kantenanschriften modelliert werden, da diese keinen Einfluss auf die Schaltreihenfolge der Transitionen hat.

Regel 3: (Eliminieren von Transitionen)
Sei zu beweisen, dass eine Transition t ausgehend von einer Anfangsmarkierung M_0 nicht feuern kann. Dann kann das zu analysierende Z-Pr/T-Netz ohne die Transition t und aller von ihr abhängigen Netzteile modelliert werden. Für das aus diesen Änderungen resultierende Z-Pr/T-Netz ist dann zu beweisen, dass ausgehend von der Anfangsmarkierung M_0 keine Markierung erreichbar ist, so dass die Transition t hätte schalten können.

Darüber hinaus kann das zu analysierende Z-Pr/T-Netz stark vereinfacht werden, wenn eine Transition nur eingeschränkt schalten kann. Mittels simulativer Inspektion werden diejenigen Teile im Erreichbarkeitsgraphen des Z-Pr/T-Netzes identifiziert, welche eliminiert werden können, ohne die Allgemeingültigkeit des zu führenden Beweises einzuschränken. Ist z.B. durch die gegebenen Prioritäten eine bestimmte Schaltreihenfolge und Schalthäufigkeit von Transitionen vorgegeben, so kann eine Modifizierung der Kantenanschriften oder eine veränderte Anfangsmarkierung Erfolg bringen. Hierbei ist jedoch zu beachten, dass die Gültigkeit des zu führenden Beweises durch diese Modifikationen nicht eingeschränkt wird. Derartige Modifikationen hängen jedoch sowohl von der Struktur des zu analysierenden Z-Pr/T-Netzes als auch von dem zu führenden Beweis ab. Es lassen sich daher diesbezüglich keine allgemeingültigen Regeln formulieren. Die Beispiele 11 und 12 verdeutlichen jedoch das zugrundeliegende Prinzip derartiger Modifikationen.

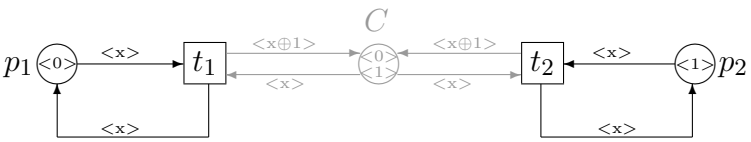


Abbildung 4.5: Modellierung von Synchronabständen

4.4.2 Technik II: Strukturelle Implementierung von Synchronabständen

Die in Z-Pr/T-Netzen geltende Schaltregel hat starken Einfluss auf das Systemverhalten des zu analysierenden Modells. Die Schaltregel spielt in einem Z-Pr/T-Netz somit eine wesentliche Rolle bei z.B. der Erreichbarkeit bzw. Nicht-Erreichbarkeit von Markierungen. Aspekte der Dynamik sind jedoch mittels struktureller Analyseverfahren wie der Berechnung von S- und T-Invarianten sowie der Identifikation von Traps und Co-Traps nicht analysierbar.

Die folgende Technik sieht daher ein strukturelle Ergänzung des zu analysierenden Z-Pr/T-Netzes um Aspekte der Dynamik mittels Implementierung sogenannter *Synchronabstände* vor. Dies hat zur Konsequenz, dass die strukturelle Analyse des resultierenden Z-Pr/T-Netzes diese zusätzlichen Informationen bzgl. der Synchronabstände inkludiert.

Mittels der Variablen j werden in Z-Pr/T-Netzen die lokalen Modellzeiten modelliert. Hinsichtlich der Schaltreihenfolge der Transitionen sind in einem Z-Pr/T-Netz gemäss des ersten Prinzips der Transitionsregel jedoch lediglich die Differenzen der j-Werte relevant. Dies kann auch mit Hilfe von Synchronabständen modelliert werden. Der Synchronabstand zwischen zwei Transitionen gibt an, wie stark die Schalthäufigkeiten eben dieser Transitionen voneinander abweichen dürfen. Zwei Z-Pr/T-Netze sind bzgl. ihres Systemverhaltens äquivalent, wenn ihre Synchronabstände äquivalent sind.

Eine Technik bei der Verifikation in Z-Pr/T-Netzen ist es daher, das zu analysierende Z-Pr/T-Netz G in ein Z-Pr/T-Netz G' zu überführen, welches die Synchronabstände des Z-Pr/T-Netzes G strukturell implementiert. In dem Z-Pr/T-Netz G' sind somit Aspekte der Dynamik strukturell eingebettet und somit auch strukturell analysierbar. Die strukturelle Analyse des Z-Pr/T-Netzes G' inkludiert somit Informationen bzgl. dynamischer Aspekte des Systems und resultiert somit unter Umständen bzgl. des zu führenden Beweises in aussagekräftigeren Ergebnissen. Diese Vorgehensweise ist erfolgsversprechend, wenn das zu analysierende Z-Pr/T-Netz für die zu beweisende Problemstellung unterbesetzt ist.

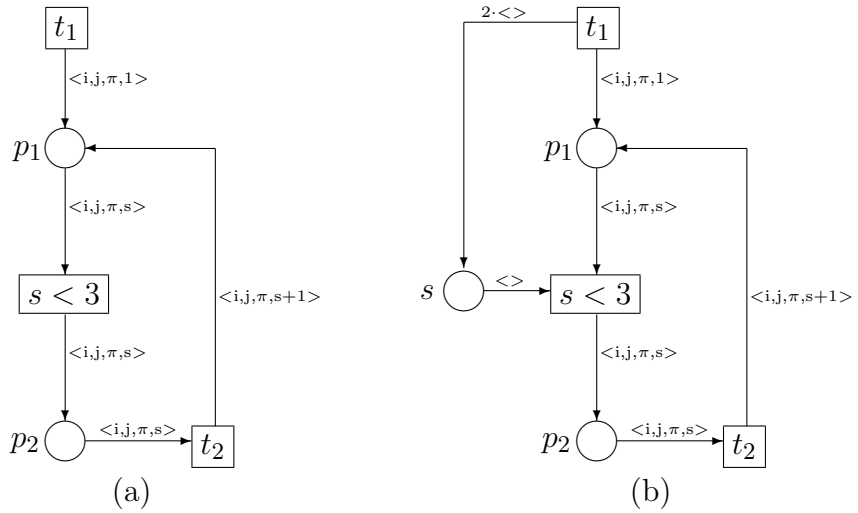


Abbildung 4.6: Strukturelle Implementierung von Guards

Seien beispielsweise zwei Transitionen t_1 und t_2 wie in Abbildung 4.5 gegeben. Sei des Weiteren gefordert, dass die Schalthäufigkeit der Transitionen t_1 und t_2 maximal um eins variiert. Dann implementiert die Stelle C eben diesen Synchronabstand der Transitionen t_1 und t_2. Da die Schaltreihenfolge der Transitionen durch die Implementierung des Synchronabstandes realisiert wird, ist eine Modellierung der Variablen i, j und π nicht erforderlich. Die Variable x dient lediglich der eindeutigen Identifizierung der Tupel. Wie die Technik der Einbettung von Synchronabständen gewinnbringend eingesetzt werden kann, zeigt das Beispiel 11.

4.4.3 Technik III: Strukturelle Implementierung von Guards

Eine weitere, im Rahmen dieser Arbeit angewandte Technik ist die strukturelle Implementierung sogenannter *Guards*. Guards spezifizieren für eine Transition in Form von Gleichungen oder Ungleichungen eine zusätzliche Bedingung für das Feuern eben dieser Transition und können durch eine zusätzliche Stelle strukturell umgesetzt werden.

Sei beispielsweise ein Z-Pr/T-Netz wie in Abbildung 4.6(a) gegeben. Dann kann der Guard $s < 3$ durch eine zusätzliche Stelle s wie in Abbildung 4.6(b) strukturell implementiert werden. Eine derartige Vorgehensweise erfolgte auch bei der Verifikation des EDFs und des PIPs in Kapitel 5 mit Einfügen der zusätzlichen Stellen C und $\neg C$ beim EDF (siehe Abb. 5.2) und Einfügen der Stellen C, CS und CS' beim PIP (siehe Abb. 5.5).

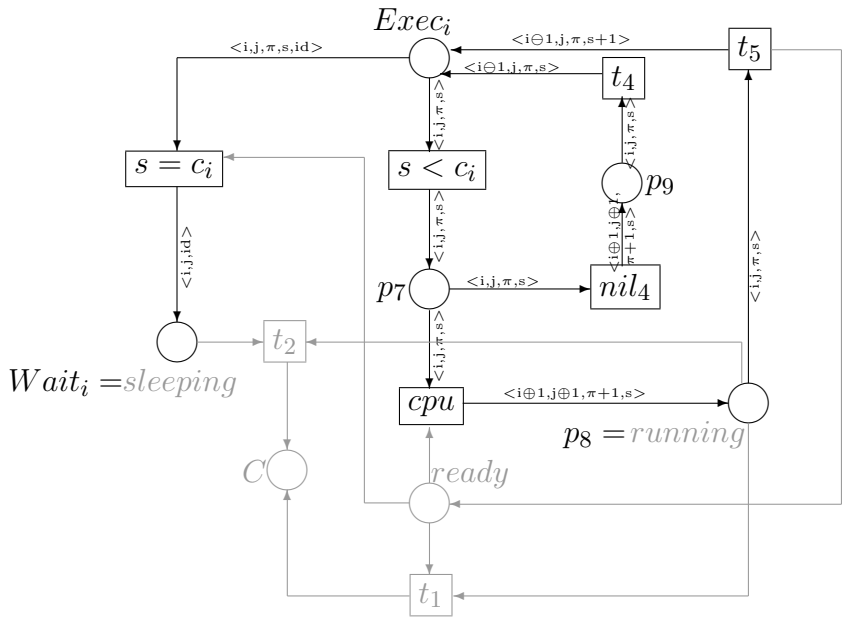


Abbildung 4.7: Modellierung von Systeminvarianten als eingebettetes Teilnetz

4.4.4 Technik IV: Konstruktion logischer Formeln (Systeminvarianten) als eingebettetes Teilnetz

Oftmals ist die Gültigkeit bzw. Nicht-Gültigkeit sogenannter Systeminvarianten zu beweisen, welche in Form logischer Formeln gegeben sind (siehe Kap. 5.2.5). Um einen Nachweis der Gültigkeit bzw. Nicht-Gültigkeit einer derartigen Systeminvarianten führen zu können, müssen zunächst die Prädikate der Systeminvarianten in Form von Stellen in dem Z-Pr/T-Netz modelliert und gemäss ihrer logischen Verknüpfungen mittels Transitionen verbunden werden.
Sei beispielsweise ein Z-Pr/T-Netz wie in Abbildung 4.7 gegeben, welches den Zugriff von Prozessen auf einen Prozessor modelliert. Die Prozesse werden in dem gegebenen Z-Pr/T-Netz durch Tupel repräsentiert. Sei nun eine Systeminvariante in Abhängigkeit der Prädikate *ready*, *running* und *sleeping* wie folgt definiert:

$$C \equiv running \wedge (ready \vee sleeping) \equiv (running \wedge ready) \vee (running \wedge sleeping)$$

Dabei bezeichne *running* den Zustand, wenn ein Prozess Zugriff auf den Prozssor hat, *ready* den Zustand, wenn ein Prozess rechenbereit ist, und *sleeping* den Zustand, wenn ein Prozess seine Ausführung bereits beendet hat. Die Stelle p_8 kann eindeutig mit dem Zustand *running* und die Stelle $Wait_i$ mit dem Zustand *sleeping* identifiziert werden. Da in dem gegebenen Z-Pr/T-Netz jedoch keine Stelle existiert, welche dem Prädikat *ready* entspricht, muss das Z-Pr/T-Netz um die entsprechende Stellen ergänzt werden. Gemäss der logischen Verknüpfung der

Prädikate muss das zu analysierende Z-Pr/T-Netz dann um ein entsprechendes Teilnetz wie in Abbildung 4.7 ergänzt werden.
Nach eben diesem Schema wurden in Kapitel 5.2.5 die Stellen *idle*, *ready*, *running*, *blockedBy* und ¬ *SystInvar* eingefügt und mittels der Transitionen t_9, t_{10} und t_{11} logisch miteinander verknüpft.
Hierbei gilt zu beachten, dass die Stellen, welche zur Einbettung der zu beweisenden Systeminvarianten hinzugefügt wurden, keine Invarianten im petrinetztheoretischen Sinne sind. D.h. die Gültigkeit bzw. Nicht-Gültigkeit sogenannter Systeminvarianten muss darüber hinaus mittels struktureller Analyseverfahren bewiesen werden.

4.5 Anwendungsbeispiele

Das Zusammenspiel der in Kapitel 4.4 aufgeführten Techniken und der in Kapitel 4.3 beschriebenen strukturellen Analyseverfahren wie der Berechnung von S- und T-Invarianten bei der Verifikation zeitbewerteter Systeme wird im Folgenden an Hand zweier Beispiele aus der Literatur verdeutlicht (vgl. [31],[32],[44]).

Beispiel 11
Sei das in Abbildung 4.8 dargestellte Z-Pr/T-Netz G_4 gegeben (vgl. [31],[32]). Der Modulus der Variablen i ist $n_i = 2$, so dass die Werte der Variablen i auf einem Wertebereich $[0, 1]$ abgebildet wird. Der Modulus n_j sei beliebig, aber hinreichend groß gewählt. Die Anfangsmarkierung M_0 lautet:

$$M_0 = (< 0,0,0 >, < 0,0,0 >, \emptyset, \emptyset, <>, \emptyset, <>, \emptyset, <>, \emptyset, <>, \emptyset, \emptyset, \emptyset)^T$$

Zu beweisen:
Für das Z-Pr/T-Netz G_4 gilt es zu beweisen, dass die Transition $t_{3.0}$ ausgehend von der Anfangsmarkierung M_0 nie feuern kann.

Beweis:
Unter Anwendung der in Kapitel 4.4 beschriebenen Techniken erfolgt im weiteren Verlauf eine Verifikation der zu beweisenden Aussage auf Basis einer strukturellen Analyse des zugehörigen Z-Pr/T-Netzes.

1. Simulative Inspektion (Technik I)
Zunächst wird das Z-Pr/T-Netz G_4 mittels simulativer Inspektion eingehend untersucht (siehe Kap. 4.4, Technik I). Auf Basis der simulativen Inspektion kann das Z-Pr/T-Netz G_4 stark vereinfacht und die Komplexität des zu führenden Beweises somit erheblich reduziert werden.

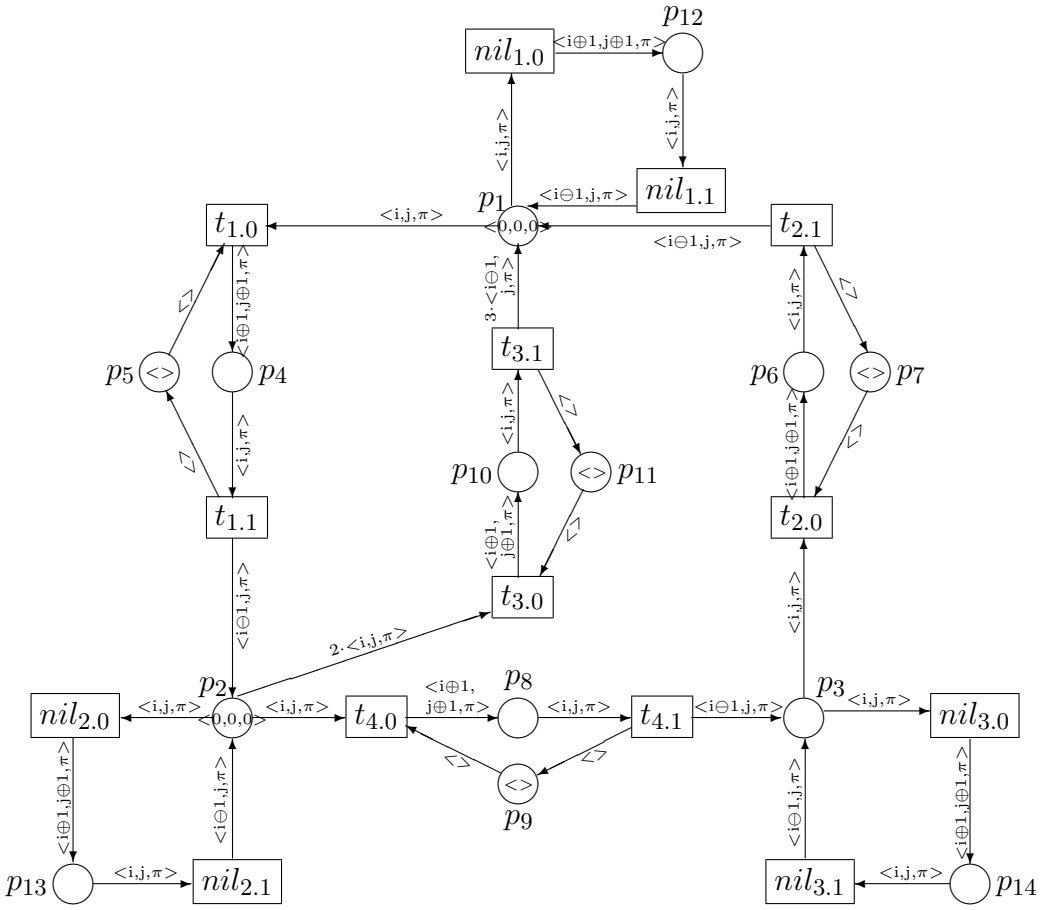


Abbildung 4.8: Z-Pr/T-Netz G_4 (vgl. [31],[32])

- *Eliminierung der NIL-Transitionen (vgl. Regel 1)*
  Das Z-Pr/T-Netz G_4 kann ohne die Transitionen $nil_{1.0}$, $nil_{2.0}$, $nil_{3.0}$ und aller von ihnen abhängigen Netzteile modelliert werden. Ist beispielsweise die Transition $nil_{1.0}$ im herkömmlichen Sinne aktiviert, dann ist entweder die Transition $t_{1.0}$ oder aber die Transition $t_{1.1}$ ebenfalls im herkömmlichen Sinne aktiviert. In beiden Fällen kann die Transition $nil_{1.0}$ nicht schalten, da diese eine Warteaktion modelliert und die Transition $t_{1.0}$ bzw. $t_{1.1}$ gemäss des vierten Prinzips der Schaltregel Vorrang hat (analog für $nil_{2.0}$ und $nil_{3.0}$). Die Transitionen $nil_{1.0}$, $nil_{2.0}$ und $nil_{3.0}$ können somit nie schalten. Das Z-Pr/T-Netz G_4 kann daher ohne diese und allen von ihnen abhängigen Netzteile modelliert werden.

- *Eliminierung der Variablen π in den Kantenanschriften (Regel 2)*
  In dem gegebenen Beispiel besitzen alle Tupel die gleiche, statische Priorität.

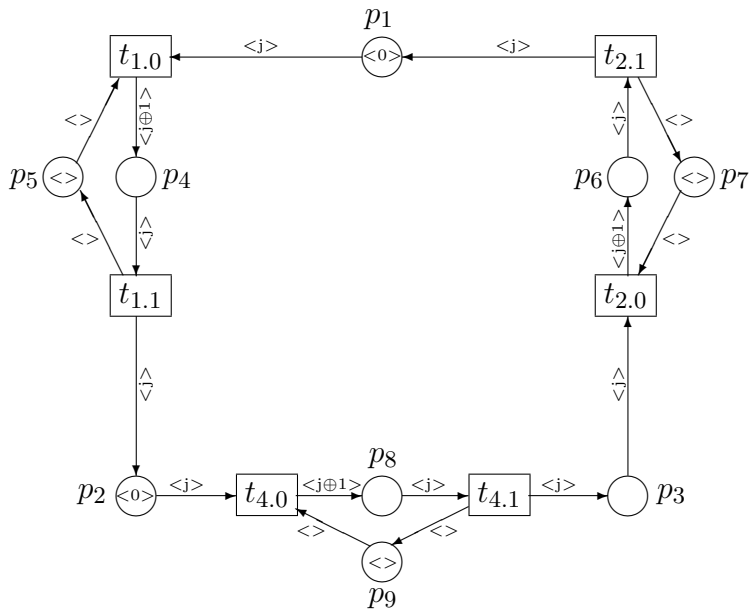


Abbildung 4.9: Z-Pr/T-Netz $G_4{}'$

Die Werte der Variablen π sind somit für alle Tupel identisch und darüber hinaus invariant. Somit findet das zweite Prinzip der Schaltregel in dem gegebenen Beispiel keine Anwendung. Das Z-Pr/T-Netz G_4 kann daher gemäss der zweiten Regel der simulativen Inspektion ohne die Variable π in den Kantenanschriften modelliert werden.

- *Eliminierung der Transition $t_{3.0}$ (Regel 3)*
  Für das Z-Pr/T-Netz G_4 gilt es zu beweisen, dass die Transition $t_{3.0}$ ausgehend von der Anfangsmarkierung M_0 nie feuern kann. Die simulative Inspektion des Z-Pr/T-Netzes G_4 zeigt, dass die Transition $t_{3.0}$ nur dann feuern kann, wenn die Stelle p_2 mit mehr als einem Tupel gleicher Variablenbelegung markiert ist. Gemäss der dritten Regel der simulativen Inspektion kann das Z-Pr/T-Netz G_4 daher ohne die Transition $t_{3.0}$ und aller von ihr abhängigen Netzteile dargestellt werden. Für das aus diesen Änderungen resultierende Z-Pr/T-Netz muss nun bewiesen werden, dass ausgehend von der Anfangsmarkierung M_0 keine Markierung erreichbar ist, so dass die Stelle p_2 mit mehr als einem Tupel gleicher Variablenbelegung markiert ist.

- *Eliminierung der Variablen i in den Kantenanschriften (Regel 2)*
  Die Werte der Variablen i können in dem Z-Pr/T-Netz, welches aus obigen Änderungen resultiert, ausschließlich auf den Stellen p_4, p_6 und p_8 Werte ungleich Null annehmen und somit die Schaltreihenfolge der Transitionen gemäss

des dritten Prinzipes der Transitionsregel beeinflussen. Eine Modellierung des Z-Pr/T-Netzes ohne die Variable i hat zur Konsequenz, dass das dynamische Verhalten des Z-Pr/T-Netzes weniger stark beschränkt wird. D.h. der Zustandsraum des daraus resultierenden Z-Pr/T-Netzes wird expandiert. Eine Modellierung des Z-Pr/T-Netzes ohne die Variable i schränkt die Gültigkeit des nachfolgenden Beweises jedoch nicht ein, da ein Zustand, welcher in dem resultierenden Z-Pr/T-Netz nicht erreichbar ist, in dem Z-Pr/T-Netz G_4 erst recht nicht erreichbar ist.

2. Strukturelle Implementierung der Synchronabstände (Technik II)
Sei das in Abbildung 4.9 dargestellte Z-Pr/T-Netz $G_4{}'$ dasjenige Z-Pr/T-Netz, welches mittels oben genannter Änderungen aus dem Z-Pr/T-Netz G_4 resultiert. Für das Z-Pr/T-Netz $G_4{}'$ ist nun zu beweisen, dass die Stelle p_2 – ausgehend von der gegebenen Anfangsmarkierung – nie mit mehr als einem Tupel gleicher Variablenbelegung markiert sein kann.
Das Z-Pr/T-Netz $G_4{}'$ ist jedoch für die gegebene Problemstellung unterbesetzt und die ermittelten S- und T-Invarianten diesbezüglich nicht aussagekräftig.
Sei zur Verdeutlichung die nachfolgende Schaltsequenz für das Z-Pr/T-Netz G_4 gegeben, welche einen möglichen Pfad in dem Erreichbarkeitsgraphen des Z-Pr/T-Netzes G_4 darstellt:

$$\begin{bmatrix} <0> \\ <0> \\ \emptyset \\ \emptyset \\ <> \\ \emptyset \\ <> \\ \emptyset \\ <> \end{bmatrix} \xrightarrow{t_{1.0}} \begin{bmatrix} \emptyset \\ <0> \\ \emptyset \\ <1> \\ \emptyset \\ \emptyset \\ <> \\ \emptyset \\ <> \end{bmatrix} \xrightarrow{t_{4.0}} \begin{bmatrix} \emptyset \\ \emptyset \\ \emptyset \\ <1> \\ \emptyset \\ \emptyset \\ <> \\ <1> \\ \emptyset \end{bmatrix} \xrightarrow{t_{1.1}} \begin{bmatrix} \emptyset \\ <1> \\ \emptyset \\ \emptyset \\ <> \\ \emptyset \\ <> \\ <1> \\ \emptyset \end{bmatrix} \xrightarrow{t_{4.1}} \begin{bmatrix} \emptyset \\ <1> \\ <1> \\ \emptyset \\ <> \\ \emptyset \\ <> \\ \emptyset \\ <> \end{bmatrix}$$

$$\xrightarrow{t_{4.0}} \begin{bmatrix} \emptyset \\ \emptyset \\ <1> \\ \emptyset \\ <> \\ \emptyset \\ <> \\ <2> \\ \emptyset \end{bmatrix} \xrightarrow{t_{2.0}} \begin{bmatrix} \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \\ <> \\ <2> \\ \emptyset \\ <2> \\ \emptyset \end{bmatrix} \xrightarrow{t_{2.1}} \begin{bmatrix} <2> \\ \emptyset \\ \emptyset \\ \emptyset \\ <> \\ \emptyset \\ <> \\ <2> \\ \emptyset \end{bmatrix} \xrightarrow{\cdots}$$

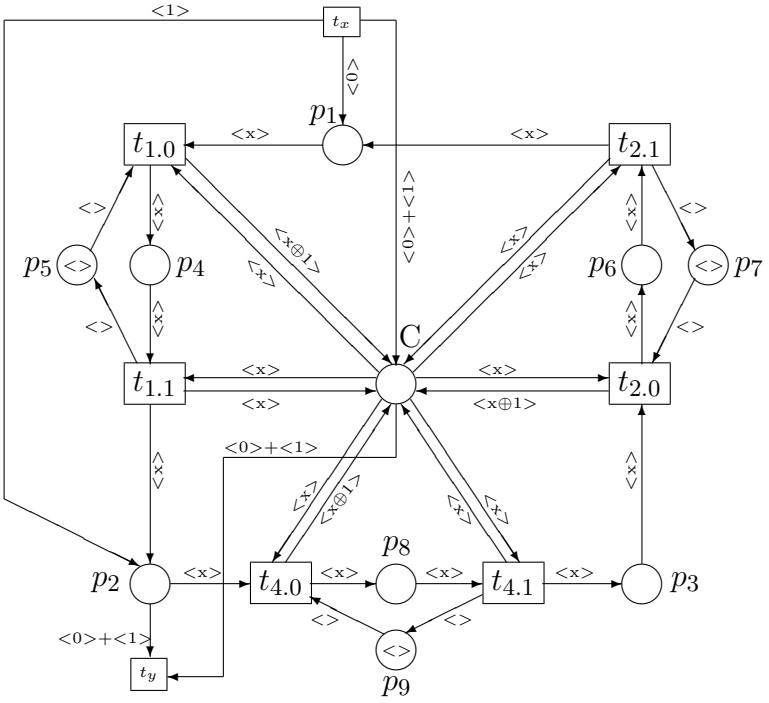


Abbildung 4.10: Z-Pr/T-Netz $G_4{''}$

An Hand der obigen Schaltsequenz wird offensichtlich, dass eine mehrfache Markierung der Stelle p_2 in dem Z-Pr/T-Netz $G_4{'}$ durch die in Z-Pr/T-Netzen geltende Schaltregel unterbunden wird. D.h. die Erreichbarkeit einer Markierung, so dass die Stelle p_2 mehrfach markiert ist, wird durch Aspekte der Dynamik verhindert. Aspekte der Dynamik können jedoch nicht strukturell analysiert werden. Es erfolgt daher eine Transformation des Z-Pr/T-Netzes $G_4{'}$ in ein Z-Pr/T-Netz $G_4{''}$, wobei in dem Z-Pr/T-Netz $G_4{''}$ das dynamische Verhalten des Z-Pr/T-Netzes $G_4{'}$ mittels Synchronabständen strukturell implementiert ist (Technik II). In dem Z-Pr/T-Netz $G_4{''}$ sind somit zusätzliche Informationen bzgl. des dynamischen Verhaltens des zu analysierenden Z-Pr/T-Netzes strukturell umgesetzt. Die strukturelle Analyse des Z-Pr/T-Netzes $G_4{''}$ inkludiert somit diese zusätzlichen Informationen und resultiert in aussagekräftigeren Ergebnissen bzgl. des zu führenden Beweises.

3. Verifikation für das Z-Pr/T-Netz $G_4{''}$ mittels T-Invarianten
Die Abbildung 4.10 zeigt das Z-Pr/T-Netz $G_4{''}$, welches durch strukturelle Implementierung der Synchronabstände aus dem Z-Pr/T-Netz $G_4{'}$ hervorgeht. Dabei werden die Synchronabstände durch die zusätzliche Stelle C realisiert. Die Schaltreihenfolge der Transitionen wird in dem Z-Pr/T-Netz $G_4{''}$ durch die Markierung

der Stelle C vorgegeben und ist somit unabhängig von den Werten der Variablen j. Eine Modellierung der Variablen j ist somit nicht notwendig. Die Kanten in dem Z-Pr/T-Netz $G_4{''}$ sind daher als einstellige Tupel über die Variable x mit Modulus $n_x = 2$ definiert. Dabei dient die Variable x lediglich der eindeutigen Identifizierung der Tupel.
Des Weiteren wurden dem Z-Pr/T-Netz $G_4{''}$ die Transitionen t_x und t_y hinzugefügt. Durch Hinzufügen der Transitionen t_x und t_y wird eine Transitionsberandung des Z-Pr/T-Netzes $G_4{''}$ erreicht, welche eine strukturelle Analyse des Z-Pr/T-Netzes mittels der Berechnung von T-Invarianten ermöglicht.

Sei die Anfangsmarkierung $M_0{''}$ des Z-Pr/T-Netzes $G_4{''}$ wie folgt gegeben:

$$M_0{''} = (\emptyset, \emptyset, \emptyset, \emptyset, <>, \emptyset, <>, \emptyset, \emptyset, \emptyset)^T$$

Durch Feuern der Transition t_x unter der Anfangsmarkierung $M_0{''}$ wird somit eine Markierung gesetzt, welche den Anfangsmarkierungen in den Z-Pr/T-Netzen G_4 und $G_4{'}$ entspricht. Demnach ist in dem Z-Pr/T-Netz $G_4{''}$ zu beweisen, dass ausgehend von der Anfangsmarkierung $M_0{''}$ eine mehrfache Markierung der Stelle p_2 nicht möglich ist. Dabei darf die Transition t_x jedoch lediglich ein Mal feuern.
Da die Transition t_y nur dann feuern kann, wenn die Stelle p_2 mit mehr als einem Tupel markiert ist, kann alternativ bewiesen werden, dass die Transition t_y in dem Z-Pr/T-Netz $G_4{''}$ nicht feuern kann. Auch hier wird ein einmaliges Feuern der Transition t_x vorausgesetzt. Der Beweis dessen erfolgt im weiteren Verlauf mittels der strukturellen Analyse des Z-Pr/T-Netzes $G_4{''}$ auf Basis der Berechnung von T-Invarianten.

Für das Z-Pr/T-Netz $G_4{''}$ existieren ausschließlich die nachfolgenden zwei T-Invarianten U_1 und U_2 (für die Berechnung der T-Invarianten siehe Anhang 6):

$$U_1 = \begin{bmatrix} 3\cdot <0> + <1> \\ 3\cdot <0> + <1> \\ <1> \\ <1> \\ <1> \\ <1> \\ 3 \cdot <> \\ 3 \cdot <> \end{bmatrix} \qquad U_2 = \begin{bmatrix} <0> + <1> \\ <0> + <1> \\ <0> + <1> \\ <0> + <1> \\ <0> + <1> \\ <0> + <1> \\ 0 \cdot <> \\ 0 \cdot <> \end{bmatrix}$$

T-Invarianten spezifizieren Schaltsequenzen zur Reproduktion einer Markierung. Sei im Folgenden vorausgesetzt, dass die Transition t_x genau ein Mal feuert. Angenommen die Transition t_y kann unter der gegebenen Voraussetzung feuern und

die Anfangsmarkierung M_0'' reproduzieren, dann muss auch eine T-Invariante U existieren, für die gilt $U(t_x) = U(t_y) = 1 \cdot <>$.
Neben den T-Invarianten U_1 und U_2 existieren jedoch keine weiteren T-Invarianten für das Z-Pr/T-Netz G_4''. Aus der Nicht-Existenz einer T-Invarianten U mit $U(t_x) = U(t_y) = 1 \cdot <>$ kann unmittelbar gefolgert werden, dass keine Schaltsequenz existiert, welche mittels jeweils einmaligem Feuern der Transitionen t_x und t_y die Anfangsmarkierung M_0'' reproduziert. D.h. ein Feuern der Transition t_y ist in dem Z-Pr/T-Netz G_5'' nicht möglich, wenn ein einmaliges Feuern der Transition t_x vorausgesetzt wird. Demzufolge kann auch die Stelle p_2 in dem Z-Pr/T-Netz G_5'' nach einmaligem Feuern der Transition t_x nicht mehrfach markiert werden. Daraus folgt wiederum unmittelbar, dass die Stelle p_2 auch in den Z-Pr/T-Netzen G_4 und G_4' nicht mehrfach markiert sein kann. Die Transition $t_{3.0}$ kann somit in dem Z-Pr/T-Netz G_4 unter der gegebenen Anfangsmarkierung nie feuern. □

An Hand des vorangegangenen Beispiels wird das Zusammenspiel der in Kapitel 4.3 vorgestellten strukturellen Analyseverfahren und der in Kapitel 4.4 beschriebenen Techniken deutlich.
Zunächst wurde das zu analysierende Z-Pr/T-Netz mittels der Techniken der simulativen Inspektion und der Implementierung von Synchronabständen transformiert. Anschließend wurde die Nicht-Erreichbarkeit einer Markierung bzw. das Nicht-Feuern einer Transition mittels struktureller Analyse des transformierten Z-Pr/T-Netzes nachgewiesen. Der Einsatz von T-Invarianten zur Verifikation obengenannter Systemeigenschaft wurde somit exemplarisch aufgezeigt.
Mittels des nachfolgenden Beispiels (vgl. [44]) wird der Zusammenhang zwischen den strukturellen Analyseverfahren und der in Kapitel 4.4 beschriebenen Techniken vertieft. Darüber hinaus wird der Einsatz von S-Invarianten zur Verifikation von Systemeigenschaften wie der Nicht-Erreichbarkeit einer Markierung aufgezeigt.

Beispiel 12
Sei das in Abbildung 4.11 dargestellte Z-Pr/T-Netz G_5 mit nachfolgender Anfangsmarkierung M_0 gegeben (vgl. [44]):

$$M_0 = (2 \cdot < 0, 0 >, \emptyset, \emptyset, <>, \emptyset, <>, \emptyset, \emptyset)^T$$

Der Modulus des Z-Pr/T-Netzes G_5 ist $n_i = 3$. D.h. die Werte der Variablen i werden auf einem Wertebereich $i \in [0, 1, 2]$ abgebildet. Der Modulus n_j sei beliebig, aber hinreichend groß gewählt.

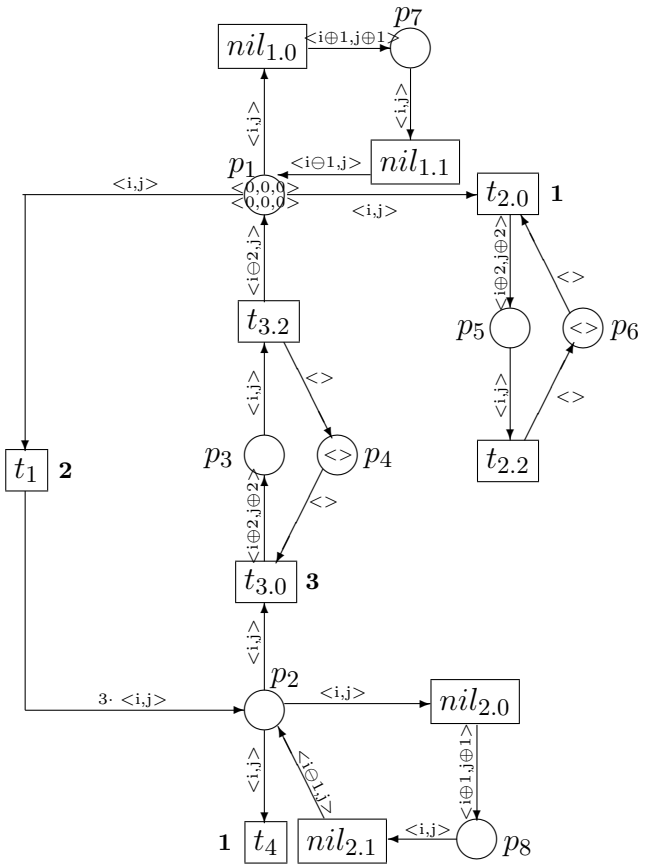


Abbildung 4.11: Z-Pr/T-Netze G_5 (vgl. [44])

Des Weiteren ist in dem Beispiel eine Form der Priorisierung gegeben, welche den Transitionen eine statische Priorität zuordnet. Ist somit eine der Transitionen in dem Z-Pr/T-Netz G_5 im herkömmlichen Sinne aktiviert, so ist sie dies immer mit der gleichen, statischen Priorität. Beispielsweise ist die Transition t_1 immer mit einer Priorität $\pi = 2$ aktiviert und die Transition $t_{2.0}$ immer mit einer Priorität $\pi = 1$. In Z-Pr/T-Netzen sind jedoch den Tupeln und nicht den Transitionen Prioritäten zugeordnet. D.h. es muss eine Abbildung der transitionsassoziierten Prioritäten, wie sie durch das Beispiel in [44] vorgegeben sind, auf die tupel-assoziierten Prioritäten gemäss Definition 23 erfolgen. Demnach muss mittels zusätzlicher Konstrukte gewährleistet werden, dass die Eingangsstellen der Transitionen immer entsprechend ihrer Prioritäten markiert sind. Von einer derartigen Modellierung wurde an dieser Stelle jedoch aufgrund der damit verbundenen zusätzlichen Komplexität abgesehen. Das Z-Pr/T-Netz G_5 wurde daher ohne die Variable π in den Kantenanschriften modelliert (vgl. Technik I, Regel 2). Die Transitionen des Z-Pr/T-Netzes G_5 sind jedoch mit den ihnen zugeordneten Prioritäten annotiert. Sind in dem Z-Pr/T-Netz G_5 mehrere Tran-

sitionen im herkömmlichen Sinne aktiviert, wobei deren Eingangsstellen mit dem gleichen Wert für die Variable j markiert sind, so erhält immer die am höchsten priorisierte Transition Vorrang.

Zu beweisen:
Für das Z-Pr/T-Netz G_5 ist zu beweisen, dass ausgehend von der Anfangsmarkierung M_0 keine Markierung erreichbar ist, so dass weder die Stelle p_1 noch die Stelle p_2 oder die Stelle p_3 markiert ist. Diese Markierung wird im weiteren Verlauf auch als leere Markierung referenziert.

Beweis:
Die Nicht-Erreichbarkeit der leeren Markierung wird im Folgenden bewiesen, indem zunächst das Z-Pr/T-Netz G_5 mittels der Technik der simulativen Inspektion transformiert und das transformierte Z-Pr/T-Netz anschließend mittels der Berechnung von S-Invarianten strukturell analysiert wird.

1. Simulative Inspektion (Technik I)
Analog zu dem vorangegangenen Beispiel 11 wird das Z-Pr/T-Netz G_5 zunächst mittels der in Kapitel 4.4 beschriebenen Technik der simulativen Inspektion vereinfacht und die Komplexität des Beweises somit erheblich reduziert.

- *Eliminierung der NIL-Transitionen (Regel 1)*
  Sowohl die Stelle p_1 als auch die Stelle p_2 haben in ihrem Nachbereich Transitionen, welche Events modellieren. Somit kann das Z-Pr/T-Netz gemäss der zweiten Regel der simulativen Inspektion ohne die Transitionen $nil_{1.0}$ und $nil_{2.0}$ und aller von ihnen abhängigen Netzteile modelliert werden. Im Falle einer Markierung der Stelle p_1 sind sowohl die Transition t_1 als auch die Transition $nil_{1.0}$ im herkömmlichen Sinne aktiviert. Auf Grund des vierten Prinzips der Transitionsregel erhält die Transition t_1 immer Vorrang vor der Transition $nil_{1.0}$. Die Transition $nil_{1.0}$ kann somit auch bei einer mehrfachen Markierung der Stelle p_1 nie schalten, so dass diese und alle von ihr abhängigen Netzteile eliminiert werden können (analog für die Transition $nil_{2.0}$).

- *Eliminierung der Transition $t_{2.0}$ und Modifikation der Anfangsmarkierung*
  In dem Z-Pr/T-Netz G_5 ist zu beweisen, dass die leere Markierung ausgehend von der Anfangsmarkierung M_0 nicht erreichbar ist. Dabei kann die initiale Tupelanzahl auf Grund der Struktur des Z-Pr/T-Netzes G_5 lediglich durch Feuern der Transitionen $t_{2.0}$ und t_4 verringert werden. Das Feuern der Transition $t_{2.0}$ wird jedoch durch die Priorisierung der Transitionen limitiert. Da die Transition t_1 höher priorisiert ist als die Transition $t_{2.0}$, kann die Transition $t_{2.0}$ genau ein Mal Feuern und die Tupelanzahl somit lediglich um eins verrin-

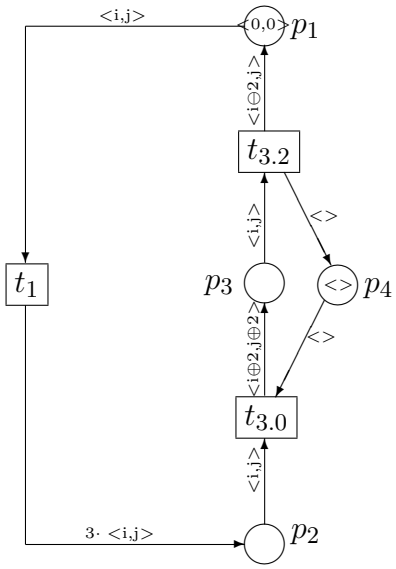


Abbildung 4.12: Z-Pr/T-Netz $G_5{}'$

gern. Es ist somit hinreichend, das Z-Pr/T-Netz G_5 ohne die Transition $t_{2.0}$ und aller von ihr abhängigen Netzteile zu modellieren und für das resultierende Z-Pr/T-Netz mit Anfangsmarkierung $M_0{}' = (< 0,0 >, \emptyset, \emptyset, <>)^T$ zu beweisen, dass die leere Markierung nicht erreichbar ist. Durch diesen Schritt werden lediglich diejenigen Pfade im Erreichbarkeitsgraphen des Z-Pr/T-Netzes G_5 eliminiert, welche ohnehin eine Erreichbarkeit der leeren Markierung ausschließen.

- *Eliminierung der Transition t_4 und Modifikation der Kantenanschrift $L(t_1, p_2)$*
  Neben der Transition $t_{2.0}$ kann des Weiteren die Transition t_4 die Tupelanzahl in dem Z-Pr/T-Netz verringern. Analog zu der Transition $t_{2.0}$ ist das Feuern der Transition t_4 in dem Z-Pr/T-Netz G_5 jedoch ebenfalls durch die gegebenen Prioritäten limitiert. Im Falle einer Markierung der Stelle p_2 feuert die Transition t_4 genau zwei Mal und konsumiert somit genau zwei der drei Tupel, mit denen die Stelle p_2 durch Feuern der Transition t_1 markiert wurde. Das Z-Pr/T-Netz G_5 kann somit ohne die Transition t_4 modelliert werden, wobei die Transition t_1 die Stelle p_2 lediglich einfach markiert. Somit werden erneut Pfade im Erreichbarkeitsgraphen des Z-Pr/T-Netzes G_5 eliminiert, welche eine Erreichbarkeit der leeren Markierung ohnehin ausschließen.

- *Eliminierung der Variablen i und j in den Kantenanschriften (Regel 2)*
  Sei das in Abbildung 4.12 dargestellte Z-Pr/T-Netz $G_5{}'$ dasjenige Z-Pr/T-Netz, welches mittels oben genannter Änderungen aus dem Z-Pr/T-Netz G_5 resultiert. Die simulative Inspektion des Z-Pr/T-Netzes $G_5{}'$ zeigt, dass weder

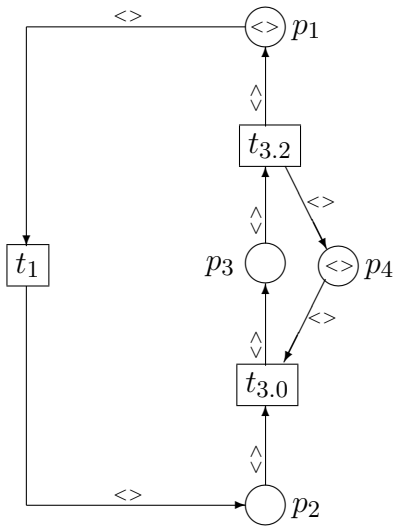


Abbildung 4.13: Z-Pr/T-Netz G_5''

das erste noch das dritte Prinzip der Transitionsregel Anwendung finden. Die Variablen i und j haben somit in dem Z-Pr/T-Netz G_5' keinen Einfluss auf die Schaltreihenfolge der Transitionen. Das Z-Pr/T-Netz G_5' kann somit ohne die Variablen i und j in den Kantenanschriften modelliert werden.

Das aus obigen Änderungen resultierende Z-Pr/T-Netz G_5'' ist in Abbildung 4.13 dargestellt. Für das Z-Pr/T-Netz G_5'' bleibt zu beweisen, dass die leere Markierung nicht erreichbar ist. Dies geschieht im Folgenden mittels S-Invarianten.

2. Verifikation für das Z-Pr/T-Netz G_5'' mittels S-Invarianten

Für das Z-Pr/T-Netz G_5'' ist zu beweisen, dass ausgehend von der Anfangsmarkierung $M_0'' = (<>, \emptyset, \emptyset, <>)^T$ die leere Markierung nicht erreichbar ist. Es muss somit immer eine der Stellen p_1, p_2 oder p_3 markiert ist. Der Nachweis dessen erfolgt im Weiteren mittels S-Invarianten.

Die Inzidenmatrix des Z-Pr/T-Netzes G_5'' lautet

$G_5''(\mathrm{p,t}) =$

	t_1	$t_{3.0}$	$t_{3.2}$
p_1	$-<>$	$\emptyset$	$<>$
p_2	$<>$	$-<>$	$\emptyset$
p_3	$\emptyset$	$<>$	$-<>$
p_4	$\emptyset$	$-<>$	$<>$

Sei der S-Vektor S wie folgt gegeben: $S = (<>, <>, <>, \emptyset)^T$. S ist eine S-Invariante für das Z-Pr/T-Netz $G_5{''}$, da gilt $S^T \bullet G_5{''}(p,t) = 0^T$.

$$\begin{aligned} S^T \bullet G_5{''}(-,t_1) &= <> \bullet(- <>) + <> \bullet <> \\ &= 0 \end{aligned}$$

$$\begin{aligned} S^T \bullet G_5{''}(-,t_{3.0}) &= <> \bullet(- <>) + <> \bullet <> \\ &= 0 \end{aligned}$$

$$\begin{aligned} S^T \bullet G_5{''}(-,t_{3.2}) &= <> \bullet <> + <> \bullet(- <>) \\ &= 0 \end{aligned}$$

Somit sind die Stellen p_1, p_2 und p_3 eine S-Invariante für das Z-Pr/T-Netz $G_5{''}$. Es gilt, dass die gewichtete Tupelanzahl auf den Stellen einer S-Invarianten konstant ist. Mit der Anfangsmarkierung $M_0{''} = (<>, \emptyset, \emptyset, <>)^T$ folgt somit, dass die gewichtete Summe der Tupel auf den Stellen der S-Invarianten S konstant eins beträgt. D.h. es ist immer eine der Stellen p_1, p_2 oder p_3 markiert. Die leere Markierung ist somit in dem Z-Pr/T-Netz $G_5{''}$ und demzufolge auch in den Z-Pr/T-Netzen $G_5{'}$ und G_5 nicht erreichbar. □

An Hand der Beispiele 11 und 12 wurde somit der Zusammenhang zwischen den strukturellen Analyseverfahren wie der Berechnung von S- und T-Invarianten und den Techniken bei der strukturellen Analyse wie der simulativen Inspektion und der Implementierung von Synchronabständen verdeutlicht. Des Weiteren belegen die Beispiele die Eignung von Z-Pr/T-Netzen zur Modellierung, Simulation und Verifikation zeitbehafteter Systeme und zeigen die Handhabbarkeit der Netzklasse bei Anwendungen geringerer Komplexität.

Z-Pr/T-Netze sind mit Kantenanschriften über die Variablen i, j und π definiert. Gerade in Hinblick auf komplexere Anwendungsbeispiele (siehe Kapitel 5) zeigt sich die Wohlfundiertheit und Notwendigkeit dieser Darstellung. Anhand der obigen Beispiele 11 und 12 konnte jedoch gezeigt werden, wie die standardisierte Darstellung von Z-Pr/T-Netzen bei Modellen von geringerer Komplexität mittels der Technik der simulativen Inspektion vereinfacht werden kann. Resultat dieser Vereinfachungen sind in hohem Maße übersichtliche und handhabbare Netze. Die Anwendbarkeit von Z-Pr/T-Netzen bei Anwendungen von größerer Komplexität wie dem EDF und dem PIP in Kapitel 5 rechtfertigt jedoch eine wie in Definition 23 vorgenommene, standardisierte Darstellung der Netzklasse mit Kantenanschriften über die Variablen i, j und π.

4.6 Fazit

In diesem Kapitel wurde die Klasse der Z-Pr/T-Netze vorgestellt. Z-Pr/T-Netze integrieren ein Konzept zur Modellierung der Systemzeit basierend auf den folgenden zwei temporären Größen – der Dauer von Aktionen und der lokalen Modellzeit.
Transitionen schalten in Z-Pr/T-Netzen zeitlos, wohingegen Zeit vergeht, während ein Tupel auf einer Stelle verweilt. Den Tupeln in Z-Pr/T-Netzen sind daher in Form der Variablen j Zeitwerte zugewiesen. Diese spezifizieren, wann ein Tupel in Bezug auf die globale Modellzeit für Transitionen wieder verfügbar ist. In Abhängigkeit der Variablen j wird für jede Stelle die lokale Modellzeit definiert. Ist eine Stelle markiert, so entspricht die lokale Modellzeit der Stelle dem Minimum aller j-Werte ihrer Markierung. Ist eine Stelle nicht markiert, so ist ihre lokale Modellzeit nicht definiert. Das Minimum aller lokalen Modellzeiten entspricht wiederum der globalen Modellzeit.
Eine Transition kann nur dann aktiviert sein, wenn die lokalen Modellzeiten aller ihrer Eingangsstellen mit der globalen Modellzeit übereinstimmen. D.h. die Tupel auf den Eingangsstellen der Transition müssen zu der aktuellen, globalen Modellzeit verfügbar sein. Ist dies für mehrere Transitionen gleichzeitig der Fall, so erhalten zunächst diejenigen Transitionen Vorrang, welche für diejenigen Tupel mit der höchsten Priorität aktiviert sind. Ist dies wiederum für mehrere Transition gleichzeitig der Fall, so erhalten diejenigen Transitionen Vorrang, welche die am weitesten fortgeschrittenen Aktionen modellieren. Warteaktionen feuern zuletzt.
Durch die Beschränkung der Wertebereiche der Variablen i und j mittels der Modulo-Operationen $\oplus$ und $\ominus$ sind Z-Pr/T-Netze endlich. Z-Pr/T-Netze sind mittels struktureller Analyseverfahren wie der Berechnung von S- und T-Invarianten sowie der Identifikation von Traps und Co-Traps strukturell analysierbar.
Bei der Verifikation mittels struktureller Analyse finden oftmals Techniken wie die simulative Inspektion, die Implementierung von Guards, die Implementierung von Synchronabständen oder die Konstruktion logischer Formeln (Syteminvarianten) als eingebettetes Teilnetz Anwendung. Zunächst wird das zu analysierende Z-Pr/T-Netz mittels der genannten Techniken wie z.B. der simulativen Inspektion oder der Implementierung von Synchronabständen transformiert und das transformierte Z-Pr/T-Netz anschließend mittels struktureller Analyseverfahren wie der Berechnung von S- und T-Invarianten strukturell analysiert.

5 Verifikation dynamischer Planungsverfahren

Die paralelle Ausführung von Prozessen in einem Betriebssystem macht eine Koordination eben dieser Prozesse sowie die Kontrolle von Zugriffen auf den Prozessor und die exklusiv nutzbaren Betriebsmittel unabdingbar. Dies ist Aufgabe sogenannter *Planungsverfahren*. Planungsverfahren übernehmen somit eine zentrale Aufgabe in einem Echtzeitsystem und stellen einen wichtigen Aspekt für die Korrektheit eines Echtzeitsystems dar. Dabei zählen das *Earliest Deadline First Protokoll (EDF)* und das *Priority Inheritance Protokoll (PIP)* zu prominenten Vertretern eben solcher Planungsverfahren.
In diesem Kapitel werden das EDF und das PIP mittels Z-Pr/T-Netzen modelliert und simuliert. Des Weiteren erfolgt eine Verifikation für das EDF und PIP basierend auf der strukturellen Analyse der zugehörigen Z-Pr/T-Netze. Die Eignung von Z-Pr/T-Netzen zur Modellierung, Simulation und Verfikation im Anwendungsbereich der Echtzeitsysteme wird anhand des EDF und PIP exemplarisch unter Beweis gestellt.

Unter einem *Prozess* versteht man eine sequenzielle Abfolge von auszuführenden, elementaren Befehlen, wobei ein Prozess durch die folgenden Parameter spezifiziert wird [5],[22]:

- **Priorität π**
  Die Priorität des Prozesses

- **Ausführungsdauer c**
  Die Prozessorzeit, die zur Beendigung des Prozesses benötigt wird

- **Startzeit b**
  Der Zeitpunkt des ersten Prozessaufrufes

- **Deadline d**
  Die Zeitspanne vom Zeitpunkt des Prozessaufrufes bis zum Zeitpunkt seiner spätesten Beendigung

- **Periode p (optional)**
  Ist ein Prozess periodisch, so gibt seine Periode p das Intervall zwischen zwei aufeinanderfolgenden Prozessaufrufen an.

Darüber hinaus kann zwischen unterbrechbaren und nicht-unterbrechbaren, sowie zwischen abhängigen und unabhängigen Prozessen unterschieden werden. Des Weiteren kann zwischen einem Single- und einem Multiprozessorbetrieb in Abhängigkeit von der Anzahl der verfügbaren Prozessoreinheiten unterschieden werden. Die nachfolgenden Betrachtungen beschränken sich auf unterbrechbare und unabhängige Prozesse, welche auf einem Singleprozessorsystem parallel ausgeführt werden.

Die Verwaltung parallel auszuführender Prozesse erfolgt in einem Echtzeitsystem mittels sogenannter *Planungsverfahren*. Zu deren Aufgaben zählt die Koordination nebenläufiger Prozesse sowie die Kontrolle von Zugriffen auf den Prozessor und die exklusiv nutzbaren Betriebsmittel. Es existiert eine Vielzahl an algorithmischen Lösungen für die Verwaltung paralleler Prozesse in einem Echtzeitsystem. Der Fokus dieser Arbeit liegt auf den *prioritätsbasierten, dynamischen Planungsverfahren*.

Den *prioritätsbasierten* Planungsverfahren liegt zu Grunde, dass jeder Prozess eine inhärente Priorität besitzt. Der Scheduler erteilt dann immer demjenigen Prozess mit der (momentan) höchsten Priorität den Zugriff auf den Prozessor. Darüber hinaus kann bei den prioritätsbasierten Planungsverfahren zwischen *statischen* und *dynamischen* Planungsverfahren unterschieden werden.

Bei einem *statischen* Planungsverfahren erfolgt die Zuteilung von Zugriffen auf den Prozessor bereits vor der Laufzeit. Da alle Prozesse eine feste, also statische Priorität besitzen, kann bereits vor Beginn der Ausführung ermittelt werden, welcher Prozess zu welchem Zeitpunkt die höchste Priorität besitzt. Im Gegensatz dazu erfolgt bei einem *dynamischen* Planungsverfahren die Zuteilung von Zugriffen auf den Prozessor sowie auf die exklusiv nutzbaren Betriebsmittel während der Laufzeit. Dies liegt darin begründet, dass die Prozesse bei einem dynamischen Planungsverfahren eine dynamische, sich über die Laufzeit verändernde Priorität besitzen. Daher fällt ein dynamischer Scheduler seine Entscheidungen während der Laufzeit auf Basis der aktuellen Prioritäten.

Im Folgenden wird anhand des EDF und des PIP die Anwendbarkeit von Z-Pr/T-Netzen zur Modellierung, Simulation und Verifikation von Anwendungen realem Komplexitätsumfangs aus dem Bereich der Echtzeitsysteme nachgewiesen. Es erfolgt daher zunächst eine Modellierung der genannten Planungsverfahren mittels Z-Pr/T-Netzen.

Dabei stützt sich die Modellierung der Anwendungsbeispiele in beiden Fällen auf die Spezifikation der Algorithmen in der *Algebra of Communicating Shared*

Resources with Value Passing(ACSR-VP) [2],[5],[20]. Bei der Algebra of Shared Resources (ACSR) handelt es sich um eine zeitbasierte Prozessalgebra, welche für die formale Spezifikation und Manipulation verteilter Echtzeitsysteme und deren Betriebsmittel entwickelt wurde [18]. Die ACSR wurde zu der ACSR-VP weiterentwickelt [2],[5], wobei letztere neben den in der ACSR bereits vorhandenen Modellierungskonzepten Mechanismen zur dynamischen Vergabe von Prioritäten sowie der Wertepropagation integriert. Auf Grund der formalen Syntax und Semantik der Modellierungssprache kann die in ACSR-VP gegebene Spezifikation des EDF und des PIP eindeutig in ein korrelierendes Z-Pr/T-Netz transformiert werden. Erläuterungen dazu sind in Anhang 6 zu finden.
Im Anschluss an die Modellierung erfolgt dann sowohl für das EDF als auch für das PIP eine Verifikation basierend auf der strukturellen Analyse der zuvor hergeleiteten Z-Pr/T-Netze.
Es wird daher zunächst in Kapitel 5.1 das EDF mittels Z-Pr/T-Netzen auf Basis der ACSR-VP Spezifikation nach Choi et al. [2],[5] modelliert. Ist eine Menge von Prozessen unter dem EDF als Planungsverfahren gegeben, so muss diese nicht zwangsläufig durchführbar sein. Eine Menge von Prozessen heißt dabei durchführbar, wenn alle Prozesse innerhalb ihrer Deadline terminieren. Da die Korrektheit in Echtzeitsystemen sowohl die Korrektheit der Ergebnisse als auch deren fristgerechte Bereitstellung umfasst, stellt die Durchführbarkeitsanalyse von Prozessen einen zentralen Aspekt bei der Verifikation von Echtzeitsystemen dar. C.L. Liu und J.W. Layland haben einen Test für die Durchführbarkeit von Prozessen unter dem EDF formuliert. Dieser Test auf Durchführbarkeit wird in Kapitel 5.1.1 vorgestellt. Jedoch ist dieser nur mit Einschränkung auf Prozesse, deren Deadline und Periode übereinstimmen, sowohl eine notwendige als auch hinreichende Bedingung für die Durchführbarkeit von Prozessen. Auf Basis der strukturellen Analyse des korrespondierenden Z-Pr/T-Netzes wird in Kapitel 5.1.2 eine Bedingung für die Durchführbarkeit von Prozessen unter dem EDF hergeleitet, welche sowohl eine notwendige als auch hinreichende Bedingung für die Durchführbarkeit von Prozessen ist.
Analog zu dem EDF wird anhand der ACSR-VP Spezifikation nach Choi et al. [2],[5] in Kapitel 5.2.2 das zu dem PIP korrespondierende Z-Pr/T-Netz abgeleitet. Die ursprüngliche Version des PIPs wurde jedoch natürlich-sprachlich und nicht in einer formal-basierten Modellierungssprache formuliert. Die fehlende formale Basis der ursprünglichen Protokollspezifikation hat in Verbindung mit der nicht zu unterschätzenden Komplexität des PIPs zu zahlreichen fehlerbehafteten Protokollspezifikationen sowie Protokollimplementierungen geführt. In Kapitel 5.2.3 und Kapitel 5.2.4 werden Fehler bzw. Fehlinterpretationen in der ursprünglichen Protokollspezifikation sowie die notwendigen Änderungen zur Fehlerbehebung erläutert. Anschließend erfolgt für die fehlerbereinigte Version des PIPs eine Ve-

rifikation der von Zöbel et al. in [49] formulierten Systeminvarianten mittels struktureller Analyse des zugehörigen Z-Pr/T-Netzes.

5.1 Das Earliest-Deadline-First-Protokoll

Das *Earliest-Deadline-First-Protokoll (EDF)* zählt zu den prioritätsbasierten, dynamischen Planungsverfahren. Die Prioritäten werden dynamisch zur Laufzeit vergeben. Es erhält dabei immer derjenige Prozess die höchste Priorität und somit Vorrang vor allen anderen Prozessen, dessen Deadline zeitlich am nächsten liegt.
Sei eine Menge von Prozessen $V = \{v_1, v_2, ..., v_n\}$ gegeben, wobei jeder Prozess v_i durch seine Ausführungsdauer c_i , Deadline d_i und Periode p_i definiert ist und bezeichne t_i die Zeitspanne, welche seit Prozessaufruf vergangen ist, mit $1 \leq i \leq n$. Des Weiteren sei d_{max} die maximale Deadline aller Prozesse mit $d_{max} = max\{d_1, d_2, ..., d_n\} + 1$. Dann berechnet sich die aktuelle Priorität eines jeden Prozesses wie folgt:

$$\pi_i \;=\; d_{max} - (d_i - t_i) \;=\; d_{max} - d_i + t_i$$

Die Differenz $d_i - t_i$ spezifiziert für jeden Prozess die verbleibende Zeit bis zum Ablaufen seiner Deadline. Da diese Differenz sich schrittweise verringert, erhöht sich die Priorität $\pi_i = d_{max} - (d_i - t_i)$ somit schrittweise. Demnach besitzt jeder Prozess v_i bei Prozessaufruf eine aktuelle Priorität $\pi_i = d_{max} - d_i$, welche auf dem Zeitintervall zwischen Prozessaufruf und Prozessbeendigung monoton steigend ist.
Basierend auf der in ACSR-VP gegebenen Spezifikation des EDFs nach Choi et al. [2],[5] wurde zunächst das korrespondierende Z-Pr/T-Netz abgeleitet. Abbildung 5.1 zeigt die ACSR-VP Spezifikation des EDFs und Abbildung 5.2 das daraus resultierende Z-Pr/T-Netz G_6. Erläuterungen zu der Herleitung von Z-Pr/T-Netzen basierend auf ACSR-VP Spezifikationen sind in Anhang 6 zu finden.
Darüber hinaus wurde dem Z-Pr/T-Netz G_6 die Stelle D hinzugefügt, deren Markierung für jeden Prozess die verbleibende Zeit bis zum Ablaufen seiner Deadline repräsentiert. Die Modellierung der Stelle D verhindert den Zugriff von Prozessen auf den Prozessor, deren Deadline bereits abgelaufen ist. Eine zusätzliche Modellierung der Stelle D war notwendig, da der Zugriff bereits verspäteter Prozesse auf den Prozessor in der ACSR-VP-Spezifikation nach Choi et al. [2],[5] nicht unterbunden wird. Dem Z-Pr/T-Netz G_6 wurde daher die Stelle D hinzugefügt, um ein derartiges Systemverhalten zu unterbinden.

$$
\begin{array}{rcl}
EDFSys & := & [\ T_1 \parallel T_2 \parallel ... \parallel T_n\ ]_{\{cpu\}} \\
T_i & := & (\ Job_i \parallel Activator_i\ ) \setminus \{start, end\} \\
\hline
Job_i & := & \emptyset : Job_i\ +\ (start?, 1)\ .\ Exec_i(0,0) \\
Exec_i(s,t) & := & (s < c_i) \longrightarrow (\{(cpu, d_{max} - d_i + t)\} : Exec_i(s+1, t+1) \\
 & & \qquad +\ \emptyset : Exec_i(s, t+1)) \\
 & & +\ (s = c_i) \longrightarrow Wait_i \\
Wait_i & := & \emptyset : Wait_i\ +\ (end?, 1)\ .\ Job_i \\
\hline
Activator_i & := & (start!, 1)\ .\ \emptyset^{d_i} : (end!, 2)\ .\ \emptyset^{p_i - d_i} : Activator_i
\end{array}
$$

Abbildung 5.1: Spezifikation des EDFs in ACSR-VP nach Choi et al. [2],[5]

Die Funktionsweise des EDFs wird im Folgenden anhand zweier Beispiele verdeutlicht. Dabei ist die in Beispiel 13 gegebene Prozessmenge durchführbar und die in Beispiel 14 gegebene Prozessmenge nicht durchführbar [2],[5].

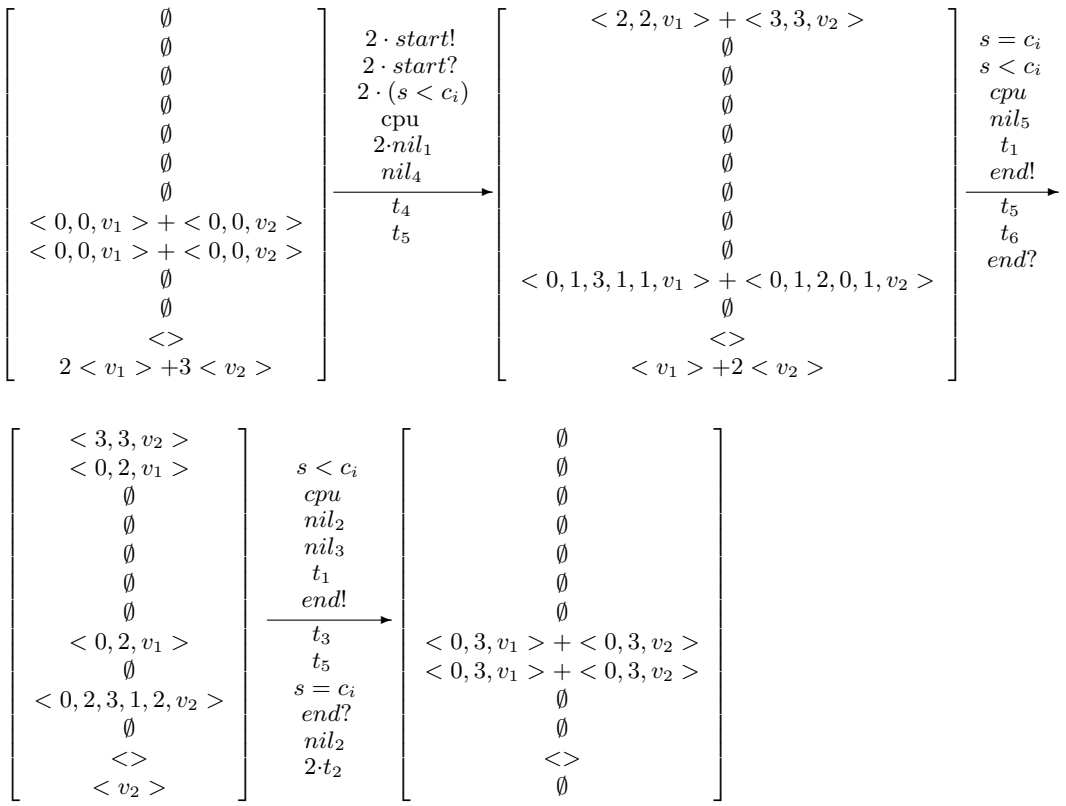
Beispiel 13
Seien zwei Prozesse v_1 und v_2 mit den nachfolgenden Parametern gegeben:

$$
\begin{array}{lll}
c_1 = 1 & d_1 = 2 & p_1 = 3 \\
c_2 = 2 & d_2 = 3 & p_2 = 3
\end{array}
$$

Die Repräsentation der Prozesse v_1 und v_2 erfolgt in dem Z-Pr/T-Netz G_6 durch eindeutig identifizierbare Tupel.

Der Modulus der Variablen i ist in dem gegebenen Beispiel $n_i = 3$, so dass die Werte der Variablen i auf einem Wertebereich $[0, 1, 2]$ abgebildet werden. Der Modulus n_j sei beliebig, aber hinreichend groß gewählt.

Die Stellen Job_i und Act_i sind somit initial jeweils mit den Tupeln $< 0, 0, v_1 > + < 0, 0, v_2 >$ markiert, die Stelle D ist initial mit den Tupeln $2 \cdot < v_1 > + 3 \cdot < v_2 >$ markiert und die Stelle CPU ist initial mit dem anonymen Tupel $<>$ markiert. Die verbleibenden Stellen sind initial nicht markiert.
Im weiteren Verlauf indizieren S-Vektoren die Stellenmenge { p_2, p_4, p_5, p_7, p_8, p_9, p_{12}, Job_i, Act_i, $Exec_i$, $Wait_i$, CPU, D }. Die verbleibenden Stellen sind aus Gründen der Übersichtlichkeit nicht mit aufgeführt. Die folgende Schaltsequenz repräsentiert dann einen möglichen Pfad im Erreichbarkeitsgraphen des Z-Pr/T-Netzes G_6:



Ausgehend von der gegebenen Anfangsmarkierung M_0 werden die Prozesse v_1 und v_2 in dem Z-Pr/T-Netz G_6 durch Feuern der Transitionen *start*! und *start*? aktiviert. Dabei berechnen sich die initialen Prioritäten für die Prozesse v_1 und v_2 mit $\pi_1 = 4 - 2 = 2$ und $\pi_2 = 4 - 3 = 1$. Nach ihrer Aktivierung konkurrieren die Prozesse v_1 und v_2 um den Zugriff auf den Prozessor, wobei Prozess v_1 auf Grund seiner höheren Priorität der Zugriff gewährt wird, während Prozess v_2 warten muss. In dem Z-Pr/T-Netz G_6 feuert somit die Transition *cpu* für das mit Prozess v_1 assoziierte Tupel $< 0, 0, 2, 0, 0, v_1 >$, wohingegen die Transition nil_4 für das mit Prozess v_2 assoziierte Tupel $< 0, 0, 1, 0, 0, v_2 >$ feuert.

Anschließend geht der Prozess v_1 in den Wartezustand über und der Prozess v_2 kann nun erstmals auf den Prozessor zugreifen kann. Die Transition *cpu* feuert somit für das mit Prozess v_2 assoziierte Tupel $< 0, 1, 2, 0, 1, v_2 >$.

Daraufhin ist die Deadline von Prozess v_1 erreicht und dieser geht in seinen Ausgangszustand über. Dementsprechend feuern die Transitionen *end*! und *end*? für das mit Prozess v_1 assoziierte Tupel $< 0, 2, v_1 >$. Der Prozess v_2 fährt hingegen mit seiner Ausführung fort und greift erneut auf den Prozessor zu. Die Transition *cpu* feuert somit erneut für das mit Prozess v_2 assoziierte Tupel $< 0, 2, 3, 1, 2, v_2 >$. Der Prozess v_2 kann seine Ausführung somit ebenfalls in-

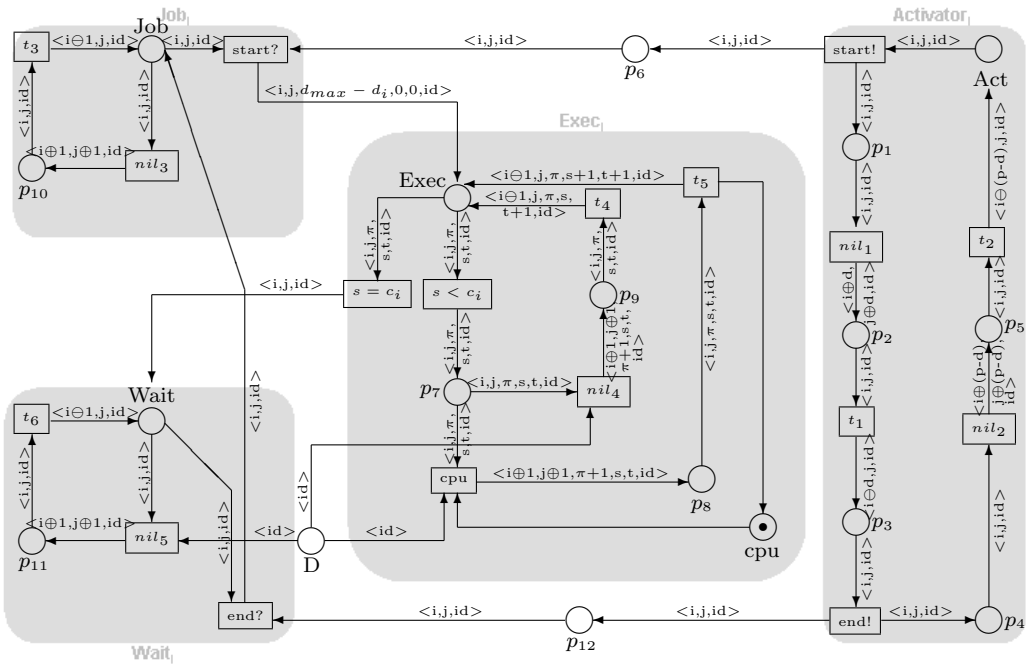


Abbildung 5.2: Z-Pr/T-Netz G_6

nerhalb seiner Deadline beenden und durch Feuern der Transitionen *end*! und *end*? in seinen Ausgangszustand übergehen. □

Die in Beispiel 13 aufgeführten Prozesse sind somit *durchführbar* (engl. *feasible*). D.h. alle Prozesse terminieren innerhalb ihrer Deadline. Es zeigt sich jedoch, dass eine derartige Lösung nicht zwangsläufig existieren muss. Zur Verdeutlichung sei das nachfolgende Beispiel gegeben [2],[5].

Beispiel 14
Seien erneut zwei Prozesse v_1 und v_2, jedoch mit den nachfolgenden Parametern gegeben:

$$\begin{array}{lll} c_1 = 2 & d_1 = 2 & p_1 = 3 \\ c_2 = 2 & d_2 = 3 & p_2 = 3 \end{array}$$

Der Modulus des Z-Pr/T-Netzes ist erneut $n_i = 3$, so dass die Werte der Variablen i auf einem Wertebereich $[0, 1, 2]$ abgebildet werden. Der Modulus n_j sei beliebig, aber hinreichend groß gewählt.

Initial sind somit die Stellen Job_i und Act_i jeweils mit den Tupeln $< 0, 0, v_1 > + < 0, 0, v_2 >$ markiert, die Stelle D mit den Tupeln $2 < v_1 > +3 < v_2 >$ und die Stelle CPU mit dem anonymen Tupel $<>$, wohingegen die verbleibenden Stellen initial nicht markiert sind.

Sei mit S-Vektoren erneut die Stellenmenge $\{ p_2, p_4, p_5, p_7, p_8, p_9, p_{12}, Job_i, Act_i, Exec_i, Wait_i, CPU, D \}$ indiziert, wohingegen die verbleibenden Stellen aus Gründen der Übersichtlichkeit nicht aufgeführt sind. Dann repräsentiert die folgende Schaltsequenz für das gegebene Beispiel einen möglichen Pfad im Erreichbarkeitsgraphen des Z-Pr/T-Netzes G_6:

$$\begin{bmatrix} \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \\ < 0, 0, v_1 > + < 0, 0, v_2 > \\ < 0, 0, v_1 > + < 0, 0, v_2 > \\ \emptyset \\ \emptyset \\ <> \\ 2 < v_1 > +3 < v_2 > \end{bmatrix} \xrightarrow[t_4 \; t_5]{2 \cdot \text{start}! \; 2 \cdot \text{start}? \; 2 \cdot (s < c_i) \; cpu \; nil_4 \; 2 \cdot nil_1} \begin{bmatrix} < 2, 2, v_1 > + < 3, 3, v_2 > \\ \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \\ < 0, 1, 3, 1, 1, v_1 > + < 0, 1, 2, 0, 1, v_2 > \\ \emptyset \\ <> \\ < v_1 > +2 < v_2 > \end{bmatrix} \xrightarrow[t_5 \; s = c_i \; end?]{2 \cdot (s < c_i) \; cpu \; nil_4 \; t_1 \; end! \; t_4}$$

$$\begin{bmatrix} < 3, 3, v_2 > \\ < 0, 2, v_1 > \\ \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \\ < 0, 2, v_1 > \\ \emptyset \\ < 0, 2, 3, 0, 2, v_2 > \\ \emptyset \\ <> \\ < v_2 > \end{bmatrix} \xrightarrow[t_3 \; t_5 \; s < c_i \; nil_2 \; 2{\cdot}t_2]{s < c_i \; cpu \; nil_2 \; nil_3 \; t_1 \; end!} \begin{bmatrix} \emptyset \\ \emptyset \\ \emptyset \\ < 0, 3, 4, 1, 3, v_2 > \\ \emptyset \\ \emptyset \\ < 0, 3, v_2 > \\ < 0, 3, v_1 > \\ < 0, 3, v_1 > + < 0, 3, v_2 > \\ \emptyset \\ \emptyset \\ <> \\ \emptyset \end{bmatrix}$$

Wie in Beispiel 13 werden die Prozesse v_1 und v_2 durch Feuern der Transitionen *start*! und *start*? mit Priorität $\pi_1 = 2$ und $\pi_2 = 1$ aktiviert. Erneut konkurrieren die Prozesse v_1 und v_2 um den Zugriff auf den Prozessor. Dieser wird auf Grund der höheren Priorität dem Prozess v_1 gewährt, während der Prozess v_2 warten muss. In dem Z-Pr/T-Netz G_6 feuert somit die Transition *cpu* für das mit Prozess v_1 assoziierte Tupel $< 0, 0, 2, 0, 0, v_1 >$, wohingegen die Transition nil_4 für das mit Prozess v_2 assoziierte Tupel $< 0, 0, 1, 0, 0, v_2 >$ feuert.

Im Gegensatz zu dem vorherigen Beispiel 13 ist die Ausführung von Prozess v_1 jedoch damit nicht abgeschlossen, so dass die Prozesse v_1 und v_2 erneut um den

Zugriff auf den Prozessor konkurrieren. Dieser wird wiederum Prozess v_1 gewährt, während Prozess v_2 erneut warten muss. Die Transition *cpu* feuert somit erneut für das mit Prozess v_1 assoziierte Tupel $< 0, 1, 3, 1, 1, v_1 >$ und die Transition nil_4 erneut für das mit Prozess v_2 assoziierte Tupel $< 0, 1, 2, 0, 1, v_2 >$.
Im Anschluss daran ist die Deadline von Prozess v_1 erreicht. Dieser konnte seine Ausführung nach zweimaligem Zugriff auf den Prozessor fristgerecht beenden und geht in seinen Anfangszustand über. In dem Z-Pr/T-Netz G_6 feuern somit die Transitionen *end*! und *end*? für das mit Prozess v_1 assoziierte Tupel $< 0, 2, v_1 >$. Der Prozess v_2 hingegen erhält nun erstmalig Zugriff auf den Prozessor. Dementsprechend feuert die Transition *cpu* für das mit Prozess v_2 assoziierte Tupel $< 0, 2, 3, 0, 2, v_2 >$. Die Deadline von Prozess v_2 ist jedoch erreicht, so dass in dem Z-Pr/T-Netz G_6 die Transition *end*! feuert. Der Prozess v_2 konnte jedoch seine Ausführung nach einmaligem Zugriff auf den Prozessor nicht erfolgreich abschließen. Der Prozess v_2 verpasst somit seine Deadline und die Transition *end*? kann nicht feuern. Demnach sind die Prozesse v_1 und v_2 nicht durchführbar. □

Die Beispiele 13 und 14 zeigen, dass Z-Pr/T-Netze einen adäquaten Formalismus zur Modellierung und Simulation komplexer Systeme aus dem Anwendungsbereich der Echtzeitsysteme darstellen.
In der Praxis ist allerdings – neben der Möglichkeit der Modellierung und Simulation sicherheitskritischer Echtzeitsysteme – vor allem die Verifikation von Systemeigenschaften mittels struktureller Analysemethoden von Interesse. D.h. es sollen Aussagen über die Korrektheit des Systems an Hand der Struktur des zu analysierenden Z-Pr/T-Netzes abgeleitet werden. Dabei beinhaltet Korrektheit in Echtzeitsystemen neben der Korrektheit der Ergebnisse auch einen temporären Aspekt. Ein Echtzeitsystem muss nicht nur die korrekten Ergebnisse liefern, es muss diese auch innerhalb gegebener Fristen liefern. Einen wesentlichen Bestandteil der Verifikation im Anwendungsbereich der Echtzeitsysteme stellt somit die *Durchführbarkeitsanalyse* (engl. *Feasibility Analysis*) für Prozesse unter einem Planungsverfahren dar. D.h. es muss für eine Prozessmenge nachgewiesen werden, dass alle Prozesse innerhalb ihrer gegebenen Fristen terminieren. Der Fokus der nachfolgenden Betrachtungen liegt somit auf dem zeitlichen Aspekt der Protokollverifikation, der sogenannten Durchführbarkeitsanalyse.

5.1.1 Durchführbarkeitsanalyse nach Liu und Layland

C.L. Liu und J.W. Layland haben in [22] eine Bedingung für die Durchführbarkeit von Prozessen formuliert.
Sei eine Menge von Prozessen $V = \{v_1, v_2, ..., v_n\}$ gegeben, wobei c_i die Ausführungsdauer, d_i die Deadline und p_i die Periode von Prozess v_i bezeich-

net, mit $1 \leq i \leq n$. Unter der Prämisse, dass für jeden Prozess $v_i \in V$ Deadline und Periode übereinstimmen, also $d_i = p_i$ gilt, stellt die folgende Ungleichung sowohl eine notwendige als auch hinreichende Bedingung für die Durchführbarkeit von Prozessen dar. Eine Menge von Prozessen ist genau dann *durchführbar*, wenn gilt:

$$\frac{c_1}{d_1} + \frac{c_2}{d_2} + ... + \frac{c_n}{d_n} \leq 1 \tag{5.1}$$

Ist die obige Ungleichung für eine Prozessmenge V erfüllt, so sind die Prozesse in V durchführbar. Ebenso gilt in der Umkehrung, dass die obige Ungleichung erfüllt ist, wenn eine Prozessmenge V durchführbar ist.

Beispiel 15
Seien zwei Prozesse v_1 und v_2 mit den folgenden Parametern gegeben:

$$\begin{array}{lll} c_1 = 1 & d_1 = 3 & p_1 = 3 \\ c_2 = 2 & d_2 = 3 & p_2 = 3 \end{array}$$

Es gilt $d_1 = p_1 = 3$ und $d_2 = p_2 = 3$. Somit stellt die Ungleichung 5.1 nach C.L. Liu und J.W. Layland sowohl eine notwendige als auch hinreichende Bedingung für die Durchführbarkeit der Prozesse v_1 und v_2 dar.

$$\frac{c_1}{d_1} + \frac{c_2}{d_2} = \frac{1}{3} + \frac{2}{3} = 1 \leq 1$$

Die Ungleichung 5.1 ist somit erfüllt. Daraus folgt unmittelbar, dass die Prozesse v_1 und v_2 durchführbar sind. □

Beispiel 16
Seien erneut zwei Prozesse v_1 und v_2, jedoch mit den nachfolgenden Parametern gegeben:

$$\begin{array}{lll} c_1 = 2 & d_1 = 3 & p_1 = 3 \\ c_2 = 2 & d_2 = 3 & p_2 = 3 \end{array}$$

Es gilt $d_1 = p_1 = 3$ und $d_2 = p_2 = 3$. Somit stellt die Ungleichung 5.1 nach C.L. Liu und J.W. Layland auch hier sowohl eine notwendige als auch hinreichende Bedingung für die Durchführbarkeit der Prozesse v_1 und v_2 dar.

$$\frac{c_1}{d_1} + \frac{c_2}{d_2} = \frac{2}{3} + \frac{2}{3} = \frac{4}{3} \not\leq 1$$

Die Ungleichung 5.1 ist nicht erfüllt. Somit sind die Prozesse v_1 und v_2 auch nicht durchführbar. □

Ist eine Menge von Prozessen $V = \{v_1, v_2, ..., v_n\}$ gegeben und ist die Prämisse $d_i = p_i$ für (mindestens) einen der Prozesse $v_i \in V$ nicht erfüllt, so stellt die von C.L. Liu und J.W. Layland formulierte Bedingung (siehe Ungleichung 5.1) ausschließlich eine hinreichende, aber keine notwendige Bedingung für die Durchführbarkeit der Prozesse in V dar [35],[39].
Erfüllt in diesem Fall eine Prozessmenge V die Ungleichung 5.1, so sind die Prozesse in V auch durchführbar. Die Umkehrung gilt jedoch nicht. D.h. es kann durchaus eine Prozessmenge V existieren, welche durchführbar ist, die die Ungleichung 5.1 jedoch nicht erfüllt. Ist die Ungleichung 5.1 nicht erfüllt, kann nicht daraus geschlossen werden, dass die Prozessmenge V auch tatsächlich nicht durchführbar ist.

Beispiel 17
Seien erneut zwei Prozesse v_1 und v_2 mit den folgenden Parametern wie in Beispiel 13 gegeben:

$$\begin{array}{lll} c_1 = 1 & d_1 = 2 & p_1 = 3 \\ c_2 = 2 & d_2 = 3 & p_2 = 3 \end{array}$$

Es gilt $d_1 \neq p_1$. Die Ungleichung 5.1 nach C.L. Liu und J.W. Layland stellt somit lediglich eine hinreichende, aber keine notwendige Bedingung für die Durchführbarkeit der Prozesse v_1 und v_2 dar.

$$\frac{c_1}{d_1} + \frac{c_2}{d_2} = \frac{1}{2} + \frac{2}{3} = \frac{7}{6} \not\leq 1$$

Die Ungleichung 5.1 ist nicht erfüllt, dennoch sind die Prozesse v_1 und v_2 durchführbar (siehe Bsp. 13). □

Die von C.L. Liu und J.W. Layland in [22] formulierte Bedingung (siehe Ungleichung 5.1) stellt somit nur dann eine sowohl notwendige als auch hinreichende Bedingung für die Durchführbarkeit von Prozessen dar, wenn für alle Prozesse $v_i \in \{v_1, v_2, ..., v_n\}$ gilt, dass $d_i = p_i$. Anderenfalls handelt es sich lediglich um eine hinreichende, aber keine notwendige Bedingung für die Durchführbarkeit von Prozessen.
Im Rahmen dieser Arbeit wurde auf Basis der strukturellen Analyse des korrespondierenden Z-Pr/T-Netzes eine allgemeingültige Bedingung für die Durchführbarkeit von Prozessen hergeleitet, welche unabhängig von der Prämisse $d_i = p_i$ für alle Prozesse $v_i \in \{v_1, v_2, ..., v_n\}$ eine sowohl notwendige als auch hinreichende Bedingung für die Durchführbarkeit von Prozessen darstellt.

5.1.2 Durchführbarkeitsanalyse mittels Z-Pr/T-Netzen

Im Folgenden wird eine allgemeingültige Bedingung für die Durchführbarkeit von Prozessen unter dem EDF als Planungsverfahren aufgezeigt, welche mittels struktureller Analyse des zugehörigen Z-Pr/T-Netzes hergeleitet wurde. Dabei ist eine Menge von Prozessen durchführbar, wenn jeder Prozess innerhalb seiner Deadline terminiert.

Die nachfolgenden Betrachtungen beziehen sich auf periodische Prozesse. Es muss daher für jeden Prozess nachgewiesen werden, dass dieser nicht nur bei einmaliger Prozessausführung, sondern auch bei jeder wiederholten Prozessausführung fristgerecht beendet wird. Es muss demnach immer ein vollständiger Ausführungszyklus betrachtet werden. Ist eine Menge von Prozessen $V = \{v_1, v_2, ..., v_n\}$ gegeben, dann wird ein Ausführungszyklus durch das kleinste gemeinsame Vielfache (kgV) der Perioden aller Prozesse aus V bestimmt. Im Rahmen einer Durchführbarkeitsanalyse muss demnach das Zeitintervall $[0, kgV]$ betrachtet werden. Innerhalb dieser Zeitspanne kommt jeder Prozess $v_i \in V$ genau (kgV/p_i)-Mal zur Ausführung. Sei $0 \leq m < (kgV/p_i)$, dann berechnen sich die absoluten Deadlines der Prozessausführungen von Prozess $v_i \in V$ wie folgt:

$$d_i{}^m = m \cdot p_i + d_i$$

Sind die in V gegebenen Prozesse durchführbar, so muss jeder der in V enthaltenen Prozesse v_i in dem Zeitintervall $[0, kgV]$ genau (kgV/p_i)-Mal zur Ausführung kommen. Dabei muss jeder Prozess v_i bei seiner $(m+1)$-ten Prozessausführung innerhalb seiner Deadline $d_i{}^m$ beendet werden, mit $0 \leq m < (kgV/p_i)$. Anderenfalls sind die Prozesse in V nicht durchführbar.

Beispiel 18
Seien zwei Prozesse v_1 und v_2 mit den folgenden Parametern gegeben:

$$\begin{array}{lll} c_1 = 2 & d_1 = 3 & p_1 = 3 \\ c_2 = 2 & d_2 = 4 & p_2 = 4 \end{array}$$

Demnach muss im Rahmen einer Durchführbarkeitsanalyse das Zeitintervall $[0, 12]$ betrachtet werden. Innerhalb dieser Zeitspanne muss der Prozess v_1 insgesamt vier Mal ausgeführt werden, der Prozess v_2 hingegen drei Mal. Die absoluten Deadlines der Prozesse v_1 und v_2 berechnen sich demnach wie folgt:

$$\begin{array}{lcll} d_1{}^0 & = & 0 \cdot 3 + 3 & = 3 \\ d_1{}^1 & = & 1 \cdot 3 + 3 & = 6 \\ d_1{}^2 & = & 2 \cdot 3 + 3 & = 9 \\ d_1{}^3 & = & 3 \cdot 3 + 3 & = 12 \end{array} \qquad \begin{array}{lcll} d_2{}^0 & = & 0 \cdot 4 + 4 & = 4 \\ d_2{}^1 & = & 1 \cdot 4 + 4 & = 8 \\ d_2{}^2 & = & 2 \cdot 4 + 4 & = 12 \end{array}$$

Sind die Prozesse v_1 und v_2 durchführbar, so muss der Prozess v_1 bei seiner ersten Ausführung innerhalb der Deadline ${d_1}^0 = 3$ terminieren, bei seiner zweiten Ausführung innerhalb der Deadline ${d_1}^1 = 6$, bei seiner dritten Ausführung innerhalb der Deadline ${d_1}^2 = 9$ und bei seiner vierten und letzten Ausführung innerhalb der Deadline ${d_1}^3 = 12$. Analog für Prozess v_2. □

Durchführbarkeitsanalyse

Es muss demnach für eine Menge von Prozessen V bewiesen werden, dass jeder Prozess v_i in dem Zeitintervall $[0, kgV]$ insgesamt (kgV/p_i)-Mal ausgeführt wird und dabei jeweils bei seiner $(m+1)$-Ausführung innerhalb der Deadline ${d_i}^m$ terminiert.

In dem Z-Pr/T-Netz G_6 entspricht das Erreichen der Deadline eines Prozesses dem Schalten der Transition *end*!. Ist die Deadline ${d_i}^m$ eines Prozesses v_i erreicht, so feuert die Transition *end*! für ein Tupel $< 0, {d_i}^m, v_i >$. Des Weiteren ist in dem Z-Pr/T-Netz G_6 die fristgerechte Beendigung eines Prozesses mit dem Feuern der Transition *end*? assoziiert. Kann ein Prozess v_i seine Ausführung innerhalb seiner Deadline ${d_i}^m$ beenden, so feuert die Transition *end*? für ein Tupel $< 0, {d_i}^m, v_i >$. Kann ein Prozess v_i seine Ausführung jedoch nicht fristgerecht beenden, so kann die Transition *end*? nicht feuern und die Stelle p_{12} bleibt weiterhin mit einem Tupel $< 0, {d_i}^m, v_i >$ markiert.

Im Rahmen einer Durchführbarkeitsanalyse muss demzufolge bewiesen werden, dass die Transition *end*? in dem Z-Pr/T-Netz G_6 für jeden Prozess $v_i \in V$ genau (kgV/p_i)-Mal feuert, wobei sie dies beim dem $(m+1)$-Mal für ein Tupel $< 0, {d_i}^m, v_i >$ tut. Der Nachweis dessen erfolgt im Weiteren mittels struktureller Analyse des Z-Pr/T-Netzes G_6 auf Basis der Identifikation nicht-markierter Co-Traps.

Strukturelle Implementierung von Guards

Im weiteren Verlauf erfolgt eine Durchführbarkeitsanalyse für Prozesse unter dem EDF auf Basis einer strukturellen Analyse der zugehörigen Z-Pr/T-Netzes G_6. Um eine derartige strukturelle Analyse des Z-Pr/T-Netzes G_6 mittels der Identifikation von nicht-markierten Co-Traps zu erleichtern, erfolgte eine strukturelle Implementierung der Guards (siehe Kapitel 4.4, Technik III). Dem Z-Pr/T-Netz G_6 wurden daher die Stellen C und $\neg C$ hinzugefügt.

Die Markierung der Stelle C repräsentiert für jeden Prozess die Anzahl der noch notwendigen Prozessorzugriffe und dient der strukturellen Modellierung der Schaltbedingung $(s < c_i)$. Die Markierung der Stelle $\neg C$ repräsentiert hingegen für jeden Prozess die Anzahl der bereits gewährten Prozessorzugriffe und

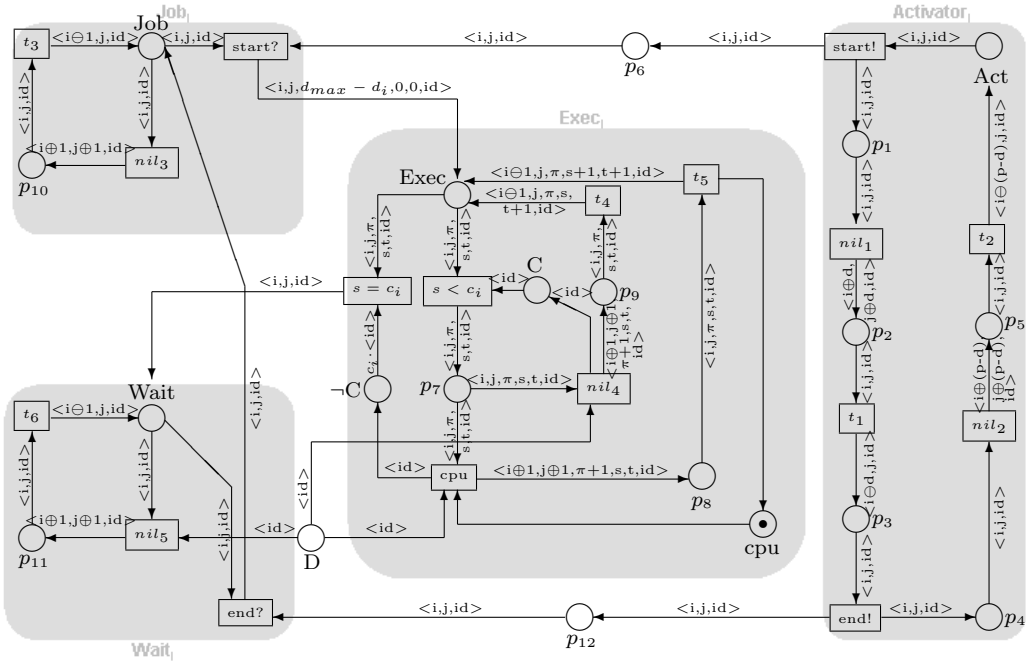


Abbildung 5.3: Z-Pr/T-Netz $G_6{}'$

modelliert somit strukturell die Schaltbedingung $(s = c_i)$.

Eine strukturelle Implementierung der Guards hat sich hinsichtlich der nachfolgenden Durchführbarkeitsanalyse als sachdienlich erwiesen. Das aus diesen Änderungen resultierende Z-Pr/T-Netz $G_6{}'$ ist in Abbildung 5.3 dargestellt.

Für das Z-Pr/T-Netz $G_6{}'$ ist zu beweisen, dass die Transition *end*? für jeden Prozess $v_i \in V$ genau (kgV/p_i)-Mal feuert, wobei sie dies bei dem $(m+1)$-Mal für ein Tupel $< 0, {d_i}^m, v_i >$ tut, mit $0 \leq m < (kgV/p_i)$.

Der Nachweis dessen beruht auf der Identifikation nicht-markierter Co-Traps. Jede Transition t, welche ein Tupel gemäss der Kantenanschrift $L(t,p)$ auf einer Stelle p des Co-Traps produziert, konsumiert auch wieder ein Tupel gemäss der Kantenanschrift $L(p',t)$ von einer Stelle p' des Co-Traps. Somit kann ein nicht-markierter Co-Trap auch nicht wieder markiert werden.

Im Rahmen des nachfolgenden Beweises kann mittels struktureller Analyse für das Z-Pr/T-Netz $G_6{}'$ der Co-Trap C identifiziert werden. Dabei ist die Stelle $Wait_i$ Bestandteil des Co-Traps C. Die Stelle $Wait_i$ ist ihrerseits wiederum eine Eingangsstelle der Transition *end*?. Ist der Co-Trap C nicht markiert, so kann er auf Grund der Eigenschaften von Co-Traps auch nicht wieder markiert werden. Dementsprechend kann die Stelle $Wait_i$ als Bestandteil des Co-Traps auch nicht

wieder markiert werden und die Transition *end*? demzufolge auch nicht schalten. Die Durchführbarkeit von Prozessen unter dem EDF kann somit auf die Existenz nicht-markierter Co-Traps zurückgeführt werden.
Zur Identifikation von Co-Traps in dem Z-Pr/T-Netz $G_6{}'$ wird ein sogenannter *Backtrace* erstellt. Mittels Backtrace können auf Basis der strukturellen Analyse des Z-Pr/T-Netzes $G_6{}'$ Zusammenhänge extrahiert werden, welche ein Nicht-Feuern der Transition *end*? bedingen. Die Methode des Backtracens beruht auf dem rückwärtsgerichteten Zurückverfolgen von Pfaden in dem Z-Pr/T-Netz. Können dabei alle Pfade des Backtrace auf Zyklen zurückgeführt werden, so bilden die beim Backtrace durchlaufenen Stellen für die mit ihnen assoziierten Tupel einen Co-Trap.

Vorgehensweise bei der Erstellung eines Backtrace
Ausgehend von einer Transition wird ein Backtrace nach folgendem Schema erstellt:

- Der Backtrace wird über *eine der Eingangsstellen* fortgesetzt. Dies begründet sich darin, dass eine Transition in einem Z-Pr/T-Netz für ein Tupel nicht feuern kann, wenn eine ihrer Eingangsstellen nicht, nicht ausreichend oder mit nicht korrelierenden Variablenwerten markiert ist. Der Backtrace wird daher über eine der nicht bzw. nicht ausreichend markierten Eingangsstellen fortgesetzt. Sind alle Eingangsstellen markiert, jedoch mit nicht-korrelierenden Tupeln, so kann der Backtrace über eine beliebige dieser Eingangsstellen fortgesetzt werden.
  - Es gilt zu beachten, dass der Backtrace mit einem Tupel fortgesetzt werden muss, dessen Werte gemäss der Kantenanschriften adaptiert sind. Wird eine Transition t mit einem Tupel $< i, j, \pi >$ bei einem Backtrace durchlaufen und sei mit der Stelle p diejenige Stelle gegeben, über die der Backtrace fortgesetzt wird, wobei $L(p,t) =< i \oplus \delta_i, j \oplus \delta_j, \pi + \delta_\pi >$ gilt. Dann muss der Backtrace mit dem Tupel $< i \ominus \delta_i, j \ominus \delta_j, \pi - \delta_\pi >$ über die Stelle p fortgesetzt werden (für $L(p,t) =< i \ominus \delta_i, j \ominus \delta_j, \pi - \delta_\pi >$ analog mit $< i \oplus \delta_i, j \oplus \delta_j, \pi + \delta_\pi >$).
  - Ist eine der Eingangsstellen der Transition t bereits mit exakt diesem Tupel durchlaufen worden, so konnte ein Zyklus identifiziert werden. Dieser Pfad des Backtrace kann somit geschlossen werden.
  - Sind alle Eingangsstellen einer Transition ausreichend und mit korrelierenden Variablenwerten markiert, so muss dieser Pfad des Backtrace abgebrochen werden. Es muss (wenn möglich) zu einer Verzweigung im Backtrace zurückgesprungen werden, wo der Backtrace über einen alter-

nativen Pfad fortgesetzt werden kann. D.h. es muss eine Transition gefunden werden, welche mehr als eine nicht bzw. nicht adäquat markierte Eingangsstelle besitzt. Anschließend muss der Backtrace über eine der alternativen Eingangsstellen fortgesetzt werden. Ist dies nicht möglich, so muss der Backtrace ohne Ergebnis abgebrochen werden.

- Bei Stellen wird der Backtrace über *alle Eingangstransitionen* fortgesetzt. Grund dafür ist, dass eine Stelle in einem Z-Pr/T-Netz nicht bzw. nicht adäquat markiert werden kann, wenn keine ihrer Eingangstransitionen feuert, so dass diese Stelle durch ihr Feuern markiert bzw. adäquat markiert ist. Daher wird der Backtrace über alle Eingangstransitionen fortgesetzt.
  - Auch hier muss der Backtrace mit einem Tupel fortgesetzt werden, dessen Werte gemäss der Kantenanschriften angepasst sind. Wird eine Stelle p mit einem Tupel $< i, j, \pi >$ bei einem Backtrace durchlaufen und sei mit t eine der Eingangstransitionen der Stelle p gegeben, wobei $L(t,p) =< i \oplus \delta_i, j \oplus \delta_j, \pi + \delta_\pi >$ gilt. Dann muss der Backtrace mit dem Tupel $< i \ominus \delta_i, j \ominus \delta_j, \pi - \delta_\pi >$ über die Transition t fortgesetzt werden (für $L(t,p) =< i \ominus \delta_i, j \ominus \delta_j, \pi - \delta_\pi >$ analog mit $< i \oplus \delta_i, j \oplus \delta_j, \pi + \delta_\pi >$).

Nach diesem Schema wird vorgegangen bis alle Pfade geschlossen sind. Konnten dabei alle Pfade auf Zyklen zurückgeführt werden, so bilden die beim Backtrace durchlaufenen Stellen einen Co-Trap. Mit jeder Stelle des Co-Traps sind diejenigen Tupel assoziiert, mit denen die jeweilige Stelle bei Erstellen des Backtrace durchlaufen wurde. Dieser Co-Trap gilt dann als nicht-markiert, wenn keine der Stellen des Co-Traps derart markiert ist, dass die Markierung einem der ihr zugeordneten Tupel entspricht. □

Backtrace für das Z-Pr/T-Netz $G_6{}'$

Nach obigem Schema wurde für das Z-Pr/T-Netz $G_6{}'$ ausgehend von der Transition *end*? und einem Tupel $< 0, {d_i}^m, v_i >$ ein Backtrace erstellt. Dieser ist in Abbildung 5.2.5 dargestellt. Dabei wurden lediglich die Parameter i, j und id mitgeführt, da die Parameter π, s und t auf die nachfolgende Durchführbarkeitsanalyse keinen Einfluss haben.

Der für das Z-Pr/T-Netz $G_6{}'$ vorgenommene Backtrace liefert einen durch die Variablen ${d_i}^m$ und v_i parametrisierten Co-Trap C.

Indiziere ein Stellenvektor die Stellenmenge $\{ p_1, p_2, ..., p_{12}, Job_i, Act_i, Exec_i, Wait_i, CPU, C, D, \neg C \}$. Dann spezifiziert der nachfolgende Stellenvektor C den mittels Backtrace identifizierten, parametrisierten Co-Trap für das Z-Pr/T-Netz $G_6{}'$:

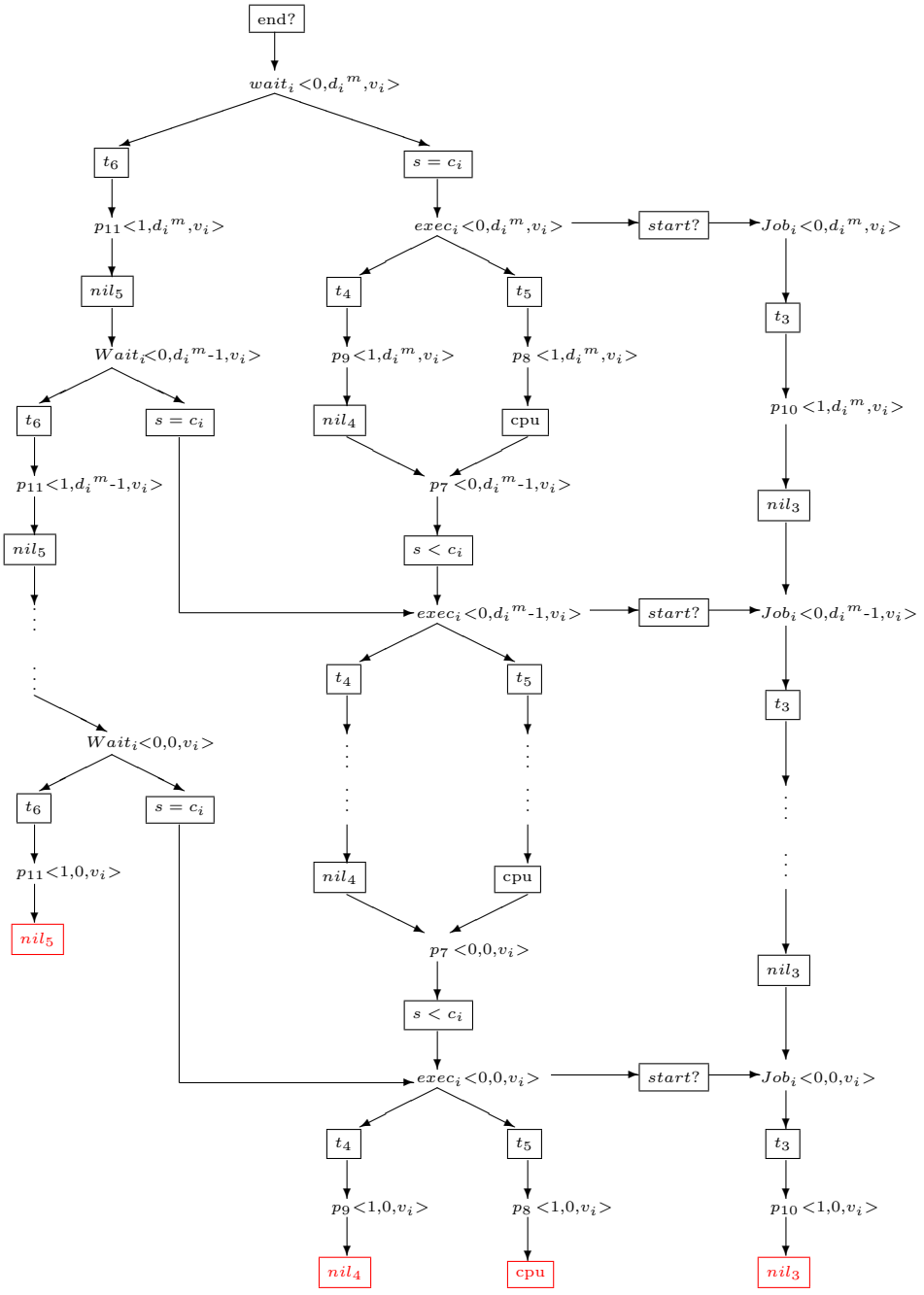


Abbildung 5.4: Backtrace für das Z-Pr/T-Netz $G_6{}'$

$$C = \begin{bmatrix} \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \\ < 0,0,v_i > + < 0,1,v_i > + ... + < 0,{d_i}^m - 1,v_i > \\ < 1,0,v_i > + < 1,1,v_i > + ... + < 1,{d_i}^m,v_i > \\ < 1,0,v_i > + < 1,1,v_i > + ... + < 1,{d_i}^m,v_i > \\ < 1,0,v_i > + < 1,1,v_i > + ... + < 1,{d_i}^m,v_i > \\ < 1,0,v_i > + < 1,1,v_i > + ... + < 1,{d_i}^m,v_i > \\ \emptyset \\ < 0,0,v_i > + < 0,1,v_i > + ... + < 0,{d_i}^m,v_i > \\ \emptyset \\ < 0,0,v_i > + < 0,1,v_i > + ... + < 0,{d_i}^m,v_i > \\ < 0,0,v_i > + < 0,1,v_i > + ... + < 0,{d_i}^m,v_i > \\ \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \end{bmatrix}$$

Der Stellenvektor C ist ein durch die Variablen ${d_i}^m$ und v_i parametrisierter Co-Trap für das Z-Pr/T-Netz ${G_6}'$.

Sei eine Prozessmenge V gegeben und gelte $0 \leq m < (kgV/p_i)$. Dann existiert für jeden Prozess $v_i \in V$ und jede $(m+1)$-te Prozessausführung ein zu C korrelierender, konstanter Co-Trap ${C_i}^m$. Ist eine Menge von Prozessen V durchführbar, so wird keiner dieser konstanten Co-Traps ${C_i}^m$ leer. Sind die in V gegebenen Prozesse jedoch nicht durchführbar, so existiert (mindestens) ein Co-Trap ${C_i}^m$, welcher leer wird. Da jede Transition, welche ein Tupel auf einer Stelle des Co-Traps produziert, auch wieder ein Tupel von einer Stelle des Co-Traps konsumiert, kann ein nicht-markierter Co-Trap somit nicht wieder markiert werden. Wird also einer der Co-Traps ${C_i}^m$ nicht-markiert, so kann auch die Eingangsstelle $Wait_i$ der Transition *end*? als Bestandteil des Co-Traps ${C_i}^m$ nicht wieder markiert werden. Aus der Existenz eines nicht-markierten Co-Traps kann somit unmittelbar auf das Nicht-Feuern der Transition *end*? für ein Tupel $< 0, {d_i}^m, v_i >$ geschlossen werden.

Sei zur Verdeutlichung das nachfolgende Beispiel gegeben.

Beispiel 19
Seien zwei Prozesse v_1 und v_2 wie in Beispiel 14 gegeben:

$$\begin{array}{lll} c_1 = 2 & d_1 = 2 & p_1 = 3 \\ c_2 = 2 & d_2 = 3 & p_2 = 3 \end{array}$$

Ein Ausführungszyklus ist dabei mit $[0, 3]$ gegeben, wobei beide Prozesse innerhalb des gegebenen Zeitraumes einmal zur Ausführung kommen müssen. Die Transition *end?* muss somit jeweils ein Mal für das Tupel $< 0, 2, v_1 >$ und ein Mal für das Tupel $< 0, 3, v_2 >$ feuern. Da Prozess v_1 derjenige Prozess mit der niedrigsten Deadline und somit mit der höchsten Priorität ist, kann dieser nicht von dem Prozess v_2 verdrängt werden, und kommt auf Grund der Bedingung $c_i \leq d_i$ zwangsläufig zur fristgerechten Beendigung. Demnach muss ausschließlich untersucht werden, ob die Transition *end?* für das mit Prozess v_2 assoziierte Tupel $< 0, 3, v_2 >$ feuert.
Indiziere ein Stellenvektor die Stellenmenge $\{\ p_1, p_2, ..., p_{12},\ Job_i,\ Act_i,\ Exec_i,\ Wait_i,\ CPU, C, D, \neg C\ \}$. Dann sei der mit Prozess v_2 und Deadline ${d_2}^0 = 3$ assoziierte Co-Trap ${C_2}^0$ wie folgt gegeben:

$${C_2}^0 = \begin{bmatrix} \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \\ < 0,0,v_2 > + < 0,1,v_2 > + < 0,2,v_2 > \\ < 1,0,v_2 > + < 1,1,v_2 > + < 1,2,v_2 > + < 1,3,v_2 > \\ < 1,0,v_2 > + < 1,1,v_2 > + < 1,2,v_2 > + < 1,3,v_2 > \\ < 1,0,v_2 > + < 1,1,v_2 > + < 1,2,v_2 > + < 1,3,v_2 > \\ < 1,0,v_2 > + < 1,1,v_2 > + < 1,2,v_2 > + < 1,3,v_2 > \\ \emptyset \\ < 0,0,v_2 > + < 0,1,v_2 > + < 0,2,v_2 > + < 0,3,v_2 > \\ \emptyset \\ < 0,0,v_2 > + < 0,1,v_2 > + < 0,2,v_2 > + < 0,3,v_2 > \\ < 0,0,v_2 > + < 0,1,v_2 > + < 0,2,v_2 > + < 0,3,v_2 > \\ \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \end{bmatrix}$$

Sei im Folgenden der Co-Trap ${C_2}^0$ zur Vereinfachung mit C_2 bezeichnet. Die Prozesse v_1 und v_2 sind nicht durchführbar, wenn der mit Prozess v_2 assoziierte Co-Trap C_2 leer bzw. nicht-markiert wird. Der Co-Trap C_2 ist unter einer Markierung M nicht-markiert, wenn gilt: ${C_2}^T \bullet M \leq 0$.

Für das gegebene Beispiel existiert mit der Markierung M' eine Folgemarkierung der Anfangsmarkierung M_0, so dass der Co-Trap C_2 unter der Markierung M' nicht-markiert ist (siehe Bsp. 14). D.h. es gilt ${C_2}^T \bullet M' = 0$:

$$\begin{bmatrix} \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \\ < 0,0,v_2 > + < 0,1,v_2 > + < 0,2,v_2 > \\ < 1,0,v_2 > + < 1,1,v_2 > + < 1,2,v_2 > + < 1,3,v_2 > \\ < 1,0,v_2 > + < 1,1,v_2 > + < 1,2,v_2 > + < 1,3,v_2 > \\ < 1,0,v_2 > + < 1,1,v_2 > + < 1,2,v_2 > + < 1,3,v_2 > \\ < 1,0,v_2 > + < 1,1,v_2 > + < 1,2,v_2 > + < 1,3,v_2 > \\ \emptyset \\ < 0,0,v_2 > + < 0,1,v_2 > + < 0,2,v_2 > + < 0,3,v_2 > \\ \emptyset \\ < 0,0,v_2 > + < 0,1,v_2 > + < 0,2,v_2 > + < 0,3,v_2 > \\ < 0,0,v_2 > + < 0,1,v_2 > + < 0,2,v_2 > + < 0,3,v_2 > \\ \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \end{bmatrix}^T \bullet \begin{bmatrix} \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \\ < 0,3,v_2 > \\ \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \\ < 0,3,v_2 > \\ < 0,3,v_1 > \\ < 0,3,v_1 > \\ \emptyset \\ \emptyset \\ <> \\ \emptyset \\ \emptyset \\ < v_2 > \end{bmatrix} = 0$$

Der Co-Trap C_2 ist somit unter der Markierung M' nicht-markiert. Da jede Transition, welche ein Tupel auf einer Stelle des Co-Traps produziert, auch wieder ein Tupel von einer Stelle des Co-Traps konsumiert, kann der Co-Trap C_2 auch nicht wieder markiert werden. Daraus folgt wiederum, dass auch die Stelle $Wait_i$ nicht mit einem Tupel $< 0,3,v_2 >$ markiert werden kann. Somit kann die Transition *end*? auch nicht für ein Tupel $< 0,3,v_2 >$ feuern. D.h. der Prozess v_2 verpasst seine Deadline und die Prozesse v_1 und v_2 sind somit nicht durchführbar. □

5.1.3 Fazit

Im Rahmen dieser Arbeit diente das EDF neben dem PIP (siehe Kap. 5.2) als Prüfstein für die Eignung von Z-Pr/T-Netzen zur Modellierung, Simulation und Analyse von Anwendungen aus dem Bereich der Echtzeitsysteme.
Das EDF wurde mittels Z-Pr/T-Netzen modelliert. Anschließend erfolgte eine Durchführbarkeitsanalyse auf Basis der strukturellen Analyse des zuvor modellierten Z-Pr/T-Netzes. Da die Korrektheit in Echtzeitsystemen sowohl die Korrektheit der Ergebnisse als auch deren fristgerechte Bereitstellung umfasst, stellt die Durchführbarkeitsanalyse von Prozessen einen zentralen Aspekt bei der Verifikation von Echtzeitsystemen dar.

Im Zuge der Durchführbarkeitsanalyse auf Basis der strukturellen Analyse des Z-Pr/T-Netzes $G_6{}'$ konnte eine Bedingung für die Durchführbarkeit von Prozessen extrahiert werden. Diese stellt – ohne Einschränkungen – sowohl eine notwendige als auch eine hinreichende Bedingung für die Durchführbarkeit von Prozessen dar. Im Gegensatz dazu ist die von C.L. Liu und J.W. Layland formulierte Ungleichung nur dann sowohl eine notwendige als auch hinreichende Bedingung für die Durchführbarkeit von Prozessen, wenn die Prämisse $d_i = p_i$ für alle Prozesse $v_i \in V$ gilt. Anderenfalls stellt sie lediglich eine hinreichende, aber keine notwendige Bedingung für die Durchführbarkeit von Prozessen dar.
Die Durchführbarkeit von Prozessen unter dem EDF kann mittels struktureller Analyse des Z-Pr/T-Netzes $G_6{}'$ auf die Identifikation nicht-markierter Co-Traps reduziert werden. Die Überprüfung der zu untersuchenden Co-Traps auf Markiertheit bzw. Nicht-Markiertheit ist jedoch unter Umständen mit einem hohen Rechenaufwand verbunden. Auf Grund der Komplexität der Berechnungen muss der praktische Nutzen dieser Durchführbarkeitsanalyse daher relativiert werden. Ungeachtet dessen zeigt die vorgenommene Durchführbarkeitsanalyse von Prozessen unter dem EDF, dass Z-Pr/T-Netze einen adäquaten Formalismus zur Modellierung, Simulation, vor allem aber auch zur Verifikation zeitbewerteter Systeme aus dem Anwendungsbereich der Echtzeitsysteme darstellen.

5.2 Das Priority-Inheritance-Protokoll

Das *Priority-Inheritance-Protokoll (PIP)* [37] zählt ebenso wie das EDF zu den prioritätsbasierten, dynamischen Planungsverfahren. D.h. die Zuteilung von Betriebsmitteln und der Zugriff auf den Prozessor werden zur Laufzeit auf Basis der aktuellen Prioritäten gefällt. Bei dem PIP kann jedoch das Problem der *Prioritätsumkehr* vermieden werden.

5.2.1 Prioritätsumkehr

Das Phänomen der *Prioritätsumkehr* wird durch die gemeinsame Nutzung von Betriebsmitteln gleichzeitig aktiver Prozesse bedingt.
Fordert ein Prozess Zugriff auf ein exklusiv nutzbares Betriebsmittel, so muss er dieses für andere Prozesse sperren. Der Prozess geht in sein sogenanntes *kritisches Gebiet* über. Konkurrieren mehrere Prozesse um ein und dasselbe Betriebsmittel, so soll immer derjenige Prozess mit der momentan höchsten Priorität Zugriff auf eben dieses erhalten. Dabei schildert folgendes Szenario das Problem der Prioritätsumkehr ([47]).
Seien drei Prozesse v_1, v_2 und v_3 in aufsteigender Reihenfolge bzgl. ihrer Prioritäten gegeben. D.h. es soll $\pi_1 < \pi_2 < \pi_3$ gelten. Angenommen der niedrig

priorisierte Prozess v_1 fordert initial Zugriff auf das Betriebsmittel R. Dieser wird ihm gewährt, so dass der Prozess v_1 das Betriebsmittel R sperren und in sein kritisches Gebiet übergehen kann. Fordert nun der hoch priorisierte Prozess v_3 ebenfalls Zugriff auf das Betriebsmittel R, so wird der Prozess v_3 von dem niedriger priorisierten Prozess v_1 blockiert, da dieser die Sperre auf das Betriebsmittel R hält. Wird nun der Prozess v_2 aktiv, so kann dieser auf Grund seiner Priorität den Prozess v_1 und somit indirekt auch den höher priorisierten Prozess v_3 für die gesamte Dauer seiner Ausführungszeit verdrängen. Dies widerspricht jedoch der Priorisierung der Prozesse, d.h. die Prioritäten sind invertiert.

5.2.2 Das Original-Priority-Inheritance-Protokoll

Zur Vermeidung des Problems der Prioritätsumkehr wurde in dem PIP das Konzept der *Prioritätsvererbung* eingeführt. Dieser Mechanismus erlaubt es einem Prozess, seine Priorität temporär an einen anderen Prozess zu vererben. Dabei folgt die Prioritätsvererbung dem Prinzip, dass ein Prozess in seinem kritischen Gebiet immer mit der höchsten Priorität von allen ebenfalls um dieses Betriebsmittel konkurrierenden Prozesse ausgeführt wird.

Fordert ein Prozess Zugriff auf ein Betriebsmittel, welches durch keinen anderen Prozess gesperrt ist, so kann er dieses Betriebsmittel sperren und in sein kritisches Gebiet übergehen. Ist dieses Betriebsmittel jedoch bereits durch einen anderen Prozess gesperrt, so wird der anfordernde Prozess blockiert. Besitzt der anfordernde Prozess des Weiteren eine höhere Priorität als der blockierende Prozess, so vererbt der anfordernde Prozess dem blockierenden Prozess seine Priorität. Dies hat zur Konsequenz, dass derjenige Prozess, welcher im Besitz des Betriebsmittels ist, unter maximaler Priorität bezüglich aller von diesem (direkt oder indirekt) blockierten Prozesse ausgeführt wird.

Auf Basis der von Sha et al. in [37] formulierten, natürlich-sprachlichen Definition des PIPs haben Choi et al. eine Spezifikation des PIPs in ACSR-VP hergeleitet [2],[5]. Die Spezifikation des PIPs in ACSR-VP erlaubt eine Transformation der Spezifikation in ein korrespondierendes Z-Pr/T-Netz gemäss der Erläuterungen in Anhang 6. Abbildung 5.4 zeigt die ACSR-VP Spezifikation des PIPs nach Choi et al. [2],[5]; Abbildung 5.5 das korrelierende Z-Pr/T-Netz G_7.

Die ACSR-VP Spezifikation des PIP nach Choi et al. [2],[5] dient hierbei nur als Grundlage für die Modellierung des PIP mittels Z-Pr/T-Netzen. Das resultierende Z-Pr/T-Netz bedarf darüber hinaus weiterer Modifikationen.

Die ACSR-VP Spezifikation von Choi et al. erlaubt keine Schachtelung kritischer Gebiete. Demnach kann ein Prozess nach der ACSR-VP Spezifikation von Choi et al. nicht mehrere Betriebsmittel gleichzeitig sperren. Dies bedeutet jedoch eine

$$
\begin{array}{rcl}
PIPSys & := & [\ (\ T_1 \parallel T_2 \parallel ... \parallel T_n \mid P\ ) \backslash \{req, chan, p, v\}]_{\{cpu\}} \\
T_i & := & (\ Job_i \parallel Activator_i\ ) \setminus \{start, end\} \\
\hline
Job_i & := & \emptyset : Job_i \ +\ (start?, 1)\ .\ Exec_i(0,0) \\
Exec_i(s,t) & := & (s < c_i \wedge s \neq cs_i) \longrightarrow (\{(cpu, \pi_i)\} : Exec_i(s+1, t+1) \ +\ \emptyset : Exec_i(s, t+1)) \\
 & & +\ (s = cs_i) \longrightarrow ((req!\pi, \pi)\ .\ Req_i(s,t) \ +\ \emptyset : Exec_i(s, t+1)) \\
 & & +\ (s = c_i) \longrightarrow Wait_i \\
Wait_i & := & \emptyset : Wait_i \ +\ (end?, 1)\ .\ Job_i \\
\hline
Req_i(s,t) & := & (p!\pi, \pi)\ .\ CS_i(s,t,\pi) + \emptyset : Req_i(s, t+1) \\
CS_i(s,t,\pi) & := & (s < cs_i + c_i') \longrightarrow (\{(cpu, \pi)\} : CS_i(s+1, t+1, \pi) \\
 & & \qquad +\ (chan?new, 1)\ .\ CS_i(s,t,new) \ +\ \emptyset : CS_i(s, t+1, \pi)) \\
 & & +\ (s = cs_i + c_i') \longrightarrow ((v!, 1)\ .\ Exec_i(s,t)) \\
\hline
P & := & (p?x, 1)\ .\ V(x) \ +\ (req?x, 1)\ .\ P \ +\ \emptyset : P \\
V(max) & := & (v?, 1)\ .\ P \ +\ \emptyset : V(max) \\
 & & +\ (req?x, 1)\ .\ ((x > max) \rightarrow ((chan!x, 1)\ .\ V(x)) \ +\ (x \leq max) \rightarrow V(max)) \\
\hline
Activator_i & := & (start!, 1)\ .\ \emptyset^{d_i} : (end!, 1)\ .\ \emptyset^{p_i - d_i} : Activator_i
\end{array}
$$

Abbildung 5.5: Spezifikation des PIPs in ACSR-VP nach Choi et al. [2],[5]

starke Vereinfachung der Komplexität des PIP. In dem zugehörigen Z-Pr/T-Netz G_7 wurde diese Einschränkung durch Einfügen zusätzlicher Struktur aufgehoben. Das Z-Pr/T-Netz G_7 wurde daher um die in Abbildung 4.5 blau gefärbten Elemente erweitert, so dass in dem resultierenden Z-Pr/T-Netz eine gleichzeitige Sperre mehrerer Betriebsmittel durch einen Prozess möglich ist.

Ist in dem Z-Pr/T-Netz G_7 für einen Prozess v_i der Guard $s = cs_i$ erfüllt, so benötigt der Prozess v_i Zugriff auf ein Betriebsmittel. Die Transition $s = cs_i$ schaltet und im Anschluss daran fordert der Prozess v_i durch Feuern der Transition $req_1!$ das entsprechende Betriebsmittel an. Ist dieses durch keinen anderen Prozess blockiert, so kann die Transition $req_1?$ feuern und der Prozess v_i kann in sein kritisches Gebiet übergehen. Ist nun der Guard $s = c_i{}'$ für den Prozess v_i erfüllt, so benötigt der Prozess v_i neben dem bereits gesperrten Betriebsmittel Zugriff auf ein weiteres Betriebsmittel. Dieses fordert er erneut durch Feuern der Transition $req_1!$ an. Ist dieses ebenfalls nicht durch einen anderen Prozess gesperrt, so kann die Transition $req_1?$ erneut feuern und der Prozess v_i wieder in sein kritisches Gebiet übergehen. Der Prozess v_i sperrt nun zwei Betriebsmittel. Benötigt der Prozess v_i eines dieser Betriebsmittel nicht mehr, so feuert die Transition $s = c_i{}''$. Der Prozess v_i gibt das entsprechende Betriebsmittel wieder frei. Da der Prozess v_i aber weiterhin die Sperre auf ein Betriebsmittel hält, setzt er seine Ausführung im kritischen Gebiet fort. Erst wenn er auch dieses Betriebsmittel nicht mehr benötigt, feuert die Transition $s = cs_i{}'$ und der Prozess gibt auch dieses Betriebsmittel wieder frei. Es gilt, dass die Transitionen

$s = c_i'$ und $s = c_i''$ höher priorisiert sind als die Transition $s < cs_i'$. Des Weiteren wurde dem Z-Pr/T-Netz G_7 die Transition $chan_2!$ hinzugefügt, welche die transitive Vererbung von Prioritäten modelliert. Dabei sei die Transition $chan_2!$ höher priorisiert als die Transition $chan_1!$.

Zur Verdeutlichung der Funktionsweise des PIPs sei das nachfolgende Beispiel gegeben.

Beispiel 20
Seien zwei Prozesse v_1 und v_2 mit den nachfolgenden Parametern gegeben:

	π_i	b_i	c_i	d_i	p_i
v_1	1	0	2	4	4
v_2	2	1	2	4	5

Des Weiteren sei die Ausführungssequenz des Prozesses v_1 mit (R, R) und die des Prozesses v_2 mit (E, R) gegeben. Dabei bezeichne E die Ausführung eines Prozesses mit Zugriff auf den Prozessor und R die Ausführung mit Zugriff auf das Betriebsmittel R (sowie den Prozessor). Daraus folgt unmittelbar:

	cs_i	cs_i'
v_1	0	2
v_2	1	1

Der Modulus der Variablen i sei $n_i = 5$, so dass die Werte der Variablen i auf einem Wertebereich $[0, 1, ..., 4]$ abgebildet werden. Der Modulus der Variablen j sei beliebig, aber hinreichend groß gewählt.

Ein S-Vektor indiziere im Folgenden die Stellenmenge $\{ p_2, p_5, p_{18}, Job_i, Act_i, Exec_i, Wait_i, Req_i, CS_i, free, blocked \}$. Die verbleibenden Stellen sind aus Gründen der Übersichtlichkeit nicht aufgeführt. Die nachfolgende Schaltsequenz stellt dann für das gegebene Beispiel einen möglichen Pfad im Erreichbarkeitsgraphen des Z-Pr/T-Netzes G_7 dar:

$$\begin{bmatrix} \emptyset \\ \emptyset \\ < 0,0,v_1 > + < 0,1,v_2 > \\ < 0,0,v_1 > + < 0,1,v_2 > \\ \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \\ < R > \\ \emptyset \end{bmatrix} \xrightarrow[\substack{\text{p?} \\ s < cs_i' \\ cpu_2 \\ nil_1 \\ t_7}]{\substack{\text{start!} \\ \text{start?} \\ s = cs_i \\ req_1! \\ req_1? \\ \text{p!}}} \begin{bmatrix} < 4,4,v_1 > \\ \emptyset \\ < 0,1,v_2 > \\ < 0,1,v_2 > \\ \emptyset \\ \emptyset \\ \emptyset \\ < 0,1,1,1,1,v_1 > \\ \emptyset \\ < R,1,v_1 > \end{bmatrix} \xrightarrow[\substack{nil_5 \\ t_5 \\ t_6}]{\substack{\text{start!} \\ \text{start?} \\ s < c_i \\ s < cs_i' \\ cpu_1 \\ nil_1}}$$

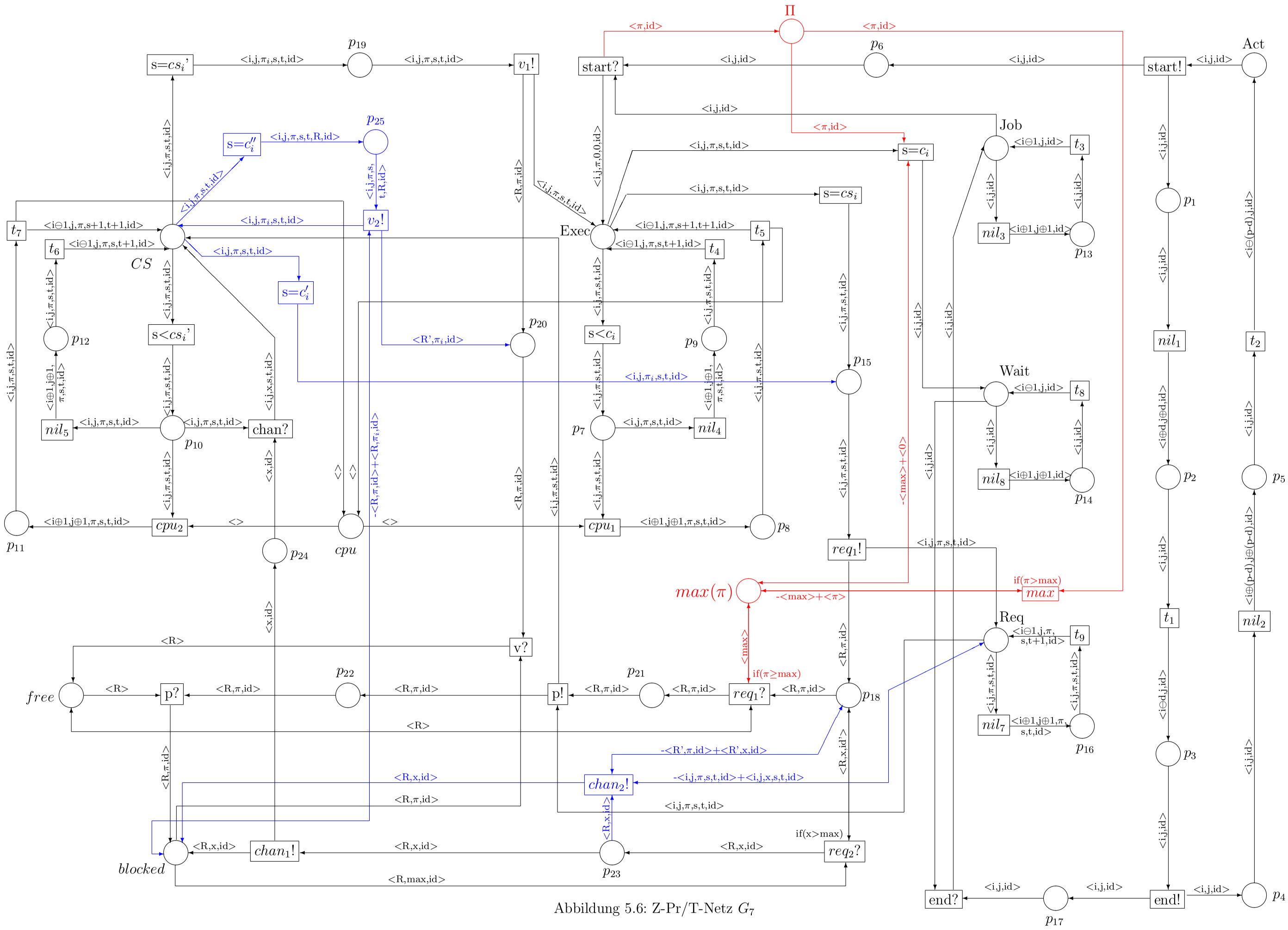
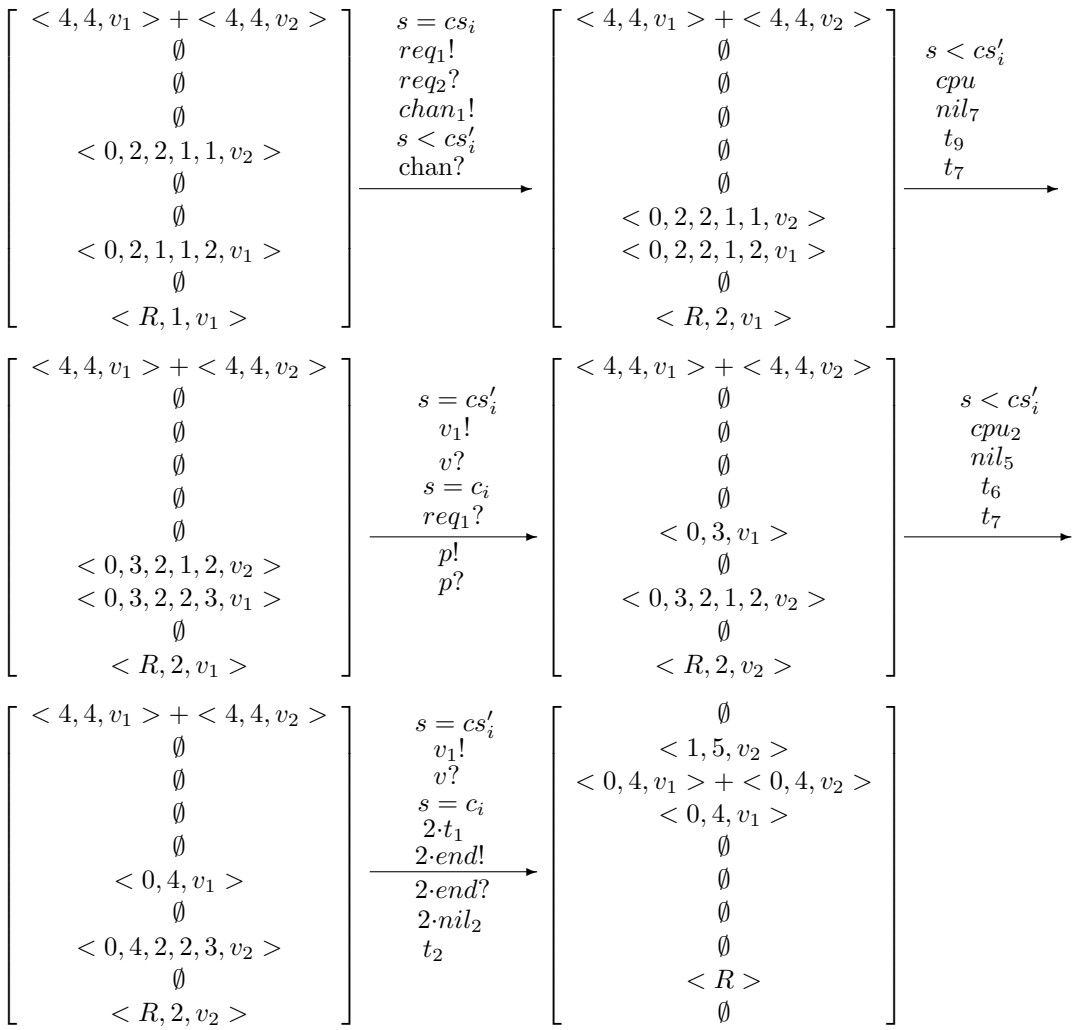


Abbildung 5.6: Z-Pr/T-Netz G_7



Initial wird der Prozess v_1 durch Feuern der Transitionen *start*! und *start*? für das mit Prozess v_1 assoziierte Tupel $< 0,0,v_1 >$ aktiviert. Der Prozess v_1 fordert unmittelbar Zugriff auf das Betriebsmittel R, welches in dem Z-Pr/T-Netz G_7 einem Feuern der Transition $req_1!$ entspricht. Das von Prozess v_1 angeforderte Betriebsmittel R ist durch keinen anderen Prozess gesperrt, so dass der Prozess v_1 dieses für sich sperren und in sein kritisches Gebiet übergehen kann. Dementsprechend ist in dem Z-Pr/T-Netz G_7 die Stelle $free$ mit dem Tupel $< R >$ markiert, so dass die Transitionen $req_1?$, $p!$ und $p?$ feuern können. Der Prozess v_1 kann nun mit der Ausführung in seinem kritischen Gebiet R beginnen und auf den Prozessor zugreifen.

Im Anschluss an diesen Ausführungsschritt wird der Prozess v_2 aktiviert. Die Prozesse v_1 und v_2 konkurrieren nun um den Prozessor. Auf Grund seiner höheren Priorität kann der Prozess v_2 auf den Prozessor zugreifen, wohingegen der Prozess v_1 warten muss. In dem Z-Pr/T-Netz G_7 feuert somit die Transition cpu_1 für das

mit Prozess v_2 assoziierte Tupel $< 0, 1, 2, 0, 0, v_2 >$ und die Transition nil_5 für das mit Prozess v_1 assoziierte Tupel $< 0, 1, 1, 1, 1, v_1 >$.

Daraufhin fordert der Prozess v_2 ebenfalls Zugriff auf das Betriebsmittel R, welches jedoch durch den Prozess v_1 gesperrt ist. In dem Z-Pr/T-Netz G_7 feuert somit die Transition $req_1!$, so dass die Stelle p_{18} mit einem Tupel $< R, 2, v_2 >$ markiert ist. Da die Stelle $free$ jedoch nicht markiert ist, kann die Transition $req_1?$ nicht feuern. Der Prozess v_1 blockiert somit den Prozess v_2. Da der Prozess v_2 jedoch eine höhere Priorität als der Prozess v_1 besitzt, vererbt der Prozess v_2 seine Priorität an den Prozess v_1. Die Prioritätsvererbung wird in dem Z-Pr/T-Netz G_7 durch Feuern der Transitionen $chan_1!$ und $chan?$ umgesetzt. Der Prozess v_1 kann nun mit Priorität $\pi_1 = 2$ auf den Prozessor zugreifen.

Anschließend gibt der Prozess v_1 das Betriebsmittel R wieder frei. Dabei fällt der Prozess v_1 wieder auf seine ursprüngliche Priorität $\pi_1 = 1$ zurück und wechselt in den Ruhezustand, da er seine Ausführung somit abgeschlossen hat.

Im Gegensatz dazu wird der Prozess v_2 wieder aktiviert. Dieser kann das Betriebsmittel R für sich sperren und mit Zugriff auf den Prozessor ebenfalls seine Ausführung beenden. □

Mittels des Mechanismus der Prioritätsvererbung erlaubt das von Sha et al. in [37] definierte *Priority-Inheritance-Protokoll (PIP)* eine Vermeidung des Problems der *Prioritätsumkehr*.

Die ursprüngliche Version des PIPs wurde jedoch natürlich-sprachlich, und nicht in einer formal-basierten Modellierungssprache definiert [37]. Eben diese fehlende formale Basis der ursprünglichen Protokolldefinition hat in Verbindung mit der nicht zu unterschätzenden Komplexität des Protokolls zu zahlreichen fehlerbehafteten Protokollspezifikationen sowie Implementierungen geführt. Im Folgenden wird ein wohlbekannter Fehler in der ursprünglichen Version des PIPs erläutert [29],[48],[49].

5.2.3 Das fehlerbereinigte Priority-Inheritance-Protokoll

Die ursprüngliche Spezifikation des PIPs beinhaltet einen Fehler, welcher sich auf die Aktualisierung von Prioritäten beim Verlassen von kritischen Gebieten bezieht [29],[48],[49]. So besagt die ursprüngliche Protokollspezifikation ([37], S.1177):

> *When a job exits a critical section, it resumes the priority it had at the point of entry into the critical section.*

Demnach fällt ein Prozess, der ein kritisches Gebiet verlässt, auf seine ursprüngliche Priorität zurück, welche er bei Eintritt in dieses kritische Gebiet inne hatte. Dabei werden jedoch alle Fälle vernachlässigt, bei denen der Prozess, der ein kritisches Gebiet verlässt, weiterhin (direkt oder indirekt) andere Prozesse blockiert. Die ursprüngliche Spezifikation des PIPs muss daher wie folgt abgeändert werden [25],[48],[49]:

> *When a job exits a critical section, it receives the maximum of its own priority and the priorities of those processes it still blocks (directly or indirectly).*

Ein Prozess, der ein kritisches Gebiet verlässt, fällt nun nicht mehr auf die Priorität zurück, welche er bei Eintritt in dieses kritische Gebiet inne hatte. Stattdessen entspricht seine neue Priorität dem Maximum seiner eigenen Priorität und aller Prioritäten derjenigen Prozesse, welche er weiterhin (direkt oder indirekt) blockiert.
Das Z-Pr/T-Netz G_7 inkludiert diese Modifikation der ursprünglichen Protokollspezifikation bereits. Dies wird an Hand des nachfolgenden Beispieles verdeutlicht (vgl. [49]).

Beispiel 21
Seien vier Prozesse v_1, v_2, v_3 und v_4 mit den folgenden Parametern gegeben:

	π_i	b_i	c_i	d_i	p_i
v_1	1	0	6	11	11
v_2	2	2	3	11	13
v_3	3	4	1	11	15
v_4	4	5	1	11	16

Die Ausführungssequenz von Prozess v_1 ist mit (S, ST, ST, ST, ST, S) gegeben, die von Prozess v_2 mit (U, UT, U), die von Prozess v_3 mit (S) und die von Prozess v_4 mit (U). Dabei bezeichne S die Ausführung eines Prozesses mit Zugriff auf das Betriebsmittel S, ST die Ausführung eines Prozesses mit Zugriff sowohl auf das Betriebsmittel S als auch auf das Betriebsmittel T, usw. Demnach gilt:

	cs_i	$cs_i{}'$	$c_i{}'$	$c_i{}''$
v_1	0	6	1	4
v_2	0	3	1	1
v_3	0	1		
v_4	0	1		

Der Modulus der Variablen i sei $n_i = 12$, so dass die Werte der Variablen i auf einem Wertebereich $[0, 1, ..., 11]$ abgebildet werden. Der Modulus der Variablen j sei beliebig, aber hinreichend groß gewählt.

Indiziere ein S-Vektor des Weiteren die Stellenmenge $\{ p_2, p_5, p_{18}, Job_i, Act_i, Exec_i, Wait_i, Req_i, CS_i, free, blocked \}$. Die nachfolgende Schaltsequenz stellt dann für das gegebene Beispiel einen möglichen Pfad im Erreichbarkeitsgraphen des Z-Pr/T-Netzes G_7 dar, wobei das Z-Pr/T-Netz G_7 ohne die rot gefärbten Elemente betrachtet wird:

$$\begin{bmatrix} \emptyset \\ \emptyset \\ \emptyset \\ <0,0,v_1> + <0,2,v_2> + <0,4,v_3> + <0,5,v_4> \\ <0,0,v_1> + <0,2,v_2> + <0,4,v_3> + <0,5,v_4> \\ \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \\ <S> + <T> + <U> \\ \emptyset \end{bmatrix} \xrightarrow[\substack{s < cs_i' \\ cpu_2 \\ nil_1 \\ t_7}]{\substack{\text{start!} \\ \text{start?} \\ s = cs_i \\ req_1! \\ req_1? \\ \text{p!} \\ \text{p?}}}$$

$$\begin{bmatrix} <11,11,v_1> \\ \emptyset \\ \emptyset \\ <0,2,v_2> + <0,4,v_3> + <0,5,v_4> \\ <0,2,v_2> + <0,4,v_3> + <0,5,v_4> \\ \emptyset \\ \emptyset \\ \emptyset \\ <0,1,1,1,1,v_1> \\ <T> + <U> \\ <S,1,v_1> \end{bmatrix} \xrightarrow[\substack{s < cs_i' \\ cpu_2 \\ t_7}]{\substack{s = c_i' \\ req_1! \\ req_1? \\ p! \\ p?}} \begin{bmatrix} <11,11,v_1> \\ \emptyset \\ \emptyset \\ <0,2,v_2> + <0,4,v_3> + <0,5,v_4> \\ <0,2,v_2> + <0,4,v_3> + <0,5,v_4> \\ \emptyset \\ \emptyset \\ \emptyset \\ <0,2,1,2,2,v_1> \\ <U> \\ <S,1,v_1> + <T,1,v_1> \end{bmatrix}$$

$$\xrightarrow[\substack{cpu_2 \\ nil_1 \\ nil_5 \\ t_6 \\ t_7}]{\substack{\text{start!} \\ \text{start?} \\ s = cs_i \\ req_1! \\ req_1? \\ p! \\ p? \\ 2(s < cs_i')}} \begin{bmatrix} <11,11,v_1> + <11,11,v_2> \\ \emptyset \\ \emptyset \\ <0,4,v_3> + <0,5,v_4> \\ <0,4,v_3> + <0,5,v_4> \\ \emptyset \\ \emptyset \\ \emptyset \\ <0,3,1,2,3,v_1> + <0,3,2,1,1,v_2> \\ \emptyset \\ <S,1,v_1> + <T,1,v_1> + <U,2,v_2> \end{bmatrix} \xrightarrow[\substack{cpu_2 \\ nil_7 \\ t_9 \\ t_7}]{\substack{s = c_i' \\ req_1! \\ req_2? \\ chan_1! \\ s < cs_i' \\ chan? \\ s < cs_i'}}$$

$$\left[\begin{array}{c} <11,11,v_1> + <11,11,v_2> \\ \emptyset \\ <T,2,v_2> \\ <0,4,v_3> + <0,5,v_4> \\ <0,4,v_3> + <0,5,v_4> \\ \emptyset \\ \emptyset \\ <0,4,2,1,2,v_2> \\ <0,4,2,3,4,v_1> \\ \emptyset \\ <S,1,v_1> + <T,2,v_1> + <U,2,v_2> \end{array}\right] \begin{array}{c} start! \\ start? \\ s = cs_i \\ req_1! \\ req_2? \\ chan_1! \\ s < cs_i' \\ chan? \\ \hline cpu_2 \\ s < cs_i' \\ nil_1 \\ 2{\cdot}nil_7 \\ 2{\cdot}t_9 \\ t_7 \end{array}$$

$$\left[\begin{array}{c} <11,11,v_1> + <11,11,v_2> + <11,11,v_3> \\ \emptyset \\ <T,2,v_2> + <S,3,v_3> \\ <0,5,v_4> \\ <0,5,v_4> \\ \emptyset \\ \emptyset \\ <0,5,2,1,3,v_2> + <0,5,3,0,1,,v_3> \\ <0,5,3,4,5,v_1> \\ \emptyset \\ <S,3,v_1> + <T,2,v_1> + <U,2,v_2> \end{array}\right] \begin{array}{c} start! \\ start? \\ s = cs_i \\ req_1! \\ req_2? \\ chan_2! \\ req_2? \\ chan_1! \\ s < cs_i' \\ chan? \\ \hline s < cs_i' \\ cpu_2 \\ 3{\cdot}nil_7 \\ 3{\cdot}t_9 \\ t_7 \end{array}$$

$$\left[\begin{array}{c} <11,11,v_1> + <11,11,v_2> + <11,11,v_3> + <11,11,v_4> \\ \emptyset \\ <S,3,v_3> + <T,4,v_2> + <U,4,v_4> \\ \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \\ <0,6,4,1,4,v_2> + <0,6,3,0,2,v_3> + <0,6,4,0,1,v_4> \\ <0,6,4,5,6,v_1> \\ \emptyset \\ <S,3,v_1> + <T,4,v_1> + <U,4,v_2> \end{array}\right] \begin{array}{c} s = c_i'' \\ v_2! \\ v? \\ req_2? \\ chan_1! \\ s < cs_i' \\ chan? \\ req_1? \\ p! \\ p? \\ \hline 2(s < cs_i') \\ cpu_2 \\ nil_5 \\ 2{\cdot}nil_7 \\ 2{\cdot}t_9 \\ t_6 \\ t_7 \end{array}$$

$$\left[\begin{array}{c} <11,11,v_1> + <11,11,v_2> + <11,11,v_3> + <11,11,v_4> \\ \emptyset \\ <S,3,v_3> + <U,4,v_4> \\ \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \\ <0,7,3,0,3,v_3> + <0,7,4,0,2,v_4> \\ <0,7,3,5,7,v_1> + <0,7,4,2,5,v_2> \\ \emptyset \\ <S,3,v_1> + <T,4,v_2> + <U,4,v_2> \end{array}\right] \begin{array}{c} s = c_i' \\ v_1! \\ v? \\ s = c_i'' \\ v_2! \\ v? \\ 2(s < cs_i') \\ cpu_2 \\ nil_5 \\ 2{\cdot}nil_7 \\ \hline 2{\cdot}t_9 \\ t_6 \\ t_7 \end{array}$$

$$\left[\begin{array}{c} <11,11,v_1> + <11,11,v_2> + <11,11,v_3> + <11,11,v_4> \\ \emptyset \\ <S,3,v_3> + <U,4,v_4> \\ \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \\ <0,8,3,0,4,v_3> + <0,8,4,0,3,v_4> \\ <0,8,3,5,8,v_1> + <0,8,4,3,6,v_2> \\ <T> \\ <S,3,v_1> + <U,4,v_2> \end{array}\right] \xrightarrow[\substack{nil_5 \\ nil_7 \\ nil_8 \\ t_8 \\ t_9 \\ t_6 \\ t_7}]{\substack{s = cs'_i \\ v_1! \\ v? \\ s = c_i \\ req_1? \\ p! \\ p? \\ cpu_2 \\ s < cs'_i}}$$

$$\left[\begin{array}{c} <11,11,v_1> + <11,11,v_2> + <11,11,v_3> + <11,11,v_4> \\ \emptyset \\ <S,3,v_3> \\ \emptyset \\ \emptyset \\ \emptyset \\ <0,9,v_2> \\ <0,9,3,0,5,v_3> \\ <0,9,3,5,9,v_1> + <0,9,4,1,4,v_4> \\ <T> \\ <S,3,v_1> + <U,4,v_4> \end{array}\right] \xrightarrow[\substack{t_9 \\ 2{\cdot}t_8 \\ t_7}]{\substack{s = cs'_i \\ v_1! \\ v? \\ s = c_i \\ s < cs'_i \\ cpu_2 \\ nil_7 \\ 2{\cdot}nil_8}}$$

$$\left[\begin{array}{c} <11,11,v_1> + <11,11,v_2> + <11,11,v_3> + <11,11,v_4> \\ \emptyset \\ <S,3,v_3> \\ \emptyset \\ \emptyset \\ \emptyset \\ <0,10,v_2> + <0,10,v_4> \\ <0,10,3,0,6,v_3> \\ <0,10,3,6,10,v_1> \\ <T> + <U> \\ <S,3,v_1> \end{array}\right] \xrightarrow[\substack{cpu_2 \\ 3{\cdot}nil_8 \\ 3{\cdot}t_8 \\ t_7}]{\substack{s = cs'_i \\ v_1! \\ v? \\ s = c_i \\ req_1? \\ p! \\ p? \\ s < cs'_i}}$$

$$\left[\begin{array}{c} <11,11,v_1> + <11,11,v_2> + <11,11,v_3> + <11,11,v_4> \\ \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \\ <0,11,v_1> + <0,11,v_2> + <0,11,v_4> \\ \emptyset \\ <0,11,3,1,7,v_3> \\ <T> + <U> \\ <S,3,v_3> \end{array}\right] \xrightarrow[\substack{4{\cdot}t_1 \\ 4{\cdot}end! \\ 4{\cdot}end? \\ 4{\cdot}nil_2 \\ t_2}]{\substack{s = cs'_i \\ v_1! \\ v? \\ s = c_i}}$$

$$\begin{bmatrix} \emptyset \\ < 2, 13, v_2 > + < 4, 15, v_3 > + < 5, 16, v_4 > \\ \emptyset \\ < 0, 11, v_1 > + < 0, 11, v_2 > + < 0, 11, v_3 > + < 0, 11, v_4 > \\ < 0, 11, v_1 > \\ \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \\ < S > + < T > + < U > \\ \emptyset \end{bmatrix}$$

Nach seiner Aktivierung beginnt der Prozess v_1 mit seiner Ausführung und sperrt zunächst das Betriebsmittel S und dann auch das Betriebsmittel T. Im Anschluss daran wird der Prozess v_2 aktiviert, welcher unmittelbar das Betriebsmittel U für sich sperrt. Die Prozesse v_1 und v_2 konkurrieren nun um den Zugriff auf den Prozessor, welcher dem Prozess v_2 gewährt wird.
Daraufhin fordert der Prozess v_2 ebenfalls Zugriff auf das Betriebsmittel T. Da dieses jedoch von Prozess v_1 gesperrt ist, wird der Prozess v_2 nun von Prozess v_1 blockiert. Der Prozess v_1 erbt die Priorität von Prozess v_2 und kann seine Ausführung mit Priorität $\pi_1 = 2$ fortsetzen.
Nun wird auch der Prozess v_3 aktiviert. Dieser fordert unmittelbar Zugriff auf das Betriebsmittel S. Da dieses jedoch von Prozess v_1 gesperrt ist, wird der Prozess v_3 nun – wie der Prozess v_2 auch – von Prozess v_1 blockiert. Der Prozess v_1 erbt die Priorität von Prozess v_3 und kann mit Priorität $\pi_1 = 3$ seine Ausführung fortsetzen.
Im Anschluss daran wird der Prozess v_4 aktiviert, welcher unmittelbar Zugriff auf das Betriebsmittel U fordert. Dieses ist jedoch von Prozess v_2 gesperrt. Der Prozess v_2 blockiert somit den Prozess v_4 und erbt die Priorität von Prozess v_4. Der Prozess v_2 wird jedoch seinerseits von Prozess v_1 blockiert. Da der Prozess v_1 somit indirekt auch den Prozess v_4 blockiert, erbt der Prozess v_1 wiederum die Priorität von Prozess v_4 und setzt seine Ausführung mit Priorität $\pi_1 = 4$ fort.
Anschließend gibt der Prozess v_1 das Betriebsmittel T wieder frei. Da der Prozess v_1 jedoch weiterhin den Prozess v_3 blockiert, setzt der Prozess v_1 seine Ausführung mit Priorität $\pi_1 = 3$ fort (und nicht – wie in der ursprünglichen Version des PIPs gefordert – mit der Priorität $\pi_1 = 1$). In dem Z-Pr/T-Netz G_7 feuert zunächst die Transition $v_2!$ und setzt die Priorität des Prozesses v_1 auf $\pi_1 = 1$. Auf Grund der Markierung der Stellen p_{18} und Req_i feuern jedoch unmittelbar die Transitionen $req_2?$, $chan_1!$ und $chan?$, so dass der Prozess v_1 seine Ausführung mit Priorität $\pi_1 = 3$ fortsetzt. Darüber hinaus wird auch der Prozess v_2 aktiviert. Dieser kann das Betriebsmittel T für sich sperren und in

sein kritisches Gebiet übergehen. Die Prozesse v_1 und v_2 konkurrieren nun um den Zugriff auf den Prozessor, welcher dem Prozess v_2 gewährt wird.
Im nächsten Ausführungsschritt gibt der Prozess v_2 das Betriebsmittel T wieder frei. Da der Prozess v_2 weiterhin den Prozess v_4 blockiert, setzt der Prozess v_2 seine Ausführung mit Priorität $\pi_2 = 4$ fort. Somit konkurrieren die Prozesse v_1 und v_2 erneut um den Prozessor, welcher wiederum dem Prozess v_2 gewährt wird.
Im Anschluss daran gibt der Prozess v_2 auch das Betriebsmittel U wieder frei. Er fällt auf die Priorität $\pi_2 = 2$ zurück. Da er seine Ausführung somit abgeschlossen hat, geht er in seinen Ausgangszustand über. Der Prozess v_4 hingegen wird wieder aktiviert. Dieser kann das Betriebsmittel U für sich sperren und seine Ausführung beginnen. Nun konkurrieren die Prozesse v_1 und v_4 um den Prozessor, welcher dem Prozess v_4 zugesprochen wird.
Anschliessend gibt der Prozess v_4 das Betriebsmittel U wieder frei. Er hat somit seine Ausführung beendet und geht in seinen Ausgangszustand über. Der Prozess v_1 kann nun auf den Prozessor zugreifen und seine Ausführung ebenfalls abschließen. Der Prozess v_1 gibt somit das Betriebsmittel S wieder frei. Daraufhin wird der Prozess v_3 wieder aktiviert, welcher das Betriebsmittel S für sich sperren und seine Ausführung ebenfalls abschließen kann. □

Mittels der modifizierten Protokollspezifikation lassen sich demnach invertierte Prioritäten vermeiden, welche in der ursprünglichen Spezifikation durch eine fehlerhafte Zuteilung von Prioritäten beim Verlassen von kritischen Gebieten bedingt wurden. Darüber hinaus erfordert die von Sha et al. in [37] formulierte Spezifikation des PIPs eine weitere Modifikation bzgl. der erneuten Vergabe frei gewordener Betriebsmittel [48],[49]. Dies wird im Folgenden erläutert.

5.2.4 Das Priority-Inheritance-Protokoll mit verzögerter Betriebsmittelvergabe

Neben dem bereits geschilderten Fehler beinhaltet die ursprüngliche Protokollspezifikation auf Grund ihrer Natürlichsprachlichkeit eine Ungenauigkeit, welche zu zahlreichen Missinterpretationen und somit zu einer Reihe von fehlerhaften Implementierungen geführt hat. Im Folgenden wird auf eine von Zöbel et al. [48],[49] aufgedeckte und weit verbreitete Fehlinterpretation der ursprünglichen Protokollspezifikation hingewiesen, welche die Vergabe frei gewordener Betriebsmittel betrifft.

Verlässt ein Prozess ein kritisches Gebiet, so wird das weitere Vorgehen in der ursprünglichen Spezifikation wie folgt geschildert ([37], S.1177):

> *When a job J exits its critical section, the binary semaphore associated with the critical section will be unlocked, and the highest priority job, if any, blocked by J will be awakened.*

Verlässt demnach ein Prozess sein kritisches Gebiet, so wird derjenige Prozess aktiviert, welcher von allen wartenden Prozessen die höchste Priorität besitzt. Die Spezifikation lässt jedoch offen, *wann* das frei gewordene Betriebsmittel erneut vergeben werden soll. Diese sprachliche Ungenauigkeit hat zu der weit verbreiteten Fehlinterpretation geführt, dass frei gewordene Betriebsmittel unmittelbar wieder vergeben werden. Eine derartige Interpretation steht jedoch in Widerspruch zu dem von Sha et al. formulierten Theorem 6 ([37], S. 1178):

> *Theorem 6: Under the basic priority inheritance protocol, if there are m semaphores wich can block job J, then J can be blocked at most m times.*

So besagt obiges Theorem, dass ein Prozess, der durch m Semaphore blockiert werden kann, auch tatsächlich höchstens m Mal blockiert wird. Die unmittelbare Vergabe frei gewordener Betriebsmittel steht jedoch in direktem Widerspruch zu obigem Theorem. Zur Verdeutlichung sei das nachfolgende Beispiel gegeben (vgl. [49]).

Beispiel 22
Seien vier Prozesse v_1, v_2, v_3 und v_4 mit den nachfolgenden Parametern gegeben:

	π_i	b_i	c_i	d_i	p_i
v_1	1	0	4	12	12
v_2	2	1	1	12	13
v_3	3	2	4	12	14
v_4	4	4	4	12	16

Sei die Ausführungssequenz von Prozess v_1 mit (S, S, S, S) gegeben, die von Prozess v_2 mit (S), die Ausführungssequenz von Prozess v_3 mit (T, ST, T) und die Ausführungssequenz von Prozess v_4 mit (T, ST, T) gegeben. Dabei bezeichne S erneut die Ausführung eines Prozesses mit Zugriff auf das Betriebsmittel S, T die Ausführung eines Prozesses mit Zugriff auf das Betriebsmittel T und ST die Ausführung eines Prozesses mit Zugriff sowohl auf das Betriebsmittel S als auch auf das Betriebsmittel T. Daraus folgt unmittelbar:

	cs_i	$cs_i{}'$	$c_i{}'$	$c_i{}''$
v_1	0	4		
v_2	0	1		
v_3	0	3	1	1
v_4	0	3	1	1

Der Modulus der Variablen i ist $n_i = 13$, so dass die Werte der Variablen i auf einem Wertebereich $[0, 1, ..., 12]$ abgebildet werden. Der Modulus der Variablen j sei beliebig, aber hinreichend groß gewählt.
Indiziere ein S-Vektor des Weiteren die Stellenmenge $\{ p_2, p_5, p_{18}, Job_i, Act_i, Exec_i, Wait_i, Req_i, CS_i, free, blocked \}$. Dann stellt die nachfolgende Schaltsequenz für das Z-Pr/T-Netz G_7 ohne die rot gefärbten Elemente einen möglichen Pfad im Erreichbarkeitsgraphen dar:

$$\left[\begin{array}{c} \emptyset \\ \emptyset \\ \emptyset \\ <0,0,v_1> + <0,1,v_2> + <0,2,v_3> + <0,4,v_4> \\ <0,0,v_1> + <0,1,v_2> + <0,2,v_3> + <0,4,v_4> \\ \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \\ <S> + <T> \\ \emptyset \end{array}\right] \xrightarrow[\substack{cpu_2 \\ nil1 \\ t_7}]{\substack{\text{start!} \\ \text{start?} \\ s = cs_i \\ req_1! \\ req_1? \\ p! \\ p? \\ s < cs_i'}}$$

$$\left[\begin{array}{c} <12,12,v_1> \\ \emptyset \\ \emptyset \\ <0,1,v_2> + <0,2,v_3> + <0,4,v_4> \\ <0,1,v_2> + <0,2,v_3> + <0,4,v_4> \\ \emptyset \\ \emptyset \\ \emptyset \\ <0,1,1,1,1,v_1> \\ <T> \\ <S,1,v_1> \end{array}\right] \xrightarrow[\substack{cpu_2 \\ nil_1 \\ nil_7 \\ t_9 \\ t_7}]{\substack{\text{start!} \\ \text{start?} \\ s = cs_i \\ req_1! \\ req_2? \\ chan_1! \\ s < cs_i' \\ chan?' \\ s < cs_i'}} \left[\begin{array}{c} <12,12,v_1> + <12,12,v_2> \\ \emptyset \\ <S,2,v_2> \\ <0,2,v_3> + <0,4,v_4> \\ <0,2,v_3> + <0,4,v_4> \\ \emptyset \\ \emptyset \\ <0,2,2,0,1,v_2> \\ <0,2,2,2,2,v_1> \\ <T> \\ <S,2,v_1> \end{array}\right]$$

$$\xrightarrow[\substack{cpu_2 \\ nil_1 \\ nil_5 \\ nil_7 \\ t_9 \\ t_6 \\ t_7}]{\substack{\text{start!} \\ \text{start?} \\ s = cs_i \\ req_1! \\ req_1? \\ \text{p!} \\ \text{p?} \\ 2{\cdot}(s < cs_i')}} \left[\begin{array}{c} <12,12,v_1> + <12,12,v_2> + <12,12,v_3> \\ \emptyset \\ <S,2,v_2> \\ <0,4,v_4> \\ <0,4,v_4> \\ \emptyset \\ \emptyset \\ <0,3,2,0,2,v_2> \\ <0,3,2,2,3,v_1> + <0,3,3,1,1,v_3> \\ \emptyset \\ <S,2,v_1> + <T,3,v_3> \end{array}\right] \xrightarrow[\substack{nil_1 \\ 2{\cdot}nil_7 \\ 2{\cdot}t_9 \\ t_7}]{\substack{s = c_i' \\ req_1! \\ req_2? \\ chan_1! \\ s < cs_i' \\ chan?' \\ s < cs_i' \\ cpu_2}}$$

$$\left[\begin{array}{c} <12,12,v_1> + <12,12,v_2> + <12,12,v_3> \\ \emptyset \\ <S,2,v_2> + <S,3,v_3> \\ <0,4,v_4> \\ <0,4,v_4> \\ \emptyset \\ \emptyset \\ <0,4,2,0,3,v_2> + <0,4,3,1,2,v_3> \\ <0,4,3,3,4,v_1> \\ \emptyset \\ <S,3,v_1> + <T,3,v_3> \end{array}\right] \xrightarrow[\begin{array}{c} s = cs_i \\ req_1! \\ req_2? \\ chan_2! \\ req_2? \\ chan_1! \\ s < cs_i' \\ chan? \end{array}]{\begin{array}{c} start! \\ start? \end{array}} \begin{array}{c} s < cs_i' \\ cpu_2 \\ 3{\cdot}nil_7 \\ 3{\cdot}t_9 \\ t_7 \end{array}$$

$$\left[\begin{array}{c} <12,12,v_1> + <12,12,v_2> + <12,12,v_3> + <12,12,v_4> \\ \emptyset \\ <S,2,v_2> + <S,4,v_3> + <T,4,v_4> \\ \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \\ <0,5,2,0,4,v_2> + <0,5,4,1,3,v_3> + <0,5,4,0,1,v_4> \\ <0,5,4,4,5,v_1> \\ \emptyset \\ <S,4,v_1> + <T,4,v_3> \end{array}\right] \xrightarrow[\begin{array}{c} v_1! \\ v? \\ s = c_i \\ req_1? \\ p! \\ p? \\ s < cs_i' \\ cpu_2 \end{array}]{s = cs_i'} \begin{array}{c} nil_8 \\ 2{\cdot}nil_7 \\ 2{\cdot}t_9 \\ t_8 \\ t_7 \end{array}$$

$$\left[\begin{array}{c} <12,12,v_1> + <12,12,v_2> + <12,12,v_3> + <12,12,v_4> \\ \emptyset \\ <S,2,v_2> + <T,4,v_4> \\ \emptyset \\ \emptyset \\ \emptyset \\ <0,6,v_1> \\ <0,6,2,0,5,v_2> + <0,6,4,0,2,v_4> \\ <0,6,4,2,4,v_3> \\ \emptyset \\ <S,4,v_3> + <T,4,v_3> \end{array}\right] \xrightarrow[\begin{array}{c} v_2! \\ v? \\ req_2? \\ chan_1! \\ s < cs_i' \\ chan? \\ req_1? \\ p! \\ p? \\ 2{\cdot}(s < cs_i') \end{array}]{s = c_i''} \begin{array}{c} cpu_2 \\ nil_5 \\ nil_8 \\ nil_7 \\ t_8 \\ t_9 \\ t_6 \\ t_7 \end{array}$$

$$\left[\begin{array}{c} <12,12,v_1> + <12,12,v_2> + <12,12,v_3> + <12,12,v_4> \\ \emptyset \\ <T,4,v_4> \\ \emptyset \\ \emptyset \\ \emptyset \\ <0,7,v_1> \\ <0,7,4,0,3,v_4> \\ <0,7,2,0,6,v_2> + <0,7,4,3,5,v_3> \\ \emptyset \\ <S,2,v_2> + <T,4,v_3> \end{array}\right] \xrightarrow[\begin{array}{c} v_1! \\ v? \\ s = c_i \\ req_1? \\ p! \\ p? \\ 2{\cdot}(s < cs_i') \\ cpu_2 \end{array}]{s = cs_i'} \begin{array}{c} nil_5 \\ 2{\cdot}nil_8 \\ 2{\cdot}t_8 \\ t_6 \\ t_7 \end{array}$$

$$\left[\begin{array}{c} <12,12,v_1> + <12,12,v_2> + <12,12,v_3> + <12,12,v_4> \\ \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \\ <0,8,v_1> + <0,8,v_3> \\ \emptyset \\ <0,8,2,0,7,v_2> + <0,8,4,1,4,v_4> \\ \emptyset \\ <S,2,v_2> + <T,4,v_4> \end{array}\right] \xrightarrow[\substack{nil_7 \\ 2{\cdot}t_8 \\ t_9 \\ t_7}]{\substack{s = c_i' \\ req_1! \\ req_2? \\ chan_1! \\ s < cs_i' \\ chan? \\ s < cs_i' \\ cpu_2 \\ 2{\cdot}nil_8}}$$

$$\left[\begin{array}{c} <12,12,v_1> + <12,12,v_2> + <12,12,v_3> + <12,12,v_4> \\ \emptyset \\ <S,4,v_4> \\ \emptyset \\ \emptyset \\ \emptyset \\ <0,9,v_1> + <0,9,v_3> \\ <0,9,4,1,5,v_4> \\ <0,9,4,1,8,v_2> \\ \emptyset \\ <S,4,v_2> + <T,4,v_4> \end{array}\right] \xrightarrow[\substack{cpu_2 \\ 3{\cdot}nil_8 \\ 3{\cdot}t_8 \\ t_7}]{\substack{s = cs_i' \\ v_1! \\ v? \\ s = c_i \\ req_1? \\ p! \\ p? \\ s < cs_i'}}$$

$$\left[\begin{array}{c} <12,12,v_1> + <12,12,v_2> + <12,12,v_3> + <12,12,v_4> \\ \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \\ <0,10,v_1> + <0,10,v_2> + <0,10,v_3> \\ \emptyset \\ <0,10,4,2,9,v_4> \\ \emptyset \\ <S,4,v_4> + <T,4,v_4> \end{array}\right] \xrightarrow[t_7]{\substack{s = c_i'' \\ v_2! \\ v? \\ s < cs_i' \\ cpu_2 \\ 3{\cdot}nil_8 \\ 3{\cdot}t_8}}$$

$$\left[\begin{array}{c} <12,12,v_1> + <12,12,v_2> + <12,12,v_3> + <12,12,v_4> \\ \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \\ <0,11,v_1> + <0,11,v_2> + <0,11,v_3> \\ \emptyset \\ <0,11,4,3,10,v_4> \\ <S> \\ <T,4,v_4> \end{array}\right] \xrightarrow[\substack{4{\cdot}end? \\ 4{\cdot}nil_2 \\ t_2}]{\substack{s = cs_i' \\ v_1! \\ v? \\ s = c_i \\ 4{\cdot}nil_8 \\ 4{\cdot}t_8 \\ 4{\cdot}t_1 \\ 4{\cdot}end!}}$$

$$\begin{bmatrix} \emptyset \\ <1,13,v_2> + <2,14,v_3> + <4,16,v_4> \\ \emptyset \\ <0,12,v_1> + <0,12,v_2> + <0,12,v_3> + <0,12,v_4> \\ <0,12,v_1> \\ \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \\ <S> + <T> \\ \emptyset \end{bmatrix}$$

Nach seiner Aktivierung kann der Prozess v_1 das Betriebsmittel S für sich sperren und mit seiner Ausführung beginnen. Im Anschluss daran wird der Prozess v_2 aktiviert und fordert ebenfalls Zugriff auf das Betriebsmittel S. Da dieses jedoch von dem Prozess v_1 gesperrt ist, blockiert der Prozess v_1 nun den Prozess v_2 und setzt seine Ausführung mit Priorität $\pi_1 = 2$ fort.

Anschließend wird der Prozess v_3 aktiviert, welcher unmittelbar das Betriebsmittel T für sich sperren kann. Die Prozesse v_1 und v_3 konkurrieren nun um den Zugriff auf den Prozessor, welcher dem Prozess v_3 gewährt wird. Daraufhin fordert der Prozess v_3 ebenfalls Zugriff auf das Betriebsmittel S. Somit blockiert der Prozess v_1 nun sowohl den Prozess v_2 als auch den Prozess v_3 und setzt seine Ausführung mit Priorität $\pi_1 = 3$ fort.

Nun wird auch der Prozess v_4 aktiviert, welcher unmittelbar Zugriff auf das Betriebsmittel T fordert. Da dieses jedoch von Prozess v_3 gesperrt ist, blockiert der Prozess v_3 den Prozess v_4 für die Dauer seines kritischen Gebietes T. Der Prozess v_3 wird jedoch seinerseits von dem Prozess v_1 blockiert. D.h. der Prozess v_3 und somit auch der Prozess v_4 warten auf die Beendigung des kritischen Gebietes S durch den Prozess v_1. Dieser setzt seine Ausführung mit Priorität $\pi_1 = 4$ fort und gibt anschließend das Betriebsmittel S wieder frei. Somit kann der Prozess v_3 dieses für sich sperren und mit seiner Ausführung fortfahren.

Im Anschluss daran gibt der Prozess v_3 das Betriebsmittel S wieder frei. Nach der ursprünglichen Spezifikation des PIPs kann nun der Prozess v_2 das Betriebsmittels S für sich sperren. Somit konkurrieren die Prozesse v_2 und v_3 um den Prozessor, welcher dem Prozess v_3 zugesprochen wird. Der Prozess v_3 gibt daraufhin das Betriebsmittel T wieder frei, welches der Prozess v_4 für sich sperren kann. Die Prozesse v_2 und v_4 konkurrieren nun um den Zugriff auf den Prozessor, welcher dem Prozess v_4 gewährt wird.

Anschließend fordert der Prozess v_4 ebenfalls Zugriff auf das Betriebsmittel S. Dieses ist jedoch durch den Prozess v_2 gesperrt und der Prozess v_4 wird für die Dauer des kritischen Gebietes S von dem Prozess v_2 blockiert. Erst nachdem der

Prozess v_2 seine Ausführung beendet und das Betriebsmittel S wieder freigegeben hat, kann der Prozess v_4 das Betriebsmittel S für sich sperren und ebenfalls seine Ausführung beenden.

In dem gegebenen Beispiel fordert der Prozess v_4 Zugriff auf die Betriebsmittel S und T. Gemäss dem Theorem 6 von Sha et al. ([37], S.1178) darf der Prozess v_4 demnach maximal zwei Mal blockiert werden. Der Prozess v_4 muss jedoch zunächst auf die Beendigung des kritischen Gebietes T durch den Prozess v_3 und auf die Beendigung des kritischen Gebietes S sowohl durch den Prozess v_1 als auch den Prozess v_2 warten. Eine unmittelbare Weitergabe frei gewordener Betriebsmittel steht also in Widerspruch zu dem Theorem 6 von Sha et al. ([37], S.1178). □

In die ursprüngliche Protokollspezifikation muss dementsprechend eine verzögerte Vergabe frei gewordener Betriebsmittel integriert werden. Eine derartige Adaption der ursprünglichen Protokollspezifikation umfasst gemäss Zöbel et al. [48],[49] folgende zwei Regeln:

> *(d1) When a critical section is left and there are processes blocked to enter this critical section and there are ready processes with higher priorities, then the blocked processes remain in their status and the critical section is registered free although there exists a set of processes attempting to enter it.*
>
> *(d2) The processor is assigned to the process with maximum priority wich is either in the status ready or in the status blocked by a critical section wich is registered free.*

Demnach kann ein Prozess nur dann ein Betriebsmittel für sich sperren, wenn kein anderer Prozess existiert, der mit einer höheren Priorität ausführbereit ist. Diese Modifikation in der Protokollspezifikation verhindert die Vergabe von Betriebsmitteln an Prozesse, denen bei dem Konkurrieren um den Prozessor auf Grund ihrer Priorität der Zugriff auf eben diesen verwehrt wird.

Die verzögerte Vergabe frei gewordener Betriebsmittel wird in dem Z-Pr/T-Netz G_7 durch die rot gefärbten Elemente realisiert. Die Markierung der Stelle $max(\pi)$ entspricht dabei der maximalen Priorität aller aktiven Prozesse. Ein Prozess kann nur dann ein Betriebsmittel für sich sperren, wenn seine Priorität größer oder gleich dieser maximalen Priorität ist. Daher wurde der Transition $req_1?$ der Guard $if(\pi \geq max)$ hinzugefügt, wobei die Transition max höher priorisiert sei als die Transition $req_1?$. Des Weiteren ist die Stelle $max(\pi)$ initial mit einem Tupel $< 0 >$ markiert, wohingegen die Stelle Π initial nicht markiert ist.

Beispiel 23

Seien erneut die Prozesse v_1, v_2, v_3 und v_4 wie in Beispiel 22 gegeben. Dabei sei erneut die Ausführungssequenz von Prozess v_1 mit (S, S, S, S) gegeben, für Prozess v_2 mit (S) , für Prozess v_3 mit (T, ST, T) und für Prozess v_4 mit (T, ST, T) .

Indiziere ein S-Vektor des Weiteren die Stellenmenge $\{ p_2, p_5, p_{18}, Job_i, Act_i, Exec_i, Wait_i, Req_i, CS_i, free, blocked, \Pi, max(\pi) \}$. Dann stellt die nachfolgende Schaltsequenz einen möglichen Pfad im Erreichbarkeitsgraphen des Z-Pr/T-Netzes G_7 dar, wobei das Z-Pr/T-Netz G_7 inklusive der rot gefärbten Elemente gegeben sei:

$$\left[\begin{array}{c} \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \\ < 0,0,v_1 > + < 0,1,v_2 > + < 0,2,v_3 > + < 0,4,v_4 > \\ < 0,0,v_1 > + < 0,1,v_2 > + < 0,2,v_3 > + < 0,4,v_4 > \\ \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \\ < S > + < T > \\ \emptyset \\ \emptyset \\ < 0 > \end{array}\right] \xrightarrow[\substack{s < cs_i' \\ cpu_2 \\ nil1 \\ t_7}]{\substack{\text{start!} \\ \text{start?} \\ s = cs_i \\ req_1! \\ req_1? \\ p! \\ p?}} \cdots\cdots\cdots \xrightarrow[\substack{nil_8 \\ 2{\cdot}nil_7 \\ 2{\cdot}t_9 \\ t_8 \\ t_7}]{\substack{s = cs_i' \\ v_1! \\ v? \\ s = c_j \\ req_1? \\ p! \\ p? \\ s < cs_i' \\ cpu_2}}$$

$$\left[\begin{array}{c} < 12,12,v_1 > + < 12,12,v_2 > + < 12,12,v_3 > + < 12,12,v_4 > \\ \emptyset \\ < S,2,v_2 > + < T,4,v_4 > \\ \emptyset \\ \emptyset \\ \emptyset \\ < 0,6,v_1 > \\ < 0,6,2,0,5,v_2 > + < 0,6,4,0,2,v_4 > \\ < 0,6,4,2,4,v_3 > \\ \emptyset \\ < S,4,v_3 > + < T,4,v_3 > \\ < 2,v_2 > + < 3,v_3 > + < 4,v_4 > \\ < 4 > \end{array}\right] \xrightarrow[\substack{s < cs_i' \\ cpu_2 \\ 2{\cdot}nil_7 \\ nil_8 \\ t_8 \\ 2{\cdot}t_9 \\ t_7}]{\substack{s = c_i'' \\ v_2! \\ v?}}$$

$$\left[\begin{array}{c} < 12,12,v_1 > + < 12,12,v_2 > + < 12,12,v_3 > + < 12,12,v_4 > \\ \emptyset \\ < S,2,v_2 > + < T,4,v_4 > \\ \emptyset \\ \emptyset \\ \emptyset \\ < 0,7,v_1 > \\ < 0,7,2,0,6,v_2 > + < 0,7,4,0,3,v_4 > \\ < 0,7,4,3,5,v_3 > \\ < S > \\ < T,4,v_3 > \\ < 2,v_2 > + < 3,v_3 > + < 4,v_4 > \\ < 4 > \end{array}\right] \xrightarrow[\substack{2{\cdot}nil_8 \\ nil_7 \\ 2{\cdot}t_8 \\ t_9 \\ t_7}]{\substack{s = cs_i' \\ v_1! \\ v? \\ s = c_i \\ max \\ req_1? \\ p! \\ p? \\ s < cs_i' \\ cpu_2}}$$

$$\left[\begin{array}{c} <12,12,v_1> + <12,12,v_2> + <12,12,v_3> + <12,12,v_4> \\ \emptyset \\ <S,2,v_2> \\ \emptyset \\ \emptyset \\ \emptyset \\ <0,8,v_1> + <0,8,v_3> \\ <0,8,2,0,7,v_2> \\ <0,8,4,1,4,v_4> \\ <S> \\ <T,4,v_4> \\ <2,v_2> + <4,v_4> \\ <4> \end{array}\right] \xrightarrow[\substack{req_1! \\ req_1? \\ p! \\ p? \\ s < cs_i' \\ cpu_2 \\ 2{\cdot}nil_8 \\ nil_7 \\ 2{\cdot}t_8 \\ t_9 \\ t_7}]{s = c_i'}$$

$$\left[\begin{array}{c} <12,12,v_1> + <12,12,v_2> + <12,12,v_3> + <12,12,v_4> \\ \emptyset \\ <S,2,v_2> \\ \emptyset \\ \emptyset \\ \emptyset \\ <0,9,v_1> + <0,9,v_3> \\ <0,9,2,0,8,v_2> \\ <0,9,4,2,5,v_4> \\ \emptyset \\ <S,4,v_4> + <T,4,v_4> \\ <2,v_2> + <4,v_4> \\ <4> \end{array}\right] \xrightarrow[\substack{v_2! \\ v? \\ s < cs_i' \\ cpu_2 \\ 2{\cdot}nil_8 \\ nil_7 \\ 2{\cdot}t_8 \\ t_9 \\ t_7}]{s = c_i''}$$

$$\left[\begin{array}{c} <12,12,v_1> + <12,12,v_2> + <12,12,v_3> + <12,12,v_4> \\ \emptyset \\ <S,2,v_2> \\ \emptyset \\ \emptyset \\ \emptyset \\ <0,10,v_1> + <0,10,v_3> \\ <0,10,2,0,9,v_2> \\ <0,10,4,3,6,v_4> \\ <S> \\ <T,4,v_4> \\ <2,v_2> + <4,v_4> \\ <4> \end{array}\right] \xrightarrow[\substack{v_1! \\ v? \\ s = c_i \\ max \\ req_1? \\ p! \\ p? \\ s < cs_i' \\ cpu_2 \\ 3{\cdot}nil_8 \\ 3{\cdot}t_8 \\ t_7}]{s = cs_i'}$$

$$\left[\begin{array}{c} < 12, 12, v_1 > + < 12, 12, v_2 > + < 12, 12, v_3 > + < 12, 12, v_4 > \\ \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \\ < 0, 11, v_1 > + < 0, 11, v_3 > + < 0, 11, v_4 > \\ \emptyset \\ < 0, 11, 2, 1, 10, v_2 > \\ < T > \\ < S, 2, v_2 > \\ < 2, v_2 > \\ < 2 > \end{array}\right] \xrightarrow{\begin{array}{c} s = cs_i' \\ v_1! \\ v? \\ s = c_i \\ 4{\cdot}nil_8 \\ 4{\cdot}t_8 \\ 4{\cdot}t_1 \\ 4{\cdot}end! \\ 4{\cdot}end? \\ 4{\cdot}nil_2 \\ t_2 \end{array}} \left[\begin{array}{c} \emptyset \\ < 1, 13, v_2 > + < 2, 14, v_3 > + < 4, 16, v_4 > \\ \emptyset \\ < 0, 12, v_1 > + < 0, 12, v_2 > + < 0, 12, v_3 > + < 0, 12, v_4 > \\ < 0, 12, v_1 > \\ \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \\ < S > + < T > \\ \emptyset \\ \emptyset \\ < 0 > \end{array}\right]$$

Es ergibt sich somit für das gegebene Beispiel auf Grund der aufgeführten Modifikationen eine abgeänderte Reihenfolge in der Ausführung. Gibt der Prozess v_3 nun das Betriebsmittel S wieder frei, so kann der Prozess v_2 dieses auf Grund der genannten Modifikationen nicht unmittelbar für sich sperren. In dem Z-Pr/T-Netz G_7 ist nach Freigabe des Betriebsmittels S durch den Prozess v_3 die Stelle p_{18} mit einem niedrigeren Wert für die Variable π markiert als die Stelle $max(\pi)$. Der Guard der Transition $req_1?$ ist nicht erfüllt. Dementsprechend kann die Transition $req_1?$ in dem Z-Pr/T-Netz G_7 nicht feuern und das Betriebsmittel S wird nicht unmittelbar vergeben.
Der Prozess v_3 beendet zunächst seine Ausführung und gibt das Betriebsmittel T wieder frei. Dieses kann nun der Prozess v_4 für sich sperren. In diesem Fall kann die Transition $req_1?$ feuern, da die Stelle p_{18} mit $< T, 4, v_4 >$ mit einem hinreichend großen Wert für die Variable π markiert ist. Der Prozess v_4 kann des Weiteren das Betriebsmittel S für sich sperren und anschließend seine Ausführung beenden. Beendet der Prozess seine Ausführung durch Feuern der Transition $s = c_i$, so feuert unmittelbar im Anschluss daran die Transition max und markiert

die Stelle $max(\pi)$ mit einem Tupel $< 2 >$. Erst jetzt kann der Prozess v_2 das Betriebsmittel S für sich sperren und seine Ausführung ebenfalls beenden.
Durch die genannten Modifikationen in der ursprünglichen Protokollspezifikation muss der Prozess v_4 lediglich auf die Beendigung des kritischen Gebietes T durch den Prozess v_3 und die Beendigung des kritischen Gebietes S durch den Prozess v_1 warten. Dies zeigt, dass eine verzögerte Vergabe frei gewordener Betriebsmittel – im Gegensatz zu einer unmittelbaren Vergabe frei gewordener Betriebsmittel – mit dem Theorem 6 aus Sha et al. ([37], S. 1178) korreliert. □

Für das aus geschilderten Änderungen resultierende Z-Pr/T-Netz G_7 wird im weiteren Verlauf die Gültigkeit der von Zöbel et al. formulierten Systeminvarianten [48],[49] mittels struktureller Analyse des Z-Pr/T-Netzes G_7 bewiesen.

5.2.5 Verifikation für das Priority-Inheritance-Protokoll mit verzögerter Betriebsmittelvergabe

In den vorangegangenen Abschnitten wurden die Funktionsweise des PIP sowie Modifikationen der ursprünglichen Protokollspezifikation zur Fehlerbehebung erläutert. Im weiteren Verlauf erfolgt eine Verifikation von Systemeigenschaften für die fehlerbereinigte Version des PIP mit verzögerter Betriebsmittelvergabe (siehe Kap. 5.2.4) basierend auf einer strukturellen Analyse des korrelierenden Z-Pr/T-Netzes. Anhand der nachfolgenden Protokollverifikation wird die Eignung von Z-Pr/T-Netzen nicht nur zur Modellierung und Simulation, sondern auch zur Verifikation komplexer Systeme nachgewiesen. Primäres Ziel des nachfolgenden Beweises ist es, die Eignung von Z-Pr/T-Netzen zur Verifikation komplexer Echtzeitsysteme zu belegen. Im Folgenden wird daher die von Zöbel et al. formulierte Systeminvariante [48],[49] mittels struktureller Analyse des Z-Pr/T-Netzes G_7 verifiziert. Die Systeminvariante nach Zöbel et al. [48],[49] ist wie folgt gegeben, wobei lediglich die Bezeichner angepasst wurden, um Konfusionen mit den im Rahmen dieser Arbeit bereits verwendeten Bezeichnern zu vermeiden.

$$
\begin{array}{ll}
(B_1) & stateScheduler = idle \\
(B_2) & \exists_1\ v : V \ \mid \ stateProcess(v) = running \\
(B_3) & \forall v_1, v_2 : V \ \mid \ v_1 \in V \ \wedge \ v_2 \in V \ \wedge \\
 & \quad stateProcess(v_1) = running \ \wedge \\
 & \quad (stateProcess(v_2) = ready \ \vee \\
 & \quad (stateProcess(v_2) = blocked \ \wedge \\
 & \quad blockedBy(v_2) \notin dom\ ownedBy)) \ \Rightarrow \\
 & \quad actualPriority(v_1) \ \geq \ actualPriority(v_2)
\end{array}
$$

Konstruktion der Systeminvarianten als eingebettetes Teilnetz
Im weiteren Verlauf erfolgt eine Verifikation obiger Systeminvarianten auf Basis der strukturellen Analyse des zugehörigen Z-Pr/T-Netzes G_7. Es muss daher zunächst ein Teilnetz, welches der zu beweisenden Systeminvarianten entspricht, konstruiert und in das zu analysierende Z-Pr/T-Netz integriert werden (siehe Kap.4.4, Technik IV).
Der Systeminvarianten nach Zöbel et al. [48],[49] liegt ein Modell zugrunde, welches eine explizite Modellierung eines *Schedulers* vorsieht. Aufgabe des Schedulers ist die Verwaltung von Prozessen. Dies umfasst unter anderem die Zuteilung von Prozessorzeit sowie die Vergabe von Betriebsmitteln. Der Scheduler kann dabei die Zustände *active* und *idle* einnehmen. Befindet der Scheduler sich im Zustand *active*, so arbeitet er alle notwendigen Schritte zur Findung seiner Scheduling-Entscheidungen ab. Währenddessen ist keiner der Prozesse aktiv. Hat der Scheduler alle Scheduling-Entscheidungen gefällt, so aktiviert er denjenigen Prozess, welcher zur Ausführung kommen soll, und wechselt selber in den Zustand *idle*. Im Gegensatz dazu modelliert das korrespondierende Z-Pr/T-Netz keinen expliziten Scheduler. Das Z-Pr/T-Netz G_7 wurde daher um zusätzliche Struktur ergänzt. Es beinhaltet die zusätzliche Stelle *idle*, welche den Zustand des Schedulers repräsentiert. Ist die Stelle *idle* markiert, so befindet der Scheduler sich im Zustand *idle*. Anderenfalls ist der Scheduler *active*. Initial ist die Stelle *idle* nicht markiert.
Zusätzlich wurden dem Z-Pr/T-Netz die Stellen *ready*, *running* und *blockedBy-Free* hinzugefügt. Die Markierung der Stelle *running* repräsentiert dabei denjenigen Prozess mit Zugriff auf den Prozessor. Die Markierung der Stelle *ready* repräsentiert die Menge aller rechenbereiten Prozesse. Die Markierung der Stelle $blockedByFree$ repräsentiert die Menge aller Prozesse, welche durch ein nicht gesperrtes Betriebsmittel blockiert werden.
Des Weiteren wurden dem Z-Pr/T-Netz G_7 die Transitionen t_{10} und t_{11} sowie die Stelle $\neg SystInvar$ hinzugefügt. Die Transition t_{10} feuert und markiert die Stelle $\neg SystInvar$, wenn die Stelle *running* mit einem niedrigeren Wert für die Variable π markiert ist als die Stelle *ready*. Die Transition t_{10} feuert also, wenn ein Prozess Zugriff auf den Prozessor hat, obwohl (mindestens) ein Prozess mit höherer Priorität existiert, welcher rechenbereit ist. Analog feuert die Transition t_{11} und markiert die Stelle $\neg SystInvar$, wenn die Stelle *running* mit einem niedrigeren Wert für die Variable π markiert ist als die Stelle $blockedByFree$. D.h. die Transition t_{11} feuert, wenn ein Prozess Zugriff auf den Prozessor hat, obwohl (mindestens) ein Prozess mit höherer Priorität existiert, welcher durch ein freies Betriebsmittel blockiert wird.
Das aus diesen Änderungen resultierende Z-Pr/T-Netz G_7' ist in Abbildung 5.6 dargestellt.

Induktive Verifikation der Systeminvarianten

Im weiteren Verlauf erfolgt eine induktive Verifikation der oben genannten Systeminvarianten nach Zöbel et al. [48],[49]. Gemäss der Methode der induktiven Verifikation (siehe Loeckx et al. [23]) muss bewiesen werden, dass die Systeminvariante initial gilt. Des Weiteren muss bewiesen werden, dass jede Folge von Zustandsübergängen die Systeminvariante bewahrt. Für das Z-Pr/T-Netz G_7 muss bewiesen werden, dass die Systeminvariante initial und immer dann erfüllt ist, wenn der Scheduler sich im Zustand *idle* befindet.
Da die Stelle *idle* initial nicht markiert ist, ist die Systeminvariante initial somit trivialer Weise erfüllt. Die Gültigkeit der Systeminvarianten muss daher für alle nachfolgenden Markierungen bewiesen werden, welche eine Markierung der Stelle *idle* inkludieren. Es muss daher gelten:

$$B_1 \;\Rightarrow\; (B_2 \wedge B_3) \qquad \equiv \qquad (B_1 \;\Rightarrow B_2) \;\wedge\; (B_1 \;\Rightarrow\; B_3)$$

Die Verifikation der Systeminvarianten kann somit in zwei Teilbeweise untergliedert werden. Zum Einen muss bewiesen werden, dass immer genau ein Prozess mit Zugriff auf den Prozessor existiert, wenn der Scheduler *idle* ist ($B_1 \Rightarrow B_2$).

Zum Anderen muss bewiesen werden, dass – unter der Voraussetzung, dass der Scheduler *idle* ist – immer demjenigen Prozess Zugriff auf den Prozessor gewährt wird, welcher von allen rechenbereiten Prozessen und durch freie Betriebsmittel blockierten Prozesse die höchste Priorität besitzt ($B_1 \Rightarrow B_3$).

Verifikation, Teil I

Der erste Teil des Beweises ($B_1 \Rightarrow B_2$) kann mit Hilfe von S-Invarianten geführt werden. Folgendes ist zu zeigen:

$$(stateScheduler = idle) \;\Rightarrow\; (\exists_1\; v : V \;\mid\; stateProcess(v) = running)$$

Es muss bewiesen werden, dass immer genau ein Prozess Zugriff auf den Prozessor hat, wenn der Scheduler *idle* ist. Ist in dem Z-Pr/T-Netz $G_7{}'$ die Stelle *idle* markiert, so muss die Summe aller Tupel auf den Stellen p_8 und p_{11} insgesamt eins betragen.
Die Stellen *cpu* und *idle* sind eine S-Invariante für das Z-Pr/T-Netz $G_7{}'$. Die mit den Stellen *cpu* und *idle* indizierten Einträge des S-Vektors S_1 sind somit $<>$, wobei die mit den verbleibenden Stellen indizierten Einträge $\emptyset$ sind. Die gewichtete Tupelanzahl auf den Stellen *cpu* und *idle* muss somit konstant sein. Da initial lediglich die Stelle *cpu* einfach markiert ist, muss somit für alle Folgemarkierungen die Summe aller Tupel auf den Stellen *cpu* und *idle* eins betragen. Es kann also entweder die Stelle *cpu* oder die Stelle *idle* markiert sein, nicht jedoch beide gleichzeitig.

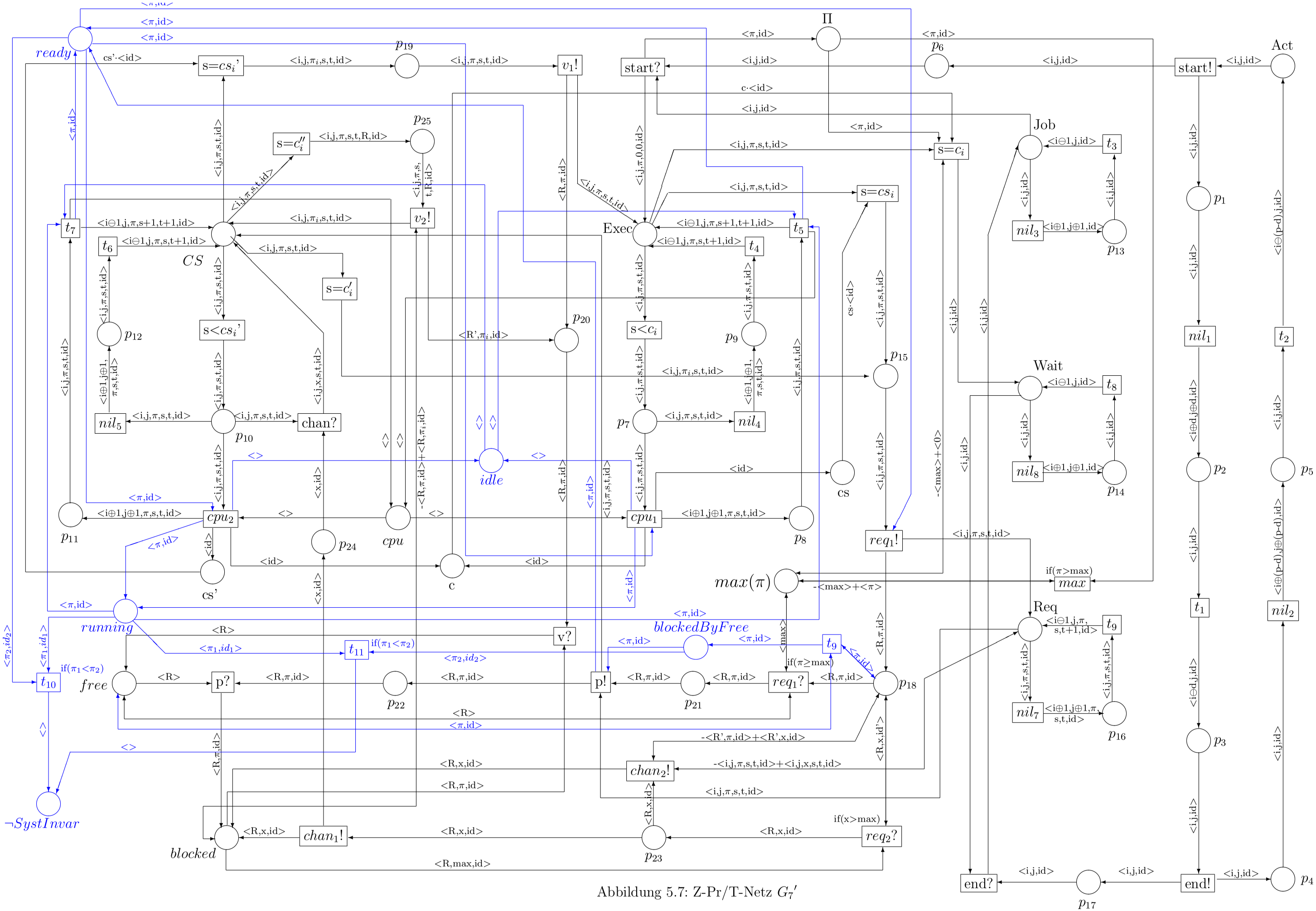


Abbildung 5.7: Z-Pr/T-Netz $G_7{}'$

Des Weiteren sind die Stellen *cpu*, p_8 und p_{11} eine S-Invariante für das Z-Pr/T-Netz $G_7{}'$. Der mit der Stelle *cpu* indizierte Eintrag des S-Vektors S_2 ist somit $<>$, die mit den Stellen p_8 und p_{11} indizierten Einträge sind $< i \oplus 1, j \oplus 1, \pi, s, t, id >$, wohingegen die mit den verbleibenden Stellen indizierten Einträge $\emptyset$ sind. Somit muss ebenso die gewichtete Tupelanzahl auf den Stellen *cpu*, p_8 und p_{11} konstant sein. Da initial lediglich die Stelle *cpu* markiert ist, muss somit die Summe aller Tupel auf den Stellen *cpu*, p_8 und p_{11} ebenfalls konstant eins betragen.

Die Systeminvariante muss nur gelten, wenn der Scheduler *idle* ist und die Stelle *idle* somit auch markiert ist. Aus einer Markierung der Stelle *idle* folgt jedoch auf Grund der S-Invarianten S_1 unmittelbar, dass die Stelle *cpu* nicht markiert ist. Ist die Stelle *cpu* jedoch nicht markiert, so folgt auf Grund der S-Invarianten S_2 wiederum, dass entweder die Stelle p_8 oder aber die Stelle p_{11} markiert ist. Aus einer Markierung der Stelle *idle* folgt somit unmittelbar, dass entweder die Stelle p_8 oder aber die Stelle p_{11} markiert ist, nicht jedoch beide gleichzeitig.

Somit kann mittels struktureller Analyse des Z-Pr/T-Netzes $G_7{}'$ der erste Teil der Verifikation der Systeminvarianten mit dem Beweis von $(B_1 \Rightarrow B_2)$ abgeschlossen werden. □

Verifikation, Teil II

Es gilt somit weiterhin zu beweisen, dass immer derjenige Prozess Zugriff auf den Prozessor hat, welcher von allen rechenbereiten und durch freie Betriebsmittel blockierten Prozesse die höchste Priorität besitzt. Es muss also gelten:

$$
\begin{aligned}
(stateScheduler = idle) \;\Rightarrow\; [\, \forall v_1, v_2 : V \;\mid\; & v_1 \in V \;\wedge\; v_2 \in V && \wedge \\
& stateProcess(v_1) = running && \wedge \\
& (stateProcess(v_2) = ready && \vee \\
& (stateProcess(v_2) = blocked && \wedge \\
& blockedBy(v_2) \notin dom\ ownedBy)) && \Rightarrow \\
& actualPriority(v_1) \geq actualPriority(v_2)\ ]
\end{aligned}
$$

$$
\begin{aligned}
\equiv \neg(stateScheduler = idle) \;\vee\; [\, \forall v_1, v_2 : V \;\mid\; & \neg(v_1 \in V) \;\vee\; \neg(v_2 \in V) && \vee \\
& \neg(stateProcess(v_1) = running) && \vee \\
& \neg(stateProcess(v_2) = ready && \vee \\
& (stateProcess(v_2) = blocked && \wedge \\
& blockedBy(v_2) \notin dom\ ownedBy)) && \vee \\
& actualPriority(v_1) \geq actualPriority(v_2)\ ]
\end{aligned}
$$

Es gilt die Gültigkeit der obigen Systeminvarianten zu verifizieren. Im Folgenden wird ein Beweis durch Widerspruch geführt, indem bewiesen wird, dass die Ne-

gation obiger Systeminvarianten unerfüllbar ist. Dabei lautet die Negation obiger Systeminvarianten:

$$
\begin{aligned}
(stateScheduler = idle) \ \wedge \ [\ \exists v_1, v_2 : V \ \mid \ & v_1 \in V \ \wedge \ v_2 \in V && \wedge \\
& stateProcess(v_1) = running && \wedge \\
& (stateProcess(v_2) = ready && \vee \\
& (stateProcess(v_2) = blocked && \wedge \\
& blockedBy(v_2) \notin dom\ ownedBy)) && \wedge \\
& actualPriority(v_1) < actualPriority(v_2)\ ]
\end{aligned}
$$

Es muss bewiesen werden, dass alle Zustände unerreichbar sind, in denen der Scheduler *idle* ist und der Prozess v_1 Zugriff auf den Prozessor hat, obwohl (mindestens) ein Prozess v_2 existiert, welcher eine höhere Priorität hat als der Prozess v_1, wobei der Prozess v_2 entweder rechenbereit oder durch ein freies Betriebsmittel blockiert ist.
Hat ein Prozess Zugriff auf den Prozessor, obwohl (mindestens) ein Prozess mit höherer Priorität existiert, welcher rechenbereit ist, dann feuert in dem Z-Pr/T-Netz ${G_7}'$ die Transition t_{10} und markiert die Stelle $\neg SystInvar$. Hat ein Prozess Zugriff auf den Prozessor, obwohl (mindestens) ein Prozess mit höherer Priorität existiert, welcher durch ein freies Betriebsmittel blockiert wird, so feuert in dem Z-Pr/T-Netz ${G_7}'$ die Transition t_{11} und markiert ebenfalls die Stelle $\neg SystInvar$. Für das Z-Pr/T-Netz ${G_7}'$ ist somit zu beweisen, dass die Stellen *idle* und $\neg SystInvar$ nie gleichzeitig markiert sein können. Der Nachweis dessen erfolgt im weiteren Verlauf strukturell mittels der Identifikation eines nicht-markierten Co-Traps. Ein nicht-markierter Co-Traps kann nicht wieder markiert werden. Somit kann aus der Zugehörigkeit der Stelle $\neg SystInvar$ zu einem nicht-markierten Co-Trap unmittelbar geschlossen werden, dass die Stelle $\neg SystInvar$ auch nicht wieder markiert werden kann.
Dabei kann die Stelle $\neg SystInvar$ nicht markiert werden, wenn weder die Transition t_{10} noch die Transition t_{11} feuert. Es wird daher ein *Backtrace* (siehe Kap. 5.1.2) über die Transitionen t_{10} und t_{11} erstellt. Der Backtrace ist in Abbildung 4.7 dargestellt. Demnach bilden die Stellen *cpu*, $\neg SystInvar$ und *running* für die mit ihnen assoziierten Variablenbelegungen den Co-Trap C_1.
Der Co-Trap C_1 gilt als *nicht markiert*, wenn die Stellen *cpu*, $\neg SystInvar$ und *running* nicht markiert sind. Der Co-Trap C_1 gilt jedoch auch dann als nicht markiert, wenn die Stellen *cpu* und $\neg SystInvar$ nicht markiert sind und die Stelle *running* darüber hinaus mit einem kleineren Wert für die Variable π markiert ist als die Stelle *ready* oder die Stelle *blockedByFree*. Anderenfalls gilt der Co-Trap als markiert. Es muss nun bewiesen werden, dass der Co-Trap C_1 nicht derart markiert werden kann.

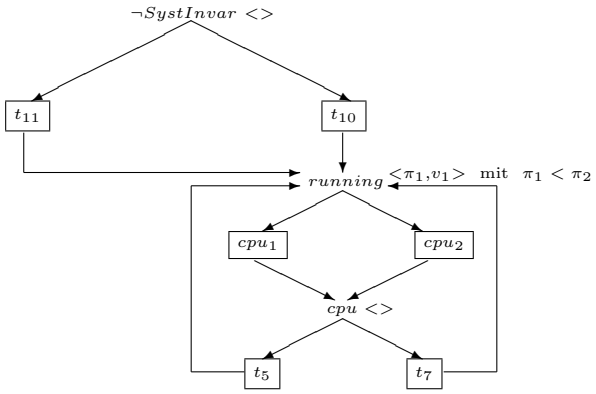


Abbildung 5.8: Backtrace für das Z-Pr/T-Netz $G_7{}'$

Ohne die Allgemeingültigkeit des nachfolgenden Beweises einzuschränken, können zunächst alle Anwendungsfälle ausgeschlossen werden, in denen ausschließlich ein einzelner Prozess aktiv ist, da in diesem Fall die Systeminvariante trivialer Weise erfüllt ist. Dementsprechend sind in diesem Fall weder die Stelle *ready* noch die Stelle *blockedByFree* markiert, so dass die Transitionen t_{10} und t_{11} somit auch nicht feuern und den Co-Trap C_1 markieren können. Es muss also ausschließlich für diejenigen Anwendungsfälle, in denen zwei oder mehr Prozesse aktiv sind, bewiesen werden, dass der Co-Trap C_1 nicht markiert werden kann.

Sei nun mit Prozess v_1 derjenige Prozess gegeben, welchem der Zugriff auf den Prozessor erteilt wurde. Dann hat in dem Z-Pr/T-Netz $G_7{}'$ entweder die Transition cpu_1 oder aber die Transition cpu_2 für das mit Prozess v_1 assoziierte Tupel gefeuert und die Stelle p_8 oder aber die Stelle p_{11} markiert. Darüber hinaus existieren weitere rechenbereite und/oder durch freie Betriebsmittel blockierte Prozesse, so dass die Stelle *ready* und/oder die Stelle *blockedByFree* entsprechend markiert ist. Demnach ist die Stelle *running* mit einem Tupel $< \pi_1, v_1 >$ markiert. Der Pfad von der Anfangsmarkierung bis zu einer Markierung der Stelle *running* ist jedoch ausgenommen von Nebenläufigkeit eindeutig. Da unter der zu betrachtenden Markierung die Stellen *cpu* und $\neg SystInvar$ nicht markiert sind, gilt der Co-Trap C_1 nur dann als markiert, wenn die Stelle *ready* und/oder die Stelle *blockedByFree* mit einem höheren Wert π_2 markiert ist. Ein Feuern der Transitionen cpu_1 und cpu_2 für einen Wert $\pi_1 < \pi_2$ wird jedoch durch die für Z-Pr/T-Netze definierte Schaltregel verhindert. Demnach kann die Stelle *running* auf Grund der in Z-Pr/T-Netzen geltenden Schaltregel nicht mit einem niedrigeren Wert für die Variable π markiert sein als die Stelle *ready* oder die Stelle *blockedByFree*. Der Co-Trap C_1 ist somit nicht markiert und kann auf Grund

der Eigenschaft von Co-Traps auch nicht wieder markiert werden. Daraus folgt wiederum, dass die Stellen *idle* und $\neg SystInvar$ nie gleichzeitig markiert sind. Mit dem Nachweis, dass die Stellen *idle* und $\neg SystInvar$ nicht gleichzeitig markiert sein können, konnte ein Widerspruch herbeigeführt und die Negation der Systeminvarianten widerlegt werden. Somit ist Gültigkeit der Systeminvarianten nach Zöbel et al. [49] auf Basis der strukturellen Analyse von Z-Pr/T-Netzen bewiesen. □

5.2.6 Fazit

In dem Kapitel 5.2 wurde das PIP mittels Z-Pr/T-Netzen modelliert und die Systeminvariante nach Zöbel et al. [48],[49] auf Basis der strukturellen Analyse des zugehörigen Z-Pr/T-Netzes verifiziert.
Es wurde daher das Z-Pr/T-Netz G_7 auf Basis der ACSR-VP Spezifikation von Choi et al. [2],[5] abgeleitet. Die ACSR-VP Spezifikation nach Choi et al. erlaubt jedoch keine Schachtelung kritischer Gebiete. Das Z-Pr/T-Netz G_7 wurde daher zunächst dahingehend erweitert, so dass eine gleichzeitige Sperre mehrerer Betriebsmittel durch einen Prozess möglich ist (siehe Kap. 5.2.2).
Die ursprüngliche Protokollspezifikation wurde natürlich-sprachlich, und nicht in einer formal-basierten Modellierungssprache definiert [37]. Eben diese fehlende formale Basis der ursprünglichen Protokolldefinition hat in Verbindung mit der nicht zu unterschätzenden Komplexität des Protokolls zu zahlreichen fehlerbehafteten Protokollspezifikationen sowie Implementierungen geführt. In Kapitel 5.2.3 und 5.2.4 wurden diese Fehler bzw. Fehlinterpretationen sowie die notwendigen Schritte zur Fehlerbehebung geschildert.
Die aus diesen Änderungen resultierende, fehlerbereinigte Version des PIP bildete die Grundlage für die Verifikation der Systeminvarianten nach Zöbel et al. [48],[49]. Zunächst wurde die zu beweisende Systeminvariante als Teilnetz in das zu analysierende Z-Pr/T-Netz eingebettet (siehe Technik IV, Kapitel 4.4). Anschließend erfolgte die Verifikation der Systeminvarianten mittels struktureller Analyse des daraus resultierenden Z-Pr/T-Netzes $G_7{}'$. Zum einen musste bewiesen werden, dass immer genau ein Prozess Zugriff auf den Prozessor hat, wenn der Scheduler sich im Zustand *idle* befindet. Zum Anderen musste bewiesen werden, dass – unter der Voraussetzung, dass der Scheduler sich im Zustand *idle* befindet – immer derjenige Prozess Zugriff auf den Prozessor hat, welcher von allen rechenbereiten und durch freie Betriebsmittel gesperrten Prozesse die höchste Priorität besitzt. Demnach musste gemäss der ersten Teilbedingung der Systeminvarianten bewiesen werden, dass im Falle einer Markierung der Stelle *idle* immer nur entweder die Stelle p_8 oder aber die Stelle p_{11} mit einem Tupel markiert ist. Die erste Teilbedingung der Systeminvarianten wurde mittels

S-Invarianten für das Z-Pr/T-Netz $G_7{}'$ verifiziert. Entsprechend der zweiten Teilbedingung der Systeminvarianten musste bewiesen werden, dass die Stellen *idle* und $\neg SystInvar$ nie gleichzeitig markiert sind. Die Verifikation dessen erfolgte auf Basis der Identifikation eines nicht-markierten Co-Traps.
Mittels der Verifikation der Systeminvarianten nach Zöbel et al. [48],[49] konnte somit der Nachweis für die Eignung von Z-Pr/T-Netzen zur Modellierung, Simulation und Verifikation zeitbewerteter Systeme und der praktische Nutzen derer im Anwendungsbereich der Echtzeitsysteme nachgewiesen werden.

6 Zusammenfassung und Ausblick

Im Rahmen dieser Arbeit wurde die Klasse der *zeitbewerteten Prädikat/Transitions-Netze (Z-Pr/T-Netze)* für die Modellierung, Simulation und Analyse komplexer Systeme aus dem Anwendgungsbereich der Echtzeitsysteme entwickelt.
Dabei geht die Klasse der Z-Pr/T-Netze mittels struktureller Modifikationen sowie einer modifizierten Schaltregel aus der Klasse der S-Pr/T-Netze hervor. Um eine Modellierung, Simulation und Analyse zeitbewerteter Systeme mittels Z-Pr/T-Netzen zu ermöglichen, wurde ein Zeitkonzept in Z-Pr/T-Netze integriert. Sei die *Systemzeit* die Zeit in dem zu modellierenden System und die *Modellzeit* die Darstellung der Systemzeit in dem Modell. In Z-Pr/T-Netzen wurde eine lokale Repräsentation der Modellzeit gewählt. Dies birgt den Vorteil, dass eine Konsistenzhaltung der Zeitwerte nicht erforderlich ist und die resultierenden Z-Pr/T-Netze somit von geringerer Komplexität sind. Den Stellen in Z-Pr/T-Netzen ist in Abhängigkeit ihrer Markierung eine *lokale Modellzeit* zugeordnet. Das Minimum aller lokalen Modellzeiten entspricht der *globalen Modellzeit*. Die Transitionen in Z-Pr/T-Netzen schalten zeitlos, wohingegen Zeit vergeht, während ein Tupel auf einer Stelle verweilt. Die Kantenanschriften in Z-Pr/T-Netzen sind durch Tripel der Form $< i, j, \pi >$ gegeben, wobei deren semantische Interpretation fest vorgegeben ist:

- Die Variable i repräsentiert die *Dauer einer Aktion*, wobei auf die Variable i die Modulus-Operationen $\oplus$ und $\ominus$ mit Modulus n_i angewendet werden. Die Werte der Variablen i werden somit auf einem durch den Modulus n_i beschränkten Wertebereich abgebildet.

- Die Variable j dient der Modellierung der *lokalen Modellzeit*. In Z-Pr/T-Netzen ist jeder Stelle eine lokale Modellzeit zugeordnet. Ist eine Stelle markiert, so ist ihre lokale Modellzeit das Minimum aller j-Werte ihrer Markierung. Ist eine Stelle nicht markiert, so ist die lokale Modellzeit für diese Stelle nicht definiert. Das Minimum aller lokalen Modellzeiten entspricht der globalen Systemzeit. Des Weiteren werden die Modulus-Operationen $\oplus$ und $\ominus$ mit Modulus n_j auf die Variable j angewendet, so dass die Werte der Variablen j auf einem durch den Modulus n_j beschränkten Wertebereich abgebildet werden.

- Die Variable π modelliert die *Priorität* eines Tupels.
- Die Prädikate der Kantenanschriften lassen sich zusätzlich, je nach spezifischen Anforderungen der Anwendung um weitere Variablen erweitern (siehe Kapitel 5.1 und 5.2).

Neben den strukturellen Modifikationen wurde die Schaltregel in Z-Pr/T-Netzen abgeändert. Dies begründet sich darin, dass das Eintreten von Zustandsübergängen in zeitbewerteten Systemen nicht nur von dem Systemzustand, sondern auch von der Systemzeit abhängt. In Z-Pr/T-Netzen finden die folgenden vier Prinzipien Anwendung:

- Es feuern immer zunächst diejenigen Transitionen, deren Eingangsstellen mit dem niedrigsten Wert für die Variable j markiert sind.
- Existieren mehrere Transitionen, deren Eingangstellen mit dem niedrigsten Wert für die Variable j markiert sind, so haben diejenigen Transitionen Vorrang, deren Eingangsstellen mit dem höchsten Wert für die Variable π markiert sind.
- Existieren mehrere Transitionen, deren Eingangsstellen mit dem niedrigsten Wert für die Variable j und dem höchsten Wert für die Variable π markiert sind, dann feuern zunächst diejenigen Transitionen, deren Eingangsstellen mit dem höchsten Wert für die Variable i markiert sind.
- Sind die Eingangsstellen mehrere Transitionen mit dem niedrigsten Wert für die Variable j und dem höchsten Wert für die Variablen π und i markiert, so haben immer diejenigen Transitionen Vorrang, welche keine Warteaktionen modellieren.

Die Werte der Variablen j spezifizieren die lokale Modellzeit einer Stelle und geben an, wann ein Tupel in Bezug auf die globale Modellzeit für Transitionen wieder verfügbar ist. Eine Transition kann nur dann aktiviert sein, wenn die lokalen Modellzeiten aller Eingangsstellen der Transition mit der globalen Modellzeit übereinstimmen. Ist dies für mehrere Transitionen gleichzeitig der Fall, so erhalten zunächst diejenigen Transitionen Vorrang, welche für diejenigen Tupel mit der höchsten Priorität aktiviert sind. Ist dies wiederum für mehrere Transition gleichzeitig der Fall, so erhalten diejenigen Transitionen Vorrang, welche die am weitesten fortgeschrittenen Aktionen modellieren. Warteaktionen feuern zuletzt.

Mittels Z-Pr/T-Netzen können zeitbewertete Systeme modelliert, simuliert und basierend auf strukturellen Analyseverfahren wie der Berechnung von S- und T-Invarianten sowie der Identifikation von Traps und Co-Traps verifiziert werden.

Die Anwendbarkeit von Z-Pr/T-Netzen zur Modellierung, Simulation aber vor allem zur Verifikation im Anwendungsbereich der Echtzeitsysteme wurde in Kapitel 5 nachgewiesen. Dabei konnten erstmals Traps und Co-Traps zur Verifikation in höheren Petrinetzen angewendet werden. Als Anwendungsbeispiele dienten dabei das EDF und das PIP. Die enorme, nicht zu unterschätzende Komplexität des EDFs und des PIPs belegen dabei die Anwendbarkeit von Z-Pr/T-Netzen zur Modellierung, Simulation und Verifikation zeitbehafteter Systeme von realem Komplexitätsumfang.
Es wurde demnach im Rahmen dieser Arbeit mit den Z-Pr/T-Netzen eine neue Petrinetzklasse entwickelt, deren Anwendbarkeit zur Modellierung, Simulation und Verifikation komplexer Systeme aus dem Anwendungsbereich der Echtzeitsysteme nachgewiesen wurde.

Um in dem Bereich der Anwendungsdomäne jedoch von praktischem Nutzen zu sein, sind eine Reihe von weiterführenden Arbeiten notwendig.

- Im Rahmen dieser Arbeit wurde die Eignung von Z-Pr/T-Netzen zur Verifikation komplexer, zeitbehafteter Systeme aus dem Anwendungsbereich der Echtzeitsysteme mittels struktureller Analyse der korrespondierenden Z-Pr/T-Netze nachgewiesen. Dabei wurden die Berechnung von S- und T-Invarianten sowie die Identifikation von Traps und Co-Traps als strukturelle Analyseverfahren betrachtet. Gerade in Hinblick auf die Diagnose sicherheitskritischer Echtzeitsysteme stellt die Dualisierung von Z-Pr/T-Netzen einen vielversprechenden Ansatz dar. Mit Diagnose wird dabei die Ermittlung von Ursachen für das Fehlverhalten eines Systems bezeichnet. Die Dualisierung von Z-Pr/T-Netzen zur Diagnose sicherheitskritischer Echtzeitsysteme ist somit ein Ansatzpunkt für weiterführende Arbeiten, welcher in der Anwendungsdomäne von großem praktischen Nutzen ist.

- Gerade der Vorgang der Modellierung komplexer bis sehr komplexer Systeme stellt eine potenzielle Fehlerquelle dar. Ein Aspekt weiterführender Arbeiten könnte daher sein, inwiefern sich Konzepte wie Hierarchisierung und Modularisierung als Hilfsmittel zur Handhabbarkeit von Komplexität in die Klasse der Z-Pr/T-Netze integrieren lassen.

- Des Weiteren ist die Entwicklung einer computergestützten Softwarelösung, welche eine Modellierung, Simulation und strukturelle Analyse von Z-Pr/T-Netzen umsetzt, für die Anwendbarkeit in der Praxis unabdingbar. Die Umsetzung eines Werkzeuges zur Modellierung, Simulation und Analyse von Z-Pr/T-Netzen offenbart darüber hinaus die Möglichkeit einer Eliminierung zu-

mindest syntaktischer Fehler mittels einer in diese Software integrierten Syntaxüberprüfung.

- Als letzter Punkt sei hier als Aspekt für weitere Arbeiten auf die automatische Codegenerierung aus den Modellen hingewiesen. Dies ist aus zweierlei Gründen in der Praxis des Anwendungsbereiches von Interesse. Zum Einen ist eine manuelle Generierung von Programmcode mit enormem Aufwand und somit auch mit Kosten verbunden, zum Anderen stellt die manuelle Programmierung auch eine zusätzliche, potenzielle Fehlerquelle dar. Es ist somit erstrebenswert auch diesen Punkt im Softwareentwicklungsprozess zu automatisieren.

Berechnung der T-Invarianten zu Beispiel 11

Die Inzidenzmatrix des Z-Pr/T-Netzes $G_4{''}$ ist wie folgt gegeben

	$t_{1.0}$	$t_{1.1}$	$t_{2.0}$	$t_{2.1}$	$t_{4.0}$	$t_{4.1}$	t_x	t_y
p_1	-<x>			<x>			<0>	
p_2		<x>			-<x>		<1>	-<0>-<1>
p_3			-<x>			<x>		
p_4	<x>	-<x>						
p_5	-<>	<>						
p_6			<x>	-<x>				
p_7			-<>	<>				
p_8					<x>	-<x>		
p_9					-<>	<>		
C	-<x> +<x⊕1>		-<x> +<x⊕1>		-<x> +<x⊕1>		<0> +<1>	-<0> -<1>

Sei der folgende T-Vektor gegeben

$$U_1 = \begin{bmatrix} 3\cdot <0> + <1> \\ 3\cdot <0> + <1> \\ <1> \\ <1> \\ <1> \\ <1> \\ 3\cdot <> \\ 3\cdot <> \end{bmatrix}$$

Ist U_1 eine T-Invariante, so muss gelten: $[G(p,t)] \circ U_1 = 0$.

$$\begin{aligned} G(p_1,-)\circ U_1 &= -<x>\circ(3<0>+<1>) + <x>\circ<1> \\ &\quad +<0>\circ(3\cdot<>) \\ &= -3<0> - <1> + <1> + 3<0> \\ &= 0 \end{aligned}$$

$$
\begin{aligned}
G(p_2,-)\circ U_1 &= <x>\circ(3<0>+<1>) - <x>\circ<1> + <1>\circ(3\cdot<>) \\
&\quad -(<0>+<1>)\ \circ\ (3\cdot<>) \\
&= 3<0>+<1>-<1>+3<1>-3<0>-3<1> \\
&= 0
\end{aligned}
$$

$$
\begin{aligned}
G(p_3,-)\circ U_1 &= -<x>\circ<1>+<x>\circ<1> \\
&= -<1>+<1> \\
&= 0
\end{aligned}
$$

$$
\begin{aligned}
G(p_4,-)\circ U_1 &= <x>\circ(3<0>+<1>) \ - \ <x>\circ(3<0>+<1>) \\
&= 0
\end{aligned}
$$

$$
\begin{aligned}
G(p_5,-)\circ U_1 &= -<>\circ(3<0>+<1>) \ + \ <>\circ(3<0+<1>) \\
&= 0
\end{aligned}
$$

$$
\begin{aligned}
G(p_6,-)\circ U_1 &= <x>\circ<1>-<x>\circ<1> \\
&= 0
\end{aligned}
$$

$$
\begin{aligned}
G(p_7,-)\circ U_1 &= -<>\circ<1>+<>\circ<1> \\
&= 0
\end{aligned}
$$

$$
\begin{aligned}
G(p_8,-)\circ U_1 &= <x>\circ<1>-<x>\circ<1> \\
&= 0
\end{aligned}
$$

$$
\begin{aligned}
G(p_9,-)\circ U_1 &= -<>\circ<1>+<>\circ<1> \\
&= 0
\end{aligned}
$$

$$
\begin{aligned}
G(C,-)\circ U_1 &= \quad (-<x>+<x\oplus 1>)\circ(3<0>+<1>) \\
&\quad +(-<x>+<x\oplus 1>)\circ<1> \\
&\quad +(-<x>+<x\oplus 1>)\circ<1> \\
&\quad +(<0>+<1>)\circ(3\cdot<>) \\
&\quad -(<0>+<1>)\circ(3\cdot<>) \\
&= -3<0> \ - \ <1> \ + \ 3<1> \ + \ <0> \ - \ <1> \\
&\quad + \ <0> \ - \ <1> \ + \ <0> \\
&= 0
\end{aligned}
$$

Sei ebenso der nachfolgende T-Vektor U_2 gegeben

$$U_2 = \begin{bmatrix} <0>+<1> \\ <0>+<1> \\ <0>+<1> \\ <0>+<1> \\ <0>+<1> \\ <0>+<1> \\ 0 \cdot <> \\ 0 \cdot <> \end{bmatrix}$$

Ist U_2 eine T-Invariante, so muss ebenfalls gelten: $[G(p,t)] \circ U_2 = 0$.

$$\begin{aligned} G(p_1,-) \circ U_2 &= -<x> \circ (<0>+<1>) + <x> \circ (<0>+<1>) \\ &\quad <0> \circ (0 \cdot <>) \\ &= 0 \end{aligned}$$

$$\begin{aligned} G(p_2,-) \circ U_2 &= <x> \circ (<0>+<1>) - <x> \circ (<0>+<1>) \\ &\quad +<1> \circ (0 \cdot <>) - (<0>+<1>) \circ (0 \cdot <>) \\ &= 0 \end{aligned}$$

$$\begin{aligned} G(p_3,-) \circ U_2 &= -<x> \circ (<0>+<1>) + <x> \circ (<0>+<1>) \\ &= 0 \end{aligned}$$

$$\begin{aligned} G(p_4,-) \circ U_2 &= <x> \circ (<0>+<1>) - <x> \circ (<0>+<1>) \\ &= 0 \end{aligned}$$

$$\begin{aligned} G(p_5,-) \circ U_2 &= -<> \circ (<0>+<1>) + <> \circ (<0+<1>) \\ &= 0 \end{aligned}$$

$$\begin{aligned} G(p_6,-) \circ U_2 &= <x> \circ (<0>+<1>) - <x> \circ (<0>+<1>) \\ &= 0 \end{aligned}$$

$$\begin{aligned} G(p_7,-) \circ U_2 &= -<> \circ (<0>+<1>) + <> \circ (<0>+<1>) \\ &= 0 \end{aligned}$$

$$\begin{aligned} G(p_8,-) \circ U_2 &= <x> \circ (<0>+<1>) - <x> \circ (<0>+<1>) \\ &= 0 \end{aligned}$$

$$
\begin{aligned}
G(p_9,-)\circ U_2 &= -<>\circ(<0>+<1>)+<>\circ(<0>+<1>)\\
&= 0
\end{aligned}
$$

$$
\begin{aligned}
G(C,-)\circ U_2 &= \quad (-<x>+<x\oplus 1>)\circ(<0>+<1>)\\
&\quad +(-<x>+<x\oplus 1>)\circ(<0>+<1>)\\
&\quad +(-<x>+<x\oplus 1>)\circ(<0>+<1>)\\
&\quad +(<0>+<1>)\circ(0\cdot<>)\\
&\quad -(<0>+<1>)\circ(0\cdot<>)\\
&= -<0>-<1>+<1>+<0>-<0>-<1>+<1>\\
&\quad +<0>-<0>-<1>+<1>+<0>\\
&= 0
\end{aligned}
$$

Transformation von ACSR-VP in Z-Pr/T-Netze

In Kapitel 5 wurden das EDF und das PIP mittels Z-Pr/T-Netzen modelliert. Dabei beruhen die Modelle in beiden Fällen auf den Protokollspezifikationen in der *Algebra of Communicating Shared Resources with Value Passing (ACSR-VP)* nach [2],[5].
Bei der *Algebra of Communicating Shared Resources (ACSR)* handelt es sich um eine zeitbasierte Prozessalgebra, welche für die formale Spezifikation und Manipulation verteilter Echtzeitsysteme und deren Betriebsmittel entwickelt wurde. Deren Ursprünge gehen dabei auf Lee et al. [18] zurück. Die Weiterentwicklung der ACSR zu der ACSR-VP beruht auf Choi et al. [2],[5] und integriert zusätzliche Mechanismen zur dynamischen Vergabe von Prioritäten sowie der Wertepropagation in die bereits vorhandenen Modellierungskonzepte der ACSR. Im Folgenden wird beschrieben, wie sich die ACSR-VP Spezifikationen des EDFs und des PIPs nach Choit et al. [2],[5] in Z-Pr/T-Netze abbilden lassen. Dabei werden im weiteren Verlauf folgende Bezeichner verwendet:

- P, P_1, P_2 und Q repräsentieren Prozesse,
- e repräsentiert ein Event,
- A repräsentiert eine Aktion,
- x_1,...,x_n sind Variablen,
- I spezifiziert eine Menge von Betriebsmitteln,
- F spezifiziert eine Menge von Eventlabels und
- C ist der Bezeichner eines Prozesses.

Für die im Folgenden vorgestellten Operatoren + , : , . und → gilt:

bindet stärker als	
:	+
.	+
→	+

$P := P_1 || P_2$

Der Prozess P spezifiziert die parallele Ausführung der Prozesse P_1 und P_2.

$P := [Q]_I$

Der Prozess P führt Prozess Q aus, wobei dieser die in I spezifizierten Betriebsmittel exklusiv nutzt.

Bsp.: $EDFSys := [\ T_1 \ || \ T_2 \ || \ ... \ || \ T_n \ ]_{\{cpu\}}$

$EDFSys$ führt die Prozesse $T_1, T_2, ..., T_n$ parallel aus, wobei diese auf das exklusiv nutzbare Betriebsmittel *cpu* zugreifen. Das Betriebsmittel *cpu* ist außerhalb von $EDFSys$ nicht zugreifbar. In dem Z-Pr/T-Netz G_6 (Abb. 5.2) wird die exklusive Nutzung des Betriebsmittels *cpu* durch die gleichnamige Stelle *cpu* realisiert.

$P := Q \setminus F$

Der Prozess P führt den Prozess Q aus, jedoch sind die in F in Form von Bezeichnern spezifizierten Events für die Umgebung nicht sichtbar.

Bsp.: $T_i := (\ Job_i \ || \ Activator_i \ ) \setminus \{start, end\}$

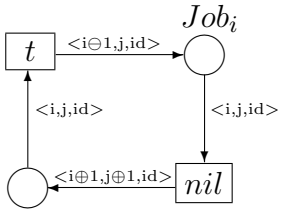
Die Prozesse Job_i und $Activator_i$ werden parallel ausgeführt und interagieren über Events miteinander. Diese Events sind in Form der Eventbezeichner *start* und *end* gegeben. Das Event *start* setzt sich aus einem Output-Event *start*! und einem Input-Event *start*? zusammen. In dem zugehörigen Z-Pr/T-Netz werden die Input- und Output-Events *start*! und *start*? jeweils durch gleichnamige Transitionen repräsentiert, welche über eine zusätzliche Stelle miteinander verknüpft sind (analog für *end*). Diese Events sind außerhalb der Prozesse Job_i, $Activator_i$ und der von ihnen aufgerufenen Prozesse $Exec_i$ und $Wait_i$ nicht zugreifbar (siehe Z-Pr/T-Netz G_6 in Abb. 5.2).

Die Verknüpfung von Transitionen, welche Input- und Output-Events darstellen, wird an späterer Stelle erläutert.

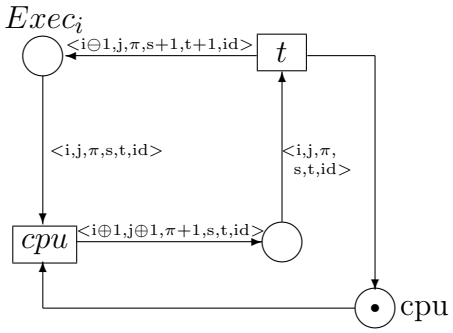
$P := A : Q$

Der Prozess P führt die Aktion A aus und fährt dann mit Prozess Q fort. Aktionen in ACSR-VP beschreiben dabei zeitkonsumierende Zustandübergänge.
Ist eine Aktion als $A = \{ (r_1, \pi_1), (r_2, \pi_2), ..., (r_n, \pi_n) \}$ definiert, so beschreibt A eine Aktion, welche auf die Betriebsmittel r_i mit Priorität π_i zugreift, mit $0 < i \leq n$. Alternativ kann eine Aktion mit $A = \emptyset$ definiert sein, welche die Warteaktion (engl. *idle action*) modelliert.
Aktionen sind in ACSR-VP über eine Dauer von einer Zeiteinheit definiert. Anderenfalls wird A^d als Abkürzung für Aktionen von Dauer d verwendet, mit $d > 1$. Aktionen mit Dauer $d = 0$ werden als Events angesehen.

Bsp.: $Job_i := \emptyset : Job_i$



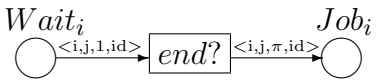
Bsp.: $Exec_i(s,t) := \{(cpu, \pi)\} : Exec_i(s+1, t+1)$



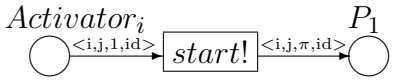
$P := e.Q$

Der Prozess P führt das Event e aus und fährt dann mit Prozess Q fort.
Mit Events werden in ACSR-VP zeitlose Zustandsübergänge bezeichnet. Events sind in ACSR-VP als ein Tupel (l, π) definiert, wobei l der Bezeichner des Events und π die Priorität ist, mit der das Event ausgeführt wird. Des Weiteren bezeichne $l = e!$ ein Ausgabeevent und $l = e?$ ein Eingabeevent.

Bsp.: $Wait_i := (end?, 1).Job_i$

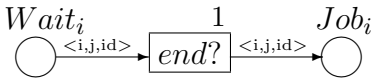


Bsp.: $Activator_i := (start!, 1).P_1$

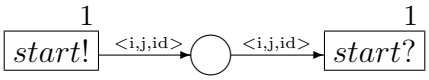


Es sei hier angemerkt, dass den Events und Aktionen in ACSR-VP statische Prioritäten zugeordnet sind, mit denen diese ausgeführt werden. Transitionen mit statischen Prioritäten werden im Rahmen dieser Arbeit daher oftmals vereinfacht modelliert. Derartige Transitionen werden zur Vereinfachung mit ihren statischen Prioritäten annotiert, wohingegen die Ein- und Ausgangskanten ohne die Variable π modelliert werden.

Bsp.: $Wait_i := (end?, 1).Job_i$



Zusätzlich müssen gleichnamige Ein- und Ausgabeevents über eine zusätzliche Stelle wie folgt miteinander verbunden werden:



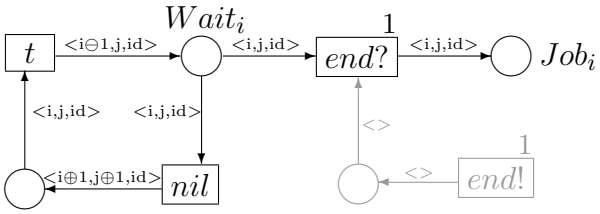
Dementsprechend kann ein Eingabeevent nur durch das zugehörige Ausgabeevent angestoßen werden. Die Transition *start*? kann beispielsweise nur dann feuern, wenn die Transition *start*! bereits gefeuert hat. Ein- und Ausgabeevents sind jedoch aneinander gekoppelt. D.h. das Eintreten eines Ausgabeevents bedingt das unmittelbare Eintreten des zugehörigen Ausgabeevents. Hat beispielsweise die Transition *start*! gefeuert, so muss die Transition *start*? im unmittelbaren Anschluss feuern. Aus diesem Grund muss denjenigen Transitionen, welche Events darstellen, bereits in der Modellierungsphase eine höhere Priorität zugewiesen werden als denjenigen Transitionen, welche Aktionen modellieren. Dabei werden Aktionen mit Dauer $\delta = 0$ als Events angesehen.

Sind nun in Bezug auf die globale Modellzeit mehrere Transitionen gleichzeitig im herkömmlichen Sinne aktiviert, so erfolgt zunächst das Feuern derjenigen Transitionen, welche Events modellieren. Erst im Anschluss daran werden durch Feuern derjenigen Transitionen, welche Aktionen modellieren, die lokalen Modellzeiten und somit auch implizit die globale Modellzeit erhöht. Es werden daher in Bezug auf die globale Modellzeit zunächst alle diejenigen Transitionen gefeuert, welche Events modellieren. Dies bewirkt eine Koppelung von Ein- und Augabeevents. Als Beispiel sei hier auf die Modellierung des EDF mittels Z-Pr/T-Netz G_6 (Abb. 5.2, Kap. 5.1) oder die Modellierung des PIP mittels Z-Pr/T-Netz G_7 (Abb. 5.5, Kap. 5.2) verwiesen.

$P := P_1 + P_2$

Der Prozess P führt Prozess P_1 oder Prozess P_2 aus, wobei die Wahl nichtdeterministisch erfolgt.

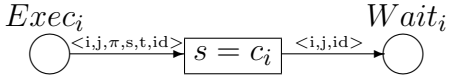
Bsp.: $Wait_i := \emptyset : Wait_i + (end?, 1).Job_i$



$P := be \rightarrow Q$

Der Prozess P führt den Prozess Q aus, falls der boolsche Ausdruck *be* erfüllt ist. Dies wird in dem korrelierenden Z-Pr/T-Netz durch einen Guard modelliert.

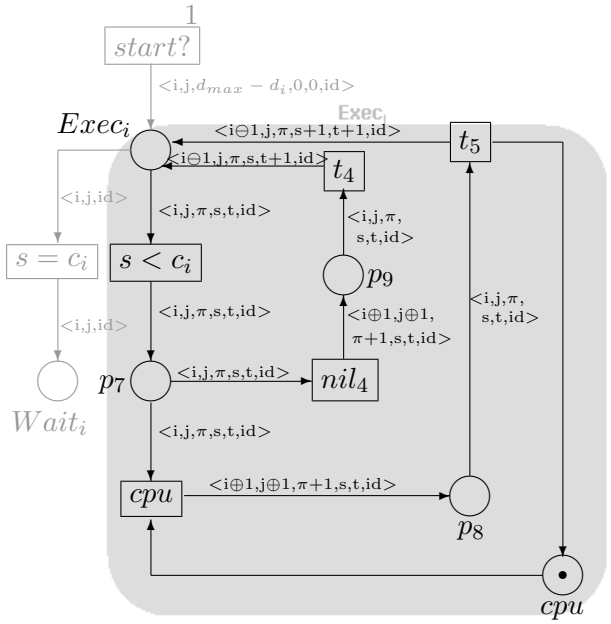
Bsp.: $Exec_i(s,t) := (s = c_i) \rightarrow Wait_i$



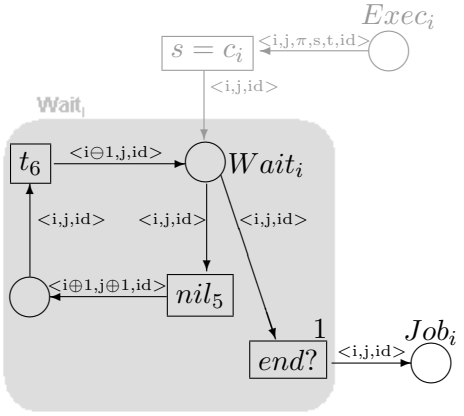
$C(x_1, ..., x_n) := P$

Mit $C(x_1, ..., x_n)$ wird ein Prozess definiert. Eine derartige Prozessdefinition macht ggf. eine Erweiterung der Kantenanschriften notwendig. Seien dazu die zwei nachfolgenden Beispiele gegeben.

Bsp.: $Exec_i(s,t) := \ (s < c_i) \rightarrow (\{(cpu, \pi)\} : Exec_i(s+1, t+1) + \emptyset : Exec_i(s, t+1))$
$+ (s = c_i) \rightarrow Wait_i$



Bsp.: $Wait_i := \emptyset : Wait_i + (end?, 1).Job_i$



Literaturverzeichnis

[1] Baumgarten, B. *Petri-Netze – Grundlagen und Anwendungen.* Spektrum Akademischer Verlag, 1996.

[2] Ben-Abdallah, H. ; Choi, J.-Y. ; Clarke, D. ; Kim, Y.-S. ; Lee, I. ; Xie, H.-L. *A Process Algebraic Approach to the Schedulability Analysis of Real-Time Systems.* Real-Time Systems, 15(3), 1998.

[3] Berthomieu, B ; Diaz, M. *Modeling and Verification of Time Dependent Systems Using Time Petri Nets.* IEEE Transactions on Software Engineering, 17(3):259–273, 1991.

[4] Cheng, A.M.K. *Real Time Systems: Scheduling, Analysis, and Verification.* John Wiley & Sons, 2002.

[5] Choi, J.-Y. ; Lee, I. ; Xie, H.-L. *The Specification and Schedulability Analysis of Real-Time Systems using ACSR.* In IEEE Real-Time Systems Symposium, pages 266–275, 1995.

[6] *DIN 443000 (Definition eines Echtzeitsystems).*

[7] Genrich, H.J. ; Lautenbach, K. *The analysis of distributed systems by means of predicate-/transition nets.* LNCS, 70:123–146, 1979.

[8] Genrich, H.J. ; Lautenbach, K. *System Modelling with High-Level Petri Nets.* Theoretical Computer Science, 13(1), 1981.

[9] Genrich, H.J. ; Lautenbach, K. *S-invariance in Predicate/Transition Nets.* Application and Theory of Petri Nets, Informatik Fachberichte Nr. 66, Bonn, 1983.

[10] Hanisch, H.-M. *Analysis of Place/Transition Nets with Timed-Arcs and its Aplication to Batch Process Control.* Application and Theory of Petri Nets, Volume 691 of Lecture Notes in Computer Science, pages 282–299, 1993.

[11] Hanisch, H.M. ; Lautenbach, K. ; Simon, C. ; Thieme, J. *Zeitstempelnetze in technischen Anwendungen*, journal = Fachberichte Informatik 2-98, Universität Koblenz-Landau, year = 1998,.

[12] Jensen, K. *Coloured Petri Nets and the Invariant-Method.* Theoretical Computer Science, 14:317–336, 1981.

[13] Jensen, K. *How to Find Invariants for Coloured Petri Nets.* MFCS, pages 327–338, 1981.

[14] Juan, E.Y.T. ; Tsai, J.J.P. ; Murata, T. ; Zhou, Y. *Reduction Methods for Real-Time Systems Using Delay Time Petri Nets.* IEEE Transactions on Software Engineering, 27(5):422–448, 2001.

[15] Kindler, V. *Lebendigkeit free-choice-ähnlicher Prädikat/Transitionsnetze*. Diplomarbeit, Universität Bonn, 1985.

[16] König, R. ; Quäck, L. *Petri-Netze in der Steuerungs- und Digitaltechnik*. Oldenbourg Verlag, 1988.

[17] Lautenbach, K. *Simple Marked-graph-like Predicate Transition Nets*. Arbeitspapiere der GMD, July 1983.

[18] Lee, I. ; Brémond-Grégoire P. ; Gerber, R. *A Process Algebraic Approach to the Specification and Analysis of Resource-Bound Real-Time Systems*. In IEEE Proceedings, pages 158–171, 1994.

[19] Leveson, N.G. ; Stolzy, J.L. *Safety Analysis Using Petri Nets*. IEEE Transactions on Software Engineering, 13(3):386–397, 1987.

[20] Lim, S.-M. ; Choi,J.-Y. *Specification and Verification of Real-Time Systems Using ACSR-VP*. In RTCSA, pages 135–142, 1997.

[21] Lin, C. ; Shan, Z. ; Liu, T. ; Qu, Y. ; Ren, F. *Modeling and Inference of Extended Interval Temporal Logic for Nondeterministic Intervals*. IEEE Transactions on Systems, Man and Cybernetics, Part A, 35(5):682 – 696, 2005.

[22] Liu, C.L. ; Layland, J.W. *Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment*. Journal of the ACM, 20(1):46–61, 1973.

[23] Loeckx, J. ; Sieber, K. ; Stansifer, R.D. *The Foundations of Program Verification*. John Wiley & Sons, Inc., New York, NY, USA, 1984.

[24] Merlin, P.M. *A Study of the Recoverability of Computing Systems*. PhD thesis, Department of Information and Computer Science, 1974.

[25] Moylan, P.J. ; Betz, R. ; Middleton, R. *The Priority Disinheritance Problem*. Technical Report EE9345, University of Newcastle, 1993.

[26] Penczek, W. ; Polrola, A. *Advances in Verification of Time Petri Nets and Timed Automata - A Temporal Logic Approach*. Springer, 2006.

[27] Petri, C.A. *Kommunikation mit Automaten*. PhD thesis, Institut für Instrumentelle Mathematik, 1962.

[28] Philippi, S. *Synthesis of Petri-Nets and Object-Oriented Concepts*. PhD thesis, Institut für Softwaretechnik, 1999.

[29] Polock, D. ; Zöbel, D. *Conformance Testing of Priority Inheritance Protocols*. In RTCSA, pages 404–408, 2000.

[30] Popova-Zeugmann, L. *On Time Invariance in Time Petri Nets*. Informatik-Berichte der HUB Nr.39, 1994.

[31] Popova-Zeugmann, L. ; Werner, M. ; Richling, J. *Using State Equation to Prove Non-Reachability in Timed Petrinets*. Fundamenta Informaticae, 55(2):187–202, 2003.

[32] Popova-Zeugmann,L. ; Werner, M. ; Richling, J. *A State Equation for Timed Petrinets*. Workshop Concurrency, Specification & Programming, Berlin, Tagungsband, 2002.

[33] Ramchandani, C. *Analysis of Asynchronous Concurrent Systems by Timed Petri Nets.* PhD thesis, Massachusetts Institute of Technology, 1974.

[34] Reisig, W. *Petrinetze: Eine Einführung.* Springer, 1986.

[35] Ripoll, I. ; Crespo, A. ; Mok, A.K. *Improvement in Feasibility Testing for Real-Time Tasks.* Real-Time Systems, 11:19.

[36] Schneeweiss, W.G. *Petri-Netze-Bilder-Buch.* LiLoLe-Verlag, 2002.

[37] Sha, L. ; Rajkumar, R. ; Lehoczky, J.P. *Priority Inheritance Protocols: An Approach to Real-Time Synchronization.* IEEE Transactions on Computers, 39(9):1175–1185, 1990.

[38] Simon, C. *A Logic of Actions and Its Application to the Development of Programmable Controllers.* PhD thesis, Universität Koblenz-Landau, 2001.

[39] Spuri, M. *Earliest Deadline Scheduling in Real-Time Systems.* PhD thesis, Scuola Superiore S. Anna, Pisa, 1995.

[40] Srba, J. *Timed-Arc Petri Nets vs. Networks of Timed Automata.* In ICATPN, pages 385–402, 2005.

[41] Starke, P.H. *Petri-Netze: Grundlagen, Anwendungen, Theorie.* VEB Deutscher Verlag der Wissenschaften, Berlin, 1980.

[42] Tsai, J.J.P. ; Yang, S.J. ; Chang, Y.-H. *Timing Constraint Petri Nets and Their Application to Schedulability Analysis of Real-Time System Specifications.* IEEE Transactions on Software Engineering, 21(1):32–49, 1995.

[43] Walter, B. *Timed Petri-Nets for Modelling and Analyzing Protocols with Real-Time Characteristics.* In Protocol Specification, Testing, and Verification, pages 149–159, 1983.

[44] Werner, M. ; Popova-Zeugmann, L. ; Richling, J. *A Method to Prove Non-Reachability in Priority Duration Petri Nets.* Fundamenta Informaticae, 61(3-4):351–368, 2004.

[45] Yao, Y. *A Petri Net Model for Temporal Knowledge Representation and Reasoning.* IEEE Transactions on Systems, Man and Cybernetics, 24(9):1374 – 1382, 1994.

[46] Zaidi, A.K. *On Temporal Logic Programming Using Petri Nets.* IEEE Transactions on Systems, Man, and Cybernetics, Part A, 29(3):245–254, 1999.

[47] Zöbel, D. *Echtzeitsysteme - Grundlagen der Planung.* Springer Verlag, Berlin, 2008.

[48] Zöbel, D. ; Albrecht, W. *Echtzeitsysteme - Grundlagen und Techniken.* Lehrbuch, International Thomson Publishing Company, Bonn, Albany, 1980.

[49] Zöbel, D. ; Polock, D. *Priority Inheritance Revisited.* In Joel Goossens, editor, 12th International Conference on Real-Time Systems (RTS'2004), pages 190–203, Paris Expo, 3 2004. Birp, 11, rue du Perche, 75003 Paris.

Index

Tabellarischer Lebenslauf

Katharina Hupf

Persönliche Daten

Name:	Katharina Hupf
Adresse:	Paulstr. 1 56070 Koblenz
Geburtsdatum:	29.09.1979
Geburtsort:	Dortmund

Schulausbildung

1986 - 1990	Kirchhörder Grundschule, Dortmund
1990 - 1999	Stadtgymnasium Dortmund
1999	Abschluss Abitur

Universitärer Werdegang

10/1999 - 10/2004	Studium der Computervisualistik an der Universität Koblenz-Landau
10/2004	Abschluss als Diplom-Informatikerin
11/2004 - Dato	Promotion am Institut für Softwaretechnik der Universität Koblenz-Landau.
12/2004 - 05/2005	Wissenschaftliche Hilfskraft an der Universität Koblenz-Landau
06/2005 - 05/2007	Promotionsstipendiatin nach dem Landesgraduiertenförderungsgesetz