Konstruktion korrekter Codegeneratoren für Speicherprogrammierbare Steuerungen

Dissertation

zur Erlangung das akademischen Grades
doktor rerum naturalium (Dr. rer. nat.)

vorgelegt der

Naturwissenschaftlichen Fakultät III
(Institut für Informatik)
der Martin–Luther–Universität Halle–Wittenberg

von Herrn
Dipl.-Inf. Dipl.-Wirt.-Inf. Dirk Pollmächer
geboren am 11.08.1977 in Markranstädt

Gutachter:

1. Prof. Dr. Wolf Zimmermann
2. Prof. Dr. Wolfgang Reisig

Datum der Verteidigung: 24. Juni 2008

Bad Lauchstädt, 15. Februar 2008

urn:nbn:de:gbv:3-000014185
[http://nbn-resolving.de/urn/resolver.pl?urn=nbn%3Ade%3Agbv%3A3-000014185]

Danksagung

Mein Dank gilt Prof. Dr. Wolf Zimmermann für die gewährte Unterstützung und den Rückhalt, auf den ich mich stets verlassen konnte. Unsere fruchtbaren Diskussionen öffneten mir oft die Augen.

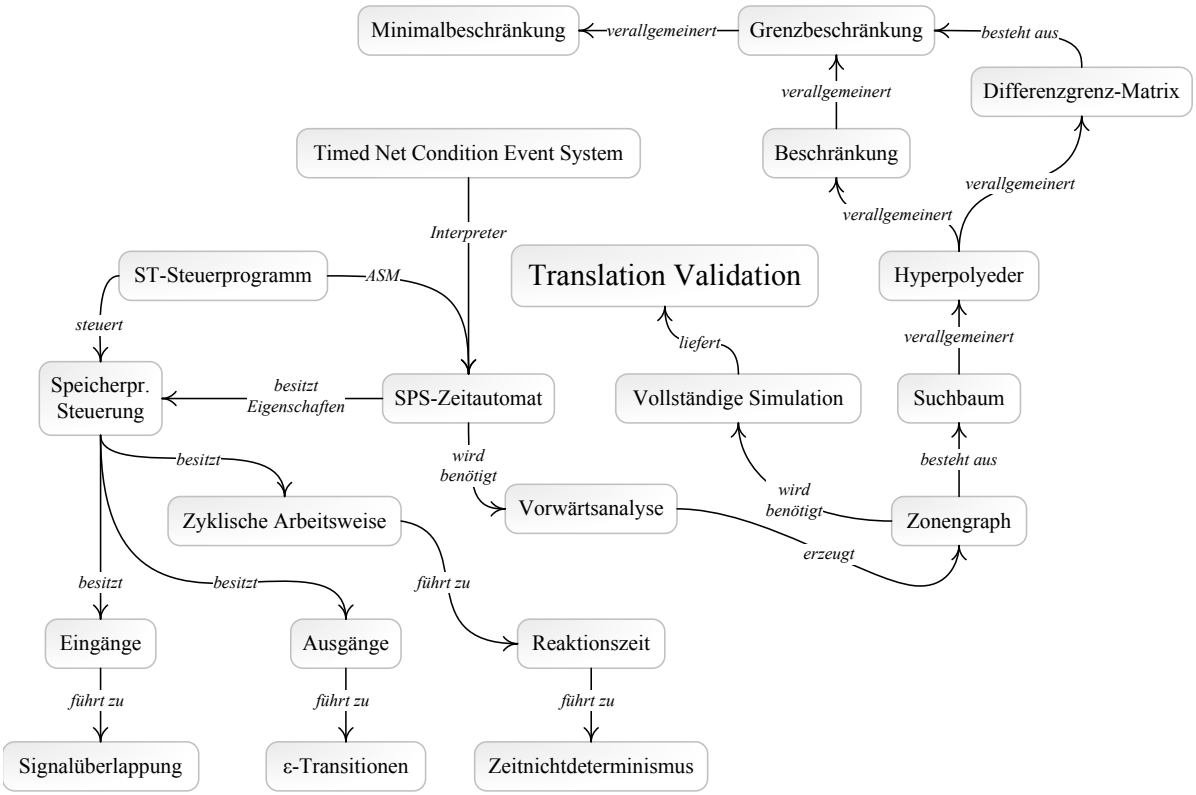
Danken möchte ich auch Prof. Dr. Wolfgang Reisig für seine Bereitschaft, das Zweitgutachten zu schreiben. Sein wachsames Auge und seine zahlreichen Hinweise trugen wesentlich zum Gelingen der Arbeit bei.

Meinen besonderen Dank möchte ich der Arbeitsgruppe für Software-Engineering und Programmiersprachen aussprechen, insbesondere Frau Dr. Roswitha Picht, Herrn Dr. Werner Gabrisch und Herrn Michael Schaarschmidt. Ohne ihre Unsterstützung, den Zusammenhalt und die manchmal nötige Portion Humor wäre diese Arbeit nie entstanden.

Ganz herzlich danken möchte ich meiner Familie und vor allem meiner Frau Daniela, die auch in schweren Zeiten eine stete Stütze für mich war. Daniela, Du zogst mich immer dann an Land, wenn ich am Leben zu ertrinken drohte.

Inhaltsverzeichnis

Übersicht wichtiger Begriffe



Glossar

Begriff	Beschreibung
Asynchronität	Eigenschaft zweier Uhren, auf unterschiedlichen Takten zu basieren und eine *Drift* zu besitzen
Ausfallhäufigkeit	„Anzahl der Fälle pro Zeiteinheit, in denen eine Komponente versagt oder fehlerhaft funktioniert“[bro03]
Ausgang (σ^a)	*Signal*, welches durch eine SPS aktiviert werden kann und einen Prozess in der Umwelt der SPS anstößt (vgl. S. 28)
Ausgangsbelegung (Σ^a)	Menge der aktiven Ausgänge (vgl. S. 28)
Drift	Abweichung im Gang zweier Uhren
Echtzeitfähigkeit	Fähigkeit eines *Prozessrechners*, innerhalb einer festgelegten Zeitspanne zu reagieren
Eingang (σ^e)	*Signal*, welches das Auftreten eines Ereignisses in der Umwelt einer *SPS* anzeigt (vgl. S. 28)
Eingangsbelegung (Σ^e)	Menge der aktiven Eingänge (vgl. S. 28)
Endlicher Automat	„Modell zur Verarbeitung von Datenstrukturen; mit endlicher Kontrollstruktur ohne Arbeitsspeicher zur Verarbeitung von Zeichenketten“[sch98]
Ereignisorientierte Arbeitsweise	Arbeitsweise, bei der ein Prozessrechner sein Steuerprogramm beim Eintreten eines Ereignisses ausführt (vgl. auch *zyklische Arbeitsweise*)
Fehlerfreiheit	Eine Fehlerrate von 0
Fehlerrate	Anzahl der Fälle, in denen ein Programm oder Modell gegen seine *Spezifikation* verstößt, bezogen auf die Größe des Programms oder Modells
Floyd-Hoare-Logik	Formales System zum Beweis der Korrektheit eines Computerprogramms

Begriff	Beschreibung
Hilfsprogramm	Hier: Computerprogramm zur Unterstützung bei der Entwicklung einer Steuerung
Maximale Zykluszeit (Δ)	Obere Zeitgrenze der *Zykluszeit* (vgl. S. 30)
Messfehler	Maximale Abweichung des gemessenen vom tatsächlichen Wert
Minimale Zykluszeit (δ)	Untere Zeitgrenze der *Zykluszeit* (vgl. S. 30)
Modell	„Materielles oder Gedankenobjekt, das einem Untersuchungsgegenstand in bestimmten Eigenschaften oder Relationen entspricht“[bro02]
Modellprüfer	*Hilfsprogramm* zur Prüfung der Eigenschaften eines *Modells*
Petrinetz	„Grafische Darstellung zur Beschreibung und Analyse von sog. nebenläufigen Prozessen“[bro03]
Prozessrechner	„Frei- oder austauschprogrammierbare Datenverarbeitungsanlage, die durch Echtzeitfähigkeit und Einzelbitverarbeitung sowie Einrichtungen zur Ein- und Ausgabe von Prozesssignalen gekennzeichnet ist“[bro02]
Reaktionszeit	„Die zwischen einem Reiz und der darauf erfolgenden Reaktion verstreichende Zeitspanne“[bro02]
Signal (σ)	„Die Darstellung von Nachrichten oder Daten mit physikalischen Mitteln“(vgl. S. 28)
Speicherprogrammierbare Steuerung (SPS)	In der Norm *IEC 61131-3* standardisierter, weit verbreiteter Typ eines *Prozessrechners*; kann in *Structured Text* programmiert werden
Spezifikation	Allgemein: „Die detaillierte Beschreibung eines Sachverhalts“[bro03]; Hier: Beschreibung der Eigenschaften einer Steuerung
Steuerprogramm	Menge von Regeln, die für einen *Prozessrechner* die Umwandlung von Eingangs- in Ausgangsinformationen beschreiben
Steuerung	„Prozess, bei dem Ausgangsgrößen zielgerichtet entsprechend vorgegebenen Eingangsgrößen beeinflusst werden“[bro02]
Structured Text (ST)	Modulare Sprache zur Programmierung von *SPS*

Begriff	Beschreibung
Symbolische Repräsentation	Zusammenfassung einer meist unendlichen Menge von Zuständen durch mathematische Darstellung gemeinsamer Eigenschaften dieser Zustände
Synchronität	Eigenschaft zweier Uhren, auf dem selben Takt zu basieren und die selbe Zeit zu messen
Timed Net Condition Event System (TNCES)	Modulares *Petrinetzmodell*, das um *Zeitbedingungen* erweitert wurde
Übersetzer	*Hilfsprogramm*, das ein Quellprogramm oder -modell in ein Zielprogramm oder -modell umwandelt, wobei das Verhalten der Quelle im Ziel erhalten bleibt
Uhrenstand	Der von einer Uhr zu einem Zeitpunkt angezeigte Wert
Zeitbedingung	Allgemein: (Un-)gleichung auf den Uhrenständen eines Zeitsystems; Hier: Vergleich von Uhren mit Konstanten (vgl. Definition 4.3)
Zyklische Arbeitsweise	Arbeitsweise, bei der ein Prozessrechner sein Steuerprogramm regelmäßig (zyklisch) ausführt, unabhängig vom Eintreten bestimmter Ereignisse (vgl. S. 30)
Zyklus	In sich geschlossener Arbeitskreis, der fortwährend von einer *SPS* ausgeführt wird (vgl. S. 30)
Zykluszeit (τ)	Zeit, die eine *SPS* zur Ausführung eines Zyklus benötigt (vgl. S. 30)

Abbildungsverzeichnis

1 Einleitung

Computer werden seit Jahren in nahezu allen industriellen Bereichen wie dem Automobilbau, der Flugzeugindustrie oder der chemischen Industrie zur *Steuerung* eingesetzt. Auf dieses Aufgabengebiet zugeschnittene Computer werden zur besseren Unterscheidung als *Prozessrechner* bezeichnet. Prozessrechner unterliegen oft besonderen Anforderungen an ihre *Sicherheit*, da eine Fehlfunktion z.B. bei einem Flugzeug in der Luft schwerwiegende Folgen haben kann. Sicherheit ist ein allgemeiner Oberbegriff und schließt Aspekte wie beispielsweise die *Fehlerrate*, *Ausfallhäufigkeit* und *Reaktionszeit* des Prozessrechners ein. Die Sicherheit der gesamten Steuerung wird durch das *Zusammenspiel ihrer Komponenten* gewährleistet, wobei jede Komponente einen oder mehrere der genannten Aspekte sicherstellt.

Um die Sicherheit einer Steuerung zu prüfen werden seit vielen Jahren *Modelle* eingesetzt. Diese erlauben das ungefährliche und formale Prüfen von Sicherheitsaspekten einzelner Komponenten oder einer gesamten Steuerung. Darüber hinaus können Modelle auch zum Entwurf von Steuerungen genutzt werden. Um die Sicherheit der realen Steuerung nicht zu gefährden, muss in diesem Fall gewährleistet werden, dass die reale Steuerung exakt das Verhalten des Modells besitzt. *Diese Forderung für eine spezielle Art von Modellen zu gewährleisten ist Anliegen dieser Arbeit.*

Die Arbeit entstand aus einer praktischen Aufgabenstellung und bezieht sich nur auf einen Teil des Gebietes der Sicherheit von Steuerungen. Der ersten Abschnitt der Einleitung gibt deshalb einen groben *Überblick* über das Gesamtgebiet. Dieser Überblick dient als Ausgangspunkt für die daran anknüpfende *Problemstellung*. Nach der Erläuterung des Problems wird der *Lösungsansatz* skizziert. Aus diesem Ansatz leitet sich der abschließend vorgestellte *Aufbau* der Arbeit ab.

1.1 Allgemeines

Speicherprogrammierbare Steuerungen

Es gibt eine Vielzahl von Prozessrechnerherstellern und eine Vielzahl von Prozessrechnern mit unterschiedlichen Eigenschaften. Eine Gemeinsamkeit aller Prozessrechner sind ein oder mehrere *Ein-* und *Ausgänge*, mit denen sie Informationen mit ihrer Umwelt austauschen. Daneben besitzt jeder Prozessrechner ein *Steuerprogramm*, nach dem er aus den Informationen an den

Eingängen Informationen seiner Ausgänge erzeugt. Für diese Arbeit relevante Unterschiede zwischen Prozessrechnern bestehen in der *Arbeitsweise*, dem *Umgang mit Zeit* sowie der *Art der Informationen* an Ein- und Ausgängen.

Prozessrechner unterscheiden sich in der *Art der Informationen*, die sie verarbeiten und mit ihrer Umwelt austauschen können. Die Ein- und Ausgänge sind beispielsweise mit elektronischen Thermometern oder Ventilen verbunden und messen oder erzeugen zu einem Zeitpunkt einen bestimmten (Spannungs-)*Wert*. Sofern der Prozessrechner intern digital arbeitet, kann er für jeden Ein- oder Ausgang nur endlich viele verschiedene Werte unterscheiden. Eine Vorgabe dieser Arbeit ist die Verwendung *binärer Ein- und Ausgänge*, d.h. der Prozessrechner kann für jeden Ein- oder Ausgang genau zwei Werte unterscheiden.

Ein weiterer Unterschied zwischen Prozessrechnern besteht im *Umgang mit Zeit*. Die meisten Prozessrechner müssen innerhalb einer vorgegebenen Zeitspanne auf eine Änderung des Wertes eines Eingangs reagieren. Diese Eigenschaft wird als *Echtzeitfähigkeit* des Prozessrechners bezeichnet. Verschärfend kommt für diese Arbeit die Forderung hinzu, dass der Prozessrechner Uhren besitzt. Diese Uhren kann der Prozessrechner nutzen, um *Zeitbedingungen* zu prüfen und auf diese zu reagieren. Welcher Art diese Zeitbedingungen sein können, wird später genauer definiert. Die Annahme von Zeitbedingungen wirkt verschärfend, da es nicht sinnvoll ist, Zeitbedingungen zu prüfen, ohne dies innerhalb einer angemessenen Zeitspanne zu tun.

Die Arbeitsweise von Prozessrechnern wird in eine *ereignisorientierte* oder *zyklische Arbeitsweise* unterschieden. Ein Prozessrechner mit ereignisorientierter Arbeitsweise führt sein Steuerprogramm erst dann aus, wenn sich der Wert eines Eingangs geändert hat, also ein *Ereignis* eingetreten ist. Bei der zyklischen Arbeitsweise liest der Prozessrechner in regelmässigen Abständen die Werte seiner Eingänge und führt jedes Mal sein Steuerprogramm aus, unabhängig von einer Änderung der Werte der Eingänge.

Für diese Arbeit wurden als Prozessrechner *Speicherprogrammierbare Steuerungen* (Abk. SPS, Engl. Programmable Logic Controllers - PLC)[WR99] vorgegeben. Bedingt durch ihre einfache Programmierung und weitgehende Standardisierung fanden SPS eine weite Verbreitung in der Industrie. SPS arbeiten *zyklisch*, sind *echtzeitfähig* und besitzen *Uhren*, mit denen sie *Zeitbedingungen* prüfen können. Die Programmierung von SPS wurde in der Norm *IEC 61131-3* [Int03] standardisiert und ist unter anderem in der PASCAL-ähnlichen Sprache *Structured Text* möglich.

Zusammenfassend sind folgende Eigenschaften für die vorausgesetzten Prozessrechner dieser Arbeit maßgebend:

1. Mehrere binäre Ein- und Ausgänge,
2. Zeitbedingungen sowie
3. die zyklische Arbeitsweise.

Modellbasierte Entwicklung

Ein *Modell* ist ein *vereinfachtes* Abbild der Wirklichkeit. In dieser Arbeit werden mathematische Modelle betrachtet, die folgende Eigenschaften besitzen:

Unabhängigkeit: Modelle vereinfachen reale Eigenschaften und können unabhängig von diesen Eigenschaften eingesetzt werden.

Einfachheit: Durch die Vereinfachung sinkt die Komplexität des Modells im Vergleich zur Realität.

Wandelbarkeit: Modelle können mit einem *Übersetzer* (Engl. Compiler) automatisch in andere Modelle oder Programme umgewandelt werden.

Prüfbarkeit: Mit einem *Modellprüfer* (Engl. Model Checker) kann geprüft werden, ob ein Modell bestimmte Bedingungen erfüllt.

Der letzte Punkt, die Prüfbarkeit von Modellen, ist für die Sicherheit von Steuerungen besonders interessant. Jede Steuerung muss eine Reihe von Bedingungen erfüllen. Zum Beispiel dürfen Bahnschranken erst dann geöffnet werden, wenn der Zug die Schranke passiert hat. Derartige Bedingungen werden in einer *Spezifikation* zusammengefasst. Anhand eines Modells einer Steuerung kann überprüft werden, ob das Modell die Bedingungen der Spezifikation erfüllt [CGP99].

In dieser Arbeit wird der *umgekehrte Weg* beschritten, um die Vorteile von Modellen zu nutzen. Dazu wird nicht aus einer Steuerung ein Modell erstellt, sondern umgekehrt aus einem Modell eine Steuerung.

Speziell zum Entwurf eines Steuerprogramms kann folgendermaßen vorgegangen werden:

1. Ein *Modell des Steuerprogramms* wird entworfen.
2. Mit Hilfe eines Modellprüfers wird *geprüft*, ob das Modell die Bedingungen der *Spezifikation* erfüllt.
3. Ist dies nicht der Fall, wird das *Modell korrigiert*.
4. Erfüllt das Modell die Bedingungen der Spezifikation, wird es mit Hilfe eines Übersetzers automatisch in ein *Steuerprogramm umgewandelt*.

Die genannten Schritte sind in Abbildung 1.1 dargestellt. Modellprüfer und Übersetzer wurden zur besseren Unterscheidung umrandet.

Die Verwendung von Modellen zum Entwurf von Steuerungen wird in dieser Arbeit unter dem Begriff *Modellbasierte Entwicklung* (Engl. *Model-Driven Engineering*) zusammengefasst. Diese Technik ist für Steuerungen schon länger bekannt [BK88, VCD+85]. Bei der Entwicklung von Software halten Modelle insbesondere unter dem Begriff der *Modellbasierten Architektur* (Engl. *Model-Driven Architecture*) [S+00] Einzug.

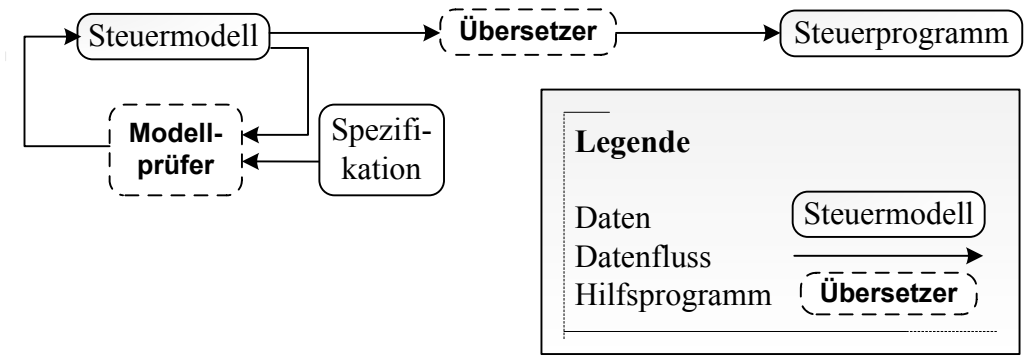


Abbildung 1.1: Softwareentwicklung bei Verwendung eines Modells.

Für diese Arbeit wurde eine spezielle Klasse von Modellen vorgegeben, sogenannte *Timed Net Condition Event Systems* (Abk. TNCES). TNCES sind *zeitbewertete Petrinetze* und erlauben die einfache *grafische Entwicklung* sowohl eines Modells der Steuerung als auch des gesteuerten Prozesses. TNCES bestehen dazu aus *Modulen*, die miteinander verbunden werden. Ferner besitzen die Plätze in TNCES ein *Alter*. Dieses Alter erlaubt die Modellierung von *Zeitbedingungen*. Modellprüfer erlauben es, Bedingungen für TNCES zu prüfen. Durch einen Übersetzer erfolgt die Umwandlung von TNCES in ein SPS-Steuerprogramm, das auf einer SPS in einer realen Anlage zur Steuerung eingesetzt werden kann. Für eine Definition von TNCES und eine ausführliche Beschreibung der Eigenschaften wird auf [Thi02] verwiesen.

Zusammenfassend sind folgende Eigenschaften von Modellen wichtig:

1. Modelle vereinfachen die Realität.
2. Modelle können zum Entwurf einer Steuerung verwendet werden.
3. Ein Modell kann bezüglich einer Spezifikation geprüft werden.

Sicherheit

Mit dem Thema Sicherheit beschäftigt sich die Informatik seit ihren Anfängen. Dabei wurde klassisch vor allem die *Fehlerrate* von Programmen untersucht. Die Fehlerrate beschreibt bezogen auf die Größe eines Programms die Anzahl der Fälle, mit der das Programm gegen seine Spezifikation verstößt. Im Idealfall beträgt die Fehlerrate Null, d.h. das Programm ist fehlerfrei. Die *Fehlerfreiheit* kann beispielsweise mit der *Floyd-Hoare-Logik* [Flo67, Hoa69] für viele Programme mathematisch nachgewiesen werden.

Wie bereits am Anfang des Kapitels angedeutet, existieren neben der Fehlerrate noch *andere Aspekte*, die für die Sicherheit relevant sind. Zu diesen Aspekten zählt z.B. die *Reaktionszeit* oder der maximale *Messfehler*. Für jeden Aspekt bestehen je nach Anwendungsfall unterschiedliche *Anforderungen*, d.h. Bedingungen, die von der Steuerung erfüllt werden müssen.

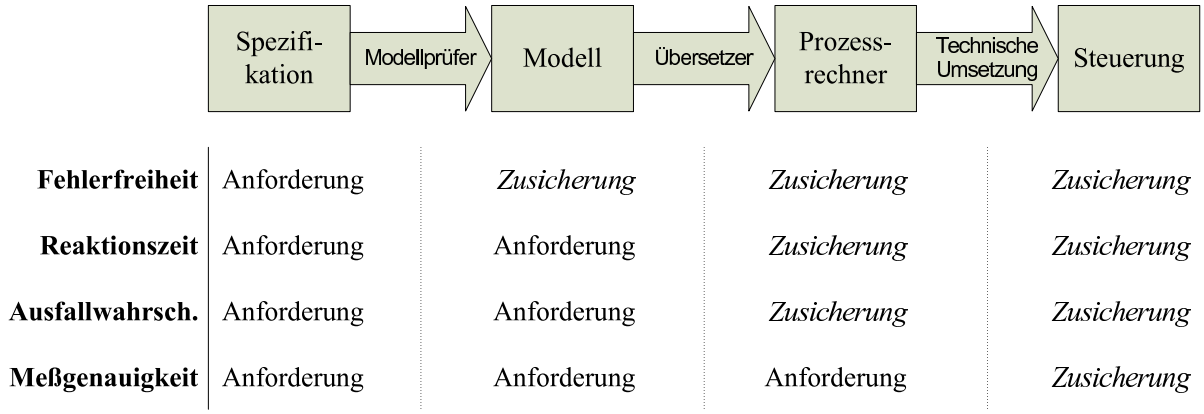


Abbildung 1.2: Phasen bei der Entwicklung einer Steuerung.

Beispielsweise bestehen andere Anforderungen an den Messfehler eines Fieberthermometers als an den eines Hochofenthermometers. Wird von der Steuerung eine Anforderung garantiert, besteht eine *Zusicherung*.

Bei der Entwicklung einer Steuerung werden *mehrere Phasen* durchlaufen, die aufeinander aufbauen. Grob können folgende Phasen unterschieden werden: 1. Erstellen einer Spezifikation der Steuerung 2. Entwurf eines Modells 3. Entwicklung eines Steuerprogramms für den Prozessrechner 4. Technische Umsetzung der Gesamtsteuerung. Nicht in jeder Phase können die Anforderungen jedes Aspekts gewährleistet werden, sondern oft können nur Anforderungen an nachfolgende Phasen beschrieben werden. Beispielsweise kann der Messfehler beim Entwurf des Modells nicht beeinflusst werden – bereits beim Entwurf des Modells wird allerdings ein maximaler Messfehler vorausgesetzt. Erst in der letzten Phase, d.h. bei der technischen Umsetzung der Gesamtsteuerung, müssen die Anforderungen aller Aspekte erfüllt werden.

In Abbildung 1.2 sind die Phasen bei der Entwicklung einer Steuerung vereinfacht dargestellt. Darunter sind ausgewählte Aspekte in Fettschrift aufgelistet. Unter jeder Phase ist dargestellt, ob in der Phase eine *Anforderung* oder *Zusicherung* des Aspektes besteht. Nachfolgende Phasen dürfen die zugesicherten Aspekte nicht *verletzen*, so entsteht eine *Kette von geforderten und zugesicherten Aspekten*.

Beispielsweise werden in der Spezifikation die Anforderungen an die Fehlerfreiheit formuliert. Bei der Entwicklung des Modells wird die Fehlerfreiheit des Modells zugesichert. Alle nachfolgenden Phasen müssen gewährleisten, dass diese Zusicherung erfüllt bleibt. Die Reaktionszeit ist eine weitere Anforderung in der Spezifikation. Im Modell kann die Reaktionszeit nicht beeinflusst, höchstens konkretisiert werden. Erst der Prozessrechner sichert die tatsächliche Reaktionszeit zu, wobei eventuell weitere Anforderungen an die Steuerstrecke bestehen.

Für den Übergang von einer in die nächste Phase werden meist *Hilfsprogramm* verwendet. Sie unterstützen die Entwicklung und helfen sicherzustellen, dass Anforderungen erfüllt werden

und Zusicherungen unverletzt bleiben. Beispielsweise kann beim Entwurf des Modells ein Modellprüfer als Hilfsprogramm verwendet werden. Mit diesem werden die Anforderungen der Spezifikation an die Fehlerfreiheit des Modells sichergestellt. Hilfsprogrammen kommt deshalb eine besondere *Bindegliedfunktion* zu.

Zusammenfassend sind folgende Aussagen wichtig:

1. Sicherheit umfasst unterschiedliche Aspekte.
2. Beim Entwurf einer Steuerung werden Phase für Phase einzelne Aspekte gefordert oder zugesichert, es entsteht eine Kette aus Anforderungen und Zusicherungen.
3. Hilfsprogramme haben in dieser Kette eine Bindegliedfunktion, da sie die einzelnen Phasen miteinander verbinden.

1.2 Problemstellung

Der Einsatz eines Modellprüfers stellt sicher, dass ein Modell seiner Spezifikation genügt. Neuere Arbeiten beschäftigen sich mit dem Nachweis der Fehlerfreiheit eines Mikroprozessors [BJK+06] sowie dem Versuch, die Fehlerfreiheit eines kompletten Prozessrechners nachzuweisen [Pau05].

Durch dieses Vorgehen wird versucht, die Sicherheit für das Modell und den Prozessrechner zu gewährleisten. Unklar bleibt jedoch, ob bei der Übersetzung des Modells in ein Steuerprogramm Fehler aufgetreten sind oder Annahmen des Modells in der Realität nicht zutreffen. Es entsteht eine *Lücke* in der Kette aus Zusicherungen. Aus dieser und den vorangegangenen Überlegungen ergibt sich das *Hauptziel* dieser Arbeit:

> **Ein Hilfsprogramm soll erstellt werden, das Folgendes erlaubt: Für ein beliebiges TNCES-Petrinetzmodell und ein dazugehöriges Steuerprogramm in Structured Text wird praktisch nachgewiesen, dass das Steuerprogramm das Verhalten des TNCES-Modells korrekt nachbildet.**

Aus dieser Aufgabenstellung ergeben sich drei wesentliche Fragen:

1. **Was heißt „korrekt“?**
2. **Treffen die Annahmen des Modells in der Realität zu?**
3. **Wie kann die Korrektheit praktisch nachgewiesen werden?**

1.3 Lösungsansatz

Im Folgenden werden die Antworten auf die gestellten Fragen kurz skizziert.

Frage 1: *Was heißt „korrekt“?*

Der Korrektheitsbegriff hat folgende Eigenschaften:

1. Mit einem Modellprüfer wird sichergestellt, dass ein Modell bestimmte Eigenschaften *immer* oder *nie* erfüllt. Der definierte Korrektheitsbegriff stellt sicher, dass das Steuerprogramm jede dieser Eigenschaften ebenfalls immer oder nie erfüllt.
2. Modelle besitzen oft Freiheiten bezüglich ihres Verhaltens, d.h. sie können in der selben Situation unterschiedlich reagieren. Eine reale Steuerung sollte dem entgegen stets nur ein Verhalten besitzen. Der Korrektheitsbegriff berücksichtigt diesen Umstand.

Frage 2: *Treffen die Annahmen des Modells in der Realität zu?*

Es wurde bereits erwähnt, dass Speicherprogrammierbare Steuerungen eine zyklische Arbeitsweise mit Reaktionszeiten besitzen. In Kapitel 4 wird gezeigt, dass Reaktionszeiten das *Verhalten eines Modells* mit Zeitbedingungen *beeinflussen* können. Reaktionszeiten müssen deshalb bei der Übersetzung des Modells in ein Steuerprogramm beachtet werden. Diesem Umstand wird durch die explizite Betrachtung von Reaktionszeiten Rechnung getragen.

Frage 3: *Wie kann die Korrektheit praktisch nachgewiesen werden?*

Eine Möglichkeit, die Korrektheit einer Übersetzung zu garantieren, ist der Nachweis der Fehlerfreiheit des Übersetzers. Für diese Arbeit wurde eine vom Übersetzer unabhängige und dadurch flexiblere Möglichkeit gewählt: Die *Translation Validation*. Dazu werden nach der Übersetzung Steuerprogramm und Modell miteinander verglichen und festgestellt, ob das Verhalten des erzeugten Programms bezüglich des Korrektheitsbegriffs dem des Modells entspricht.

Eine Herausforderung ergibt sich aus der Komplexität der verwendeten Modelle und Steuerprogramme, die einen direkten Vergleich erschwert. So beansprucht die Definition von TNCES in [Thi02] ca. 50 Seiten, die Norm *IEC 61131-3* [Int03] umfasst über 200 Seiten.

Um dieser Herausforderung zu begegnen, werden sowohl das TNCES-Modell als auch das Steuerprogramm auf ein einfaches *Basismodell* zurückgeführt. Eine sehr einfache Form von Modellen sind Automaten. Automaten mit Zeitbedingungen werden als *Zeitautomaten* (Engl. timed automata)[AD94]) bezeichnet. Es existiern zahlreiche Varianten von Zeitautomaten mit unterschiedlichen Eigenschaften, je nach der Art der betrachteten Zeitbedingungen, Anzahl

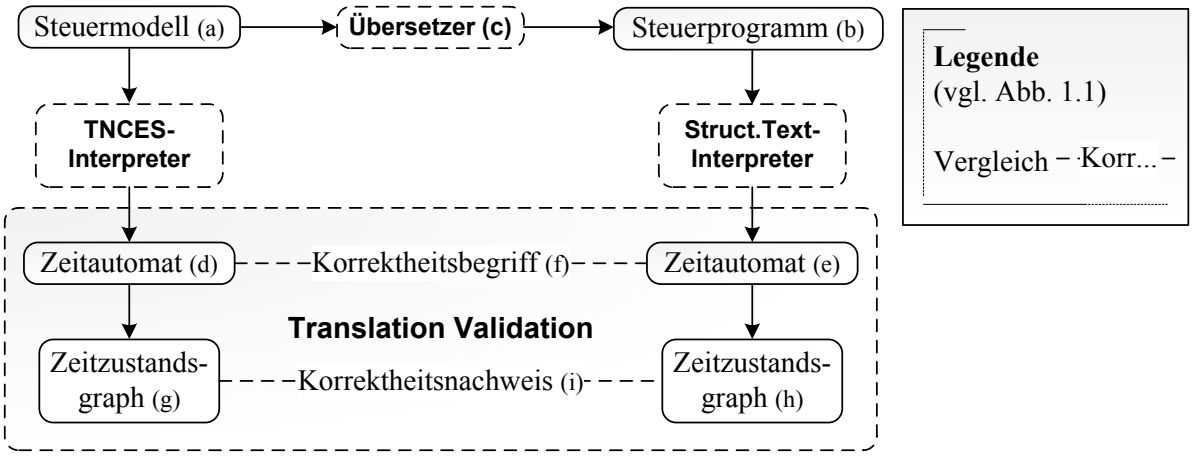


Abbildung 1.3: Vorgehen zur Translation Validation von TNCES-Modellen und Structured Text-Programmen.

der Uhren sowie möglichen Ein- und Ausgaben. Um die einfache und effiziente Modellierung von Steuerprogrammen und TNCES-Modellen zu erlauben, wurden Zeitautomaten aufgabenspezifisch angepasst.

Die Analyse von Modellen mit Zeitbedingungen erfordert komplexe Algorithmen, deren Speicherplatz- und Rechenzeitbedarf im ungünstigsten Fall exponentiell in der Anzahl der Uhren wächst [AD94]. Deshablb ist dies Suche nach effizienten Möglichkeiten zur Analyse von Modellen mit Zeitbedingungen seit vielen Jahren ein Forschungsschwerpunkt.

Der Vergleich von Zeitautomaten stellt dabei eine besondere Herausforderung dar. Ziel ist es, einen solchen Vergleich in *annehmbarer Geschwindigkeit* und bei *annehmbarem Speicherverbrauch* durchzuführen, d.h. in ähnlicher Geschwindigkeit und mit ähnlichem Speicherverbrauch wie die Modellprüfung. Folgende Überlegung steht hinter dieser Aussage: Der Korrektheitsnachweis der Übersetzung eines Modells ist nicht sinnvoll, wenn die Korrektheit des Modells nicht geprüft werden kann.

Praktisches Vorgehen

Um dieses Ziel zu erreichen, wird auf den Forschungsergebnissen zur Analyse von Zeitautomaten aufgebaut. Die Techniken zur Analyse von Zeitautomaten werden für den Vergleich erweitert und angepasst, wobei Reaktionszeiten aus Effizienzgründen in den Vergleich direkt einbezogen werden. Das Vorgehen zum Vergleich eines Steuerprogramms und TNCES-Modells ist in Abbildung 1.3 schematisch dargestellt.

Als Steuermodelle werden in dieser Arbeit ausschließlich *TNCES-Modelle* betrachtet (a). Als Steuerprogramme werden ausschließlich Structured Text-Programme betrachtet, deren Verhalten wird von der Speicherprogrammierbaren Steuerung, deren Arbeitsweise und Eigenschaften

wie Reaktionszeiten vorgegeben (b). Ein *Übersetzer* (c) transformiert das TNCES-Modell in ein Steuerprogramm, wobei die Korrektheit dieser Übersetzung nachgewiesen werden soll.

Um die Translation Validation durchzuführen, werden Steuermodell und -programm in Zeitautomaten umgewandelt (d,e). Dazu wurden *Interpreter für TNCES und Structured Text* entwickelt, die es erlauben ein TNCES-Modell bzw. ein ST-Programm in einen Zeitautomaten zu transformieren. Die Verwendung von Zeitautomaten hat neben der *Vereinfachung der Translation Validation* folgende Vorteile:

1. Die Translation Validation kann einfach auf *weitere Arten* von Modellen und Steuerprogrammen übertragen werden.
2. Die Translation Validation kann auch zur Prüfung der Korrektheit von *Modell- oder Programmtransformationen* verwendet werden.
3. Der Kern an Operationen muss nur *einmal* definiert, bewiesen und implementiert werden.
4. Das *umfangreiche Wissen* aus der Theorie der Zeitautomaten kann für die Translation Validation nutzbar gemacht werden.

Der für die Translation Validation verwendete *Korrektheitsbegriff* (f) legt fest, wann ein Zeitautomat das Verhalten des anderen korrekt nachbildet. Um den Korrektheitsbegriff praktisch zu prüfen, wird das Verhalten der beiden Zeitautomaten miteinander verglichen. Das Verhalten eines Zeitautomaten hängt sowohl von dem *Zustand* ab, in dem sich der Zeitautomat befindet, als auch vom *Stand der Uhren* des Zeitautomaten. Die Kombination von Zuständen und *Uhrenständen* der Zeitautomaten werden *symbolisch repräsentiert* (g,h) und bilden *Graphen*, die im Wesentlichen *endlichen Automaten* entsprechen. Anhand dieser Graphen wird die Korrektheit der Übersetzung praktisch geprüft (i) und die Translation Validation abgeschlossen.

1.4 Aufbau der Arbeit

Im anschließenden Kapitel 2 werden Alternativen zum obigen Ansatz und wissenschaftliche Arbeiten diskutiert, die vergleichbare Ziele verfolgen. Darauf aufbauend werden in Kapitel 3 die für diese Arbeit wichtigen Definitionen und Begriffe eingeführt.

Ein Zeitautomatenmodell zur einheitlichen Modellierung von Programmen für Speicherprogrammierbare Steuerungen und TNCES-Modellen wird in Kapitel 4 entwickelt. Außerdem wird gezeigt, wie diese Zeitautomaten analysiert und effizient miteinander verglichen werden können. In Kapitel 5 wird gezeigt, wie Zeitzustände und -graphen von Zeitsystemen effizient dargestellt werden können. Die praktische Umsetzung einer Translation Validation ist Thema des Kapitels 6.Anschließend wird noch eine Zusammenfassung der Arbeit und ein kurzer Ausblick auf mögliche fortführende Arbeiten gegeben.

2 Verwandte Arbeiten

Die wesentliche Aufgabenstellung dieser Arbeit, die Prüfung der Korrektheit der Implementierung eines Modells, leitet sich aus der Praxis ab. Dabei werden mehrere Forschungsbereiche überschnitten:

1. Der Einsatz von Modellen für den Entwurf von Steuerungen ist seit langem eine Fragestellung auf dem Gebiet der Ingenieurstechnik, gewinnt unter anderem in Form der Model Driven Architecture aber auch für Softwaresysteme im Bereich des Softwareengineering an Bedeutung.
2. Besondere Fragestellungen werfen neben diskreten Modellen solche Modelle auf, die die Formulierung von Zeitbedingungen erlauben. Diese Fragestellungen haben zu einer Herausbildung einer eigenen Theorie von Zeitsystemen und -modellen geführt.
3. Ein weiterer Problemkreis ist die Analyse und Verifikation von Modellen, bei dem unter anderem die Darstellung einer grossen Anzahl von Modellzuständen von Interesse ist. Diese Fragestellung hat zu einer Vielzahl an Forschungsaktivitäten auf dem Gebiet geeigneter Datenstrukturen geführt.
4. Schließlich sind für die Implementierung von Modellen Methoden aus dem Übersetzerbau von Bedeutung. Besonders der Nachweis der Korrektheit einer Übersetzung ist aus Sicht der Verifikation eines Modells interessant, um die Korrektheit einer Modellimplementierung festzustellen.

Eine grobe Einordnung dieser Arbeit in das Forschungsumfeld der Korrektheit von Übersetzern wird im ersten Abschnitt vorgenommen. Anschließend werden verschiedene Modelle vorgestellt, die zur Modellierung von Zeitsystemen verwendet werden können. Spezielle Anforderungen wie die Beachtung von Reaktionszeiten sowie die Übersetzung dieser Modelle sind Themen der darauffolgenden Abschnitte. Schließlich werden in den letzten beiden Abschnitten Methoden zur Analyse und zum Vergleich derartiger Modelle vorgestellt.

2.1 Korrektheit von Übersetzern

Dieser Abschnitt bezieht sich auf Punkt (c) aus Abbildung 1.3, den Übersetzer. Ein Übersetzer transformiert ein Quellprogramm oder -modell in ein Zielprogramm bzw. -modell. Es sind verschiedene Möglichkeiten bekannt, um die Korrektheit der Transformation bezüglich

vorgegebener Kriterien sicherzustellen. Zum einen kann die Korrektheit des Übersetzers direkt bewiesen werden, indem die Korrektheit der einzelnen Tranformationsschritte innerhalb des Übersetzers nachgewiesen wird. Es existieren mehrere Arbeiten auf diesem Gebiet, beispielsweise [HP92, Str02]. Die Erarbeitung und der Einsatz von Werkzeugen zur Konstruktion korrekter Übersetzer war Ziel des *Verifix*-Projektes [GDG+96, GGZ04].

Eine Technik, die im Idealfall völlig unabhängig vom Übersetzer angewendet werden kann, ist die *Translation Validation* [PSS98]. Dabei steht nicht der Beweis der Korrektheit des Übersetzers, sondern der Beweis der Korrektheit des Zielprogramms in Bezug zum Quellprogramm im Vordergrund. Dazu wird die Semantik des Quell- und Zielprogrammes durch Modelle nachgebildet und die Korrektheit der Transformation bezüglich einer Verfeinerungsrelation nachgewiesen. Eine Adaption dieser Technik für den Korrektheitsnachweis realer optimierender Übersetzer findet sich in [Nec00] und [ZPFG03].

Bewertung

Bei der Verifikation eines Übersetzers wird die Korrektheit der einzelnen Übersetzungsschritte meist mit Hilfe von Beweisprüfern formal nachgewiesen. Jede Änderung oder jeder zusätzliche Transformationschritt wie beispielsweise eine Optimierung muss als korrekt nachgewiesen werden. Dieser Prozess ist technisch für komplexe Übersetzer anspruchsvoll. Zudem ist ein Nutzer an einen Übersetzer gebunden.

Im Gegensatz dazu erlaubt die Translation Validation die Prüfung eines Zielprogrammes weitgehend unabhängig vom Übersetzer. Dazu werden Modelle zur Nachbildung der Semantik eines Quell- und Zielprogrammes genutzt und miteinander verglichen. Aufgrund der Unabhängigkeit vom Übersetzer wird auch die Verifikation der erzeugten Programme komplexer Übersetzer realistisch.

Nach unserer Kenntnis wurden weder eine Translation Validation noch der direkte Korrektheitsnachweis von Übersetzern für Zeitmodelle realisiert.

2.2 Modelle mit Zeitbeschränkungen

Dieser Abschnitt bezieht sich auf die verwendeten Modelle und Steuerprogramme, d.h. auf die Punkte (a) und (b) in Abbildung 1.3. Sowohl Speicherprogrammierbare Steuerungen als Zielsystem sowie Petrinetzmodelle als Quellmodell wurden vorgegeben. Die Eigenschaften Speicherprogrammierbarer Steuerungen werden in Kapitel 3 „Grundlagen“ genauer betrachtet, an dieser Stelle werden lediglich wissenschaftlich relevante Arbeiten zu Petrinetzen vorgestellt.

Petrinetze sind Modelle zur Darstellung nebenläufiger Systeme, die mindestens aus Plätzen, Transitionen, Marken sowie Flusskanten bestehen. Eine Erklärung dieser Begriffe findet sich z.B. in [Rei82].

Zeitbewertete Petrinetze

Der erste Vorschlag zur Erweiterung von Petrinetzen um Zeitbedingungen wird in [Mer74] beschrieben. Derartige Netze werden im weiteren als *Zeitbewertete Petrinetze* bezeichnet (engl. *Time Petri Nets*). Im Petrinetzmodell von [Mer74] brauchen die Transitionen eines Petrinetzes eine vorgegebene Zeit zum Schalten. Daneben sind noch verschiedene andere Möglichkeiten bekannt, Petrinetzen um Zeitbedingungen zu erweitern. Ein Überblick findet man in [Bow96].

Neben Transitionen können auch Plätze [CDD98] oder Kanten Zeitbedingungen besitzen. Im Falle zeitbewerteter Kanten kann entweder das Alter von Marken [AN01] oder das Alter der Plätze betrachtet werden. Dabei existieren je nach Semantik der Petrinetze oft Unterschiede in deren theoretischen Eigenschaften, was eine allgemeine Vergleichbarkeit zeitbewerteter Petrinetze erschwert [BCH+05].

Timed Net Condition Event Systems

Timed Net Condition Event Systems (kurz *TNCES*) sind modulare, zeitbewertete Petrinetze [Thi02]. In TNCES besitzen Plätze ein Alter, welches die Zeit angibt, die seit der letzten Änderung der Anzahl der Marken des Platzes vergangen ist. Von Plätzen ausgehende Flusskanten sind mit einem Zeitintervall beschriftet, das beschreibt, wie alt ein Platz sein muss, damit Marken aus dem Platz entfernt werden können. Ferner besitzen TNCES Inhibitor- und Eventkanten. Inhibitorkanten gehen von Plätzen aus und hemmen oder aktivieren Transitionen des TNCES. Eventkanten verbinden Transitionen untereinander und synchronisieren diese.

Inhibitor- und Eventkanten können unabhängige, d.h. nicht durch Flusskanten verbundene Petrinetze miteinander verbinden. Das ermöglicht die Einführung von TNCS-Modulen, wobei Inhibitor- und Eventkanten Ein- und Ausgänge eines Moduls bilden. Module können hierarchisch verschachtelt werden und erlauben den Aufbau komplexer Steuerungen aus einfachen Komponenten. Diese Vorgehensweise ermöglicht auch die unabhängige Modellierung von Steuerung und Steuerstrecke sowie die Untersuchung des Gesamtmodells.

Für TNCES existiert ein Modellprüfer zur Analyse bestimmter Bedingungen [Thi02] sowie die Möglichkeit zur Generierung von Steuerprogrammen für Speicherprogrammierbare Steuerungen [TH02].

Bewertung

TNCES stellen für den Anwender ein abstraktes und leistungsfähiges Modell für die einheitliche Modellierung und Analyse von Steuerung und Steuerstrecke zur Verfügung. Ein wesentlicher Vorteil von TNCES ist ihre Modularität, welche die Komplexität sowohl bei der Modellierung als auch der Analyse von TNCES verringert. Ein TNCES kann zudem mehrere, sich überlappende Eingangsignale betrachten. Neben der Überlappung kann auch die Zeitgleichheit

von Signalen berücksichtigt werden.

Der hohe Abstraktionsgrad und die Komplexität von TNCES stellen aus theoretischer Sicht allerdings einen Nachteil dar, da TNCES über eine sehr komplizierte Semantik verfügen. Insbesondere die Auswertung der Schaltregeln kann unter ungünstigen Umständen bei einer hohen Anzahl von Eventkanten zu einer hohen Laufzeitkomplexität führen, was die direkte Implementierung von TNCES unter Echtzeitbedingungen erschwert. Obgleich TNCES beschränkt sind, erfordert die hohe Anzahl möglicher Markierungen und Uhren eine effiziente Analyse. TNCES betrachten ferner keine Reaktionszeiten, die bei einer Implementierung auf einer realen Speicherprogrammierbaren Steuerung auftreten.

2.3 Basismodelle

Sowohl Steuerprogramme als auch TNCES haben eine sehr einsatzspezifische und komplexe Semantik und eignen sich deshalb nur bedingt für einen direkten Vergleich. Um die Abhängigkeit von der konkret verwendeten Programmiersprache und dem verwendeten Modell zu verringern und dabei auf einer einfachen und theoretisch fundierten Basis aufzusetzen, werden TNCES und Steuerprogramm auf ein Basismodell abgebildet. Dieser Abschnitt bezieht sich auf die Punkte (a) und (b) in Abbildung 1.3 und stellt mögliche Basismodelle vor.

2.3.1 SPS-Automaten

Das Projekt UniFormM [KBPOB99] der Universität Bremen hatte die Entwicklung von Methoden zur Spezifikation und Entwicklung von Echtzeitsystemen zum Ziel. Speziell zur Modellierung von SPS-Steuerungen wurden dazu *SPS-Automaten* (engl. PLC-Automata) [Die00] entwickelt, die die Modellierung von SPS-Steuerungen erlauben. Ein SPS-Automat besteht im Wesentlichen aus einer endlichen Menge von Zuständen und Zustandsübergängen sowie Ein- und Ausgaben. Zusätzlich kann für jeden Zustand angegeben werden, wie lange nach Erreichen des Zustands bestimmte Eingaben ignoriert werden sollen. Das ermöglicht die Modellierung einfacher Zeitbedingungen. Die Semantik von SPS-Automaten ist an die zyklische Arbeitsweise Speicherprogrammierbarer Steuerungen angepasst.

SPS-Automaten dienen vor allem dem Entwurf von SPS-Steuerungen. Sie können in die weiter unten beschriebenen Zeitautomaten nach Alur/Dill transformiert und mit Standardtools wie Uppaal [BDL04] oder Kronos [Yov97] verifiziert werden. Ein Steuerprogramm für SPS kann aus SPS-Automaten generiert werden. Dazu dient die Entwicklungsumgebung Moby-/PLC [Tap98, DT01].

Bewertung

SPS-Automaten stellen ein einfaches Modell für den Entwurf von SPS-Steuerprogrammen dar. Sie besitzen die für eine Steuerung notwendige Möglichkeit zur Ausgabe von Signalen und ahmen die Semantik einer Speicherprogrammierbaren Steuerung nach. Dabei werden die Reaktionszeiten der SPS beachtet. Ferner können SPS-Automaten verifiziert werden.

Ein SPS-Automat kann zu einem Zeitpunkt nur auf ein Eingabesignal reagieren, d.h. ein Zustandsübergang eines SPS-Automaten hängt von genau einem Eingangssignal ab. Treten zwei Eingangssignale gleichzeitig auf, steht nicht fest, welches der Signale vom SPS-Automaten zuerst betrachtet wird. Die Reihenfolge beeinflusst jedoch das Verhalten der Steuerung, wenn der Automat sich bei unterschiedlicher Signalreihenfolge unterschiedlich verhält. Das kann zu einem unerwünschten Nichtdeterminismus auch bei einem formal deterministischen SPS-Automaten führen. Da SPS-Automaten eine Reaktionszeit größer 0 besitzen und jeweils nur ein Signal pro Zustandsübergang betrachten, können sie das zeitgleiche Auftreten zweier Signale nicht feststellen.

Ein weiterer Nachteil von SPS-Automaten zeigt sich bei der Behandlung von Signalen, die über einen Zeitraum an einer Steuerung anliegen können. Für die Steuerung kann die Menge gleichzeitig anliegender, d.h. sich überlappender Signale sowie deren Dauer von Interesse sein. Das Auswerten der Dauer eines Signales ist mit SPS-Zeitautomaten nicht möglich. Um auf sich überlappende Signale zu reagieren, können Hilfssignale für den Anfang und das Ende eines Signals sowie Hilfszustände für die Menge der sich überlappenden Signale eingeführt werden. Die Anzahl der Hilfszustände kann allerdings exponentiell in der Anzahl sich überlappender Signale steigen.

Die Möglichkeit, Zeitbedingungen zu prüfen sind im Vergleich zu TNCES eingeschränkt, da SPS-Automaten nur eine implizite Uhr zur Sperrung von Eingangssignalen besitzen. Aufgrund der einfacheren Semantik ist die Modellierung komplexer Steuerungen im Vergleich zu TNCES möglicherweise aufwändiger.

2.3.2 Zeitautomaten

Ein einfaches Modell für Zeitsysteme mit mehreren Uhren sind Zeitautomaten. Zeitautomaten wurden ursprünglich von Alur und Dill [AD90, AD94, Alu99] als Erweiterung von Büchi-Automaten [Büc62] auf Eingabeworten unendlicher Länge definiert, da Zeitsysteme oft kontinuierlich laufen. Ein Zeitautomat besitzt wie ein endlicher Automat endliche Mengen von Zuständen, Eingabesymbolen und Zustandsübergängen, einen Startzustand sowie eine Menge von Finalzuständen. Zudem besitzt ein Zeitautomat eine endliche Menge von Uhren. Zustandsübergänge hängen vom Eingabesymbol sowie dem Stand der Uhren des Zeitautomaten ab, bei einem Zustandswechsel können Uhren zurückgesetzt werden. Zuständsübergänge erfolgen ohne zeitliche Verzögerung.

Zeitautomaten wurden in den vergangenen Jahren stark erforscht und zur Verifikation zahlreicher Zeitsysteme eingesetzt. Dazu wurden mehrer Modellprüfer entwickelt [Yov97, GPC03, BDL04]. Diese erlauben die Verifikation eines Modells bezüglich einer meist in Timed Computation Tree Logic (TCTL) [ACD90] verfassten Spezifikation. Andere Arbeiten beschäftigen sich mit der Erzeugung von Testfällen aus einer als Zeitautomaten gegebenen Spezifikation [CO00].

Varianten

Nach ihrer Definition wurden zahlreiche Erweiterungen von Zeitautomaten vorgeschlagen und untersucht. Zu den wichtigsten gehören Zeitautomaten mit Invarianten, bei denen für jeden Zustand Bedingungen auf den Uhren die Menge der Finalzustände ersetzen. Additive Bedingungen für Uhren ($u_1 + u_2 \leq c$, wobei u_1 und u_2 Uhren und c eines Konstante ist) werden in [BD00] untersucht. Welche Auswirkungen es hat, wenn Uhren nicht ausschließlich auf 0 sondern beispielsweise auf eine Konstante zurückgesetzt werden, wird in [BDFP04] untersucht. Zeitautomaten mit ϵ-Zustandsübergängen werden in [BGP96, BPDG98] untersucht. Zeitautomaten mit ϵ-Zustandsübergängen sind echt ausdrucksstärker als Zeitautomaten nach Alur/Dill. Unter welchen Bedingungen Zeitautomaten mit ϵ-Zustandsübergängen in Zeitautomaten nach Alur/Dill transformiert werden können, wird in [DGP97] untersucht. Eine weitere Variante sind Zeitautomaten, die Symbole ausgeben können (timed i/o automata [KLSV03, KLSV06]).

Bewertung

Zeitautomaten erkennen Wörter, deren einzelne Buchstaben mit einem Zeitpunkt versehen werden. Dieser Ansatz hat analog zu endlichen Automaten zu einer Theorie zeitbewerteter Sprachen geführt. Aufgrund ihrer Einfachheit und wesentlichen Fortschritten in der Theorie sowie durch die Entwicklung entsprechender Werkzeuge haben sich Zeitautomaten vielfach bei der Modellieren und Validieren von Zeitsystemen in der Praxis bewährt. Da Zeitautomaten keine Ausgabe besitzen, eignen sie sich für Steueraufgaben allerdings nur bedingt. Insbesondere für die Beantwortung der Fragestellung dieser Arbeit besitzen Zeitautomaten einige Nachteile:

Signale einer Steuerung erstrecken sich in der Realität oft über einen Zeitraum. Zeitbewertete Steuerungen müssen auf Signale mit Dauer reagieren können, beispielsweise um festzustellen, welche Signale gleichzeitig anliegen und benötigen einen Mechanismus, derartige Signale abzubilden. Damit ein Zeitautomat nach Alur/Dill auf mehrere anliegende Signale reagieren kann, muss für jeden möglichen Vorgang ein Start- und Endsymbol eingeführt werden, die entsprechend den Anfang oder das Ende eines Signals beschreiben. Daneben muss der Zeitautomat sich in Zwischenzuständen merken, welche Signale aktiv oder nicht aktiv sind. Die entstehende Menge von Zwischenzuständen steigt exponentiell in der Anzahl betrachteter Vorgänge.

Ein Zustandsübergang eines Zeitautomaten ist jeweils von genau einem Eingabesymbol des Eingabeworts abhängig. Die Definition der Eingabe von Zeitautomaten in [AD94] fordert einen positiven Zeitabstand zwischen zwei aufeinanderfolgenden Symbolen und schließt damit das gleichzeitige Auftreten mehrerer Symbole aus. In den Grenzen der Genauigkeit einer realen Steuerung können Vorgänge durchaus gleichzeitig stattfinden. Da keine Zeit zwischen gleichzeitigen Symbolen vergeht, existiert a priori keine Ordnung zwischen diesen Symbolen. Von der Ordnung der Symbole kann aber das Verhalten des Zeitautomaten abhängen und gleichzeitig eintretende Symbole führen zu einem unerwünschten Nichtdeterminismus bei formal deterministischen Zeitautomaten. Um Zeitautomaten zu konstruieren, die sich auch im Fall gleichzeitig eintretender Symbole deterministisch verhalten, muss die Gleichzeitigkeit zweier Symbole in einem Zwischenzustand erkannt werden. Die Anzahl solcher Zwischenzustände steigt exponentiell in der Anzahl möglicherweise gleichzeitig auftretender Symbole.

Zeitautomaten benötigen nach Alur/Dill keine Zeit, um Zustandsübergänge auszuführen. Der Annahme entsprechend, dass physikalische Vorgänge, so schnell sie auch seien mögen, stets eine endliche Geschwindigkeit besitzen, haben reale Systeme immer eine Verzögerung. Zustandswechsel beispielsweise eines Steuercomputers benötigen deshalb in der Realität stets Zeit. Die Zeit, die ein Steuercomputer für einen Zustandswechsel benötigt, kann meist nach oben abgeschätzt werden und wird als maximale Reaktionszeit bezeichnet.

Um die maximale Reaktionszeit eines Systems mit Zeitautomaten nach Alur/Dill zu modellieren, muss ein die Umgebung des Zeitautomaten repräsentierendes Symbol eingeführt werden, welches die Zustandswechsel des Zeitautomaten von außen anstößt. Des Weiteren müssen für jede Transition des Zeitautomaten Zwischenzustände eingefügt werden, die die Reaktionszeit der Transition repräsentieren. Die korrekte Nachbildung der komplexen Semantik beispielsweise einer Speicherprogrammierbaren Steuerung vergrößert die Komplexität eines Zeitautomaten erheblich.

Die Zeitfolge der Symbole eines Eingabeworts eines Zeitautomaten nach Alur/Dill wird in [AD94] als streng monoton und in der Zeit fortschreitend gefordert. In [Tri99] werden Zeitfolgen und Zeitautomaten untersucht, deren Zeitfortschritt nicht gesichert ist. Konvergiert beispielsweise die Zeitfolge der Symbole eines unendlichen Eingabeworts nicht gegen unendlich sondern gegen einen Zeitpunkt, so treten in endlichem Zeitraum unendlich viele Symbole im Eingabewort auf. Zeitautomaten können in einem solchen Fall eventuell eine unendliche Zahl von Zustandsübergängen ausführen und damit in der Zeit nicht weiter voranschreiten. Als Analogon zu *Deadlocks* werden deartige Zustandsfolgen auch als *Timelocks* [Tri99] bezeichnet und Zeitautomaten, die Timelocks besitzen können, heißen zeno. Derartige Zeitautomaten können auf realen Systemen nicht nachgebildet werden, da jedes System eine minimale Reaktionszeit besitzt.

2.3.3 Zeitautomaten mit Reaktionszeit

In [WDR04] wird eine veränderte Semantik für Zeitautomaten untersucht. Grundlage dieser Semantik ist die Annahme, dass bei Zeitautomaten Transitionen „so früh wie möglich“ („As Soon As Possible.“ - ASAP), in realen Implementierungen aber mit einer gewissen Verzögerung und deshalb nur „fast so früh wie möglich“ ausgeführt werden. Die in [WDR04] untersuchte Semantik wird deshalb auch als „Almost As Soon As Possible“-Semantik (AASAP) bezeichnet und führt eine Reaktionszeit für Transitionen ein. Zeitautomaten mit AASAP-Semantik können mehr Zustände erreichen, als Zeitautomaten ohne AASAP-Semantik. Es wird deshalb vorgeschlagen, für eine Implementierung mit Reaktionszeit zu prüfen, ob das Zeitautomatenmodell unerwünschte Zustände, die mit ASAP-Semantik ausgeschlossen sind, mit der AASAP-Semantik ebenfalls nicht erreicht werden können. In [WDMR04] wird ferner untersucht, ob für einen gegebenen Zeitautomaten und eine Menge unerwünschter Zustände eine Reaktionszeit gefunden werden kann, so dass der Zeitautomat die unerwünschten Zustände unter AASAP-Semantik nicht erreichen kann.

Bewertung

Die Arbeiten basieren auf der Feststellung, dass die Implementierung eines Zeitautomaten von dessen theoretischem Verhalten abweicht, wobei von der Implementierung auch Zustände erreicht werden können, die im theoretischen Fall unerreichbar sind. Es werden zwei Ansätze zur Lösung dieses Problems vorgeschlagen: Zum einen beschreibt der entscheidbare Begriff der „Implementierbarkeit“, wie gutmütig sich ein gegebener Zeitautomat unter der Annahme von Reaktionszeiten verhält. Ferner wird eine Möglichkeit vorgestellt, aus einem gegebenen Zeitautomaten einen Zeitautomaten zu bilden, der die AASAP-Semantik einer Implementierung besitzt. Dies ermöglicht die Verifikation eines Modells unter Implementierungsbedingungen durch Modellprüfer.

Die AASAP-Semantik bezieht sich auf ereignisorientierte Zeitsysteme, nicht jedoch auf zyklische Zeitsysteme wie SPS. Ferner werden nur obere Zeitschranken für die Reaktionszeit betrachtet. Für die Korrektheit einer Implementierung ist die minimale Reaktionszeit allerdings ebenfalls wichtig, da sie den Zeitfortschritt der Implementierung sichert und zur Unereichbarkeit von Zuständen in der Implementierung führen kann. Schließlich wird in [WDR04, WDMR04] nicht beschrieben, wie die Korrektheit einer Implementierung abgesichert werden könnte, was gerade Ziel dieser Arbeit ist.

2.4 Transformation von Zeitsystemen

In [Wil99] und [MW99] wird eine Möglichkeit beschrieben, SPS-Programme mit Zeitbedingungen zu prüfen. Als Sprache für SPS-Programme wird eine Teilmenge des *Instruction List*-

Formats (IL) unterstützt. Ein SPS-Programm wird in einen Zeitautomaten transformiert, wobei ein direkter Ansatz verfolgt wird, der die IL-Befehle eines Programms in Zustände und Zustandsübergänge des resultierenden Zeitautomaten umwandelt. Der entstandene Zeitautomat kann anschließend mit Modellprüfern untersucht werden.

Bewertung

Ziel dieser Arbeiten ist die Untersuchung des Verhaltens eines SPS-Programms. Dazu werden alle Schritte des Programms voll ausmodelliert.

Aufgrund dieser direkten Abbildung eines IL-Programms steigt die Zustandsanzahl der gewonnenen Zeitautomaten trotz starker Einschränkungen wie die Beschränkung auf Zahlen zwischen -10 und 10 schnell an. Damit ist das gewählte Verfahren unpraktikabel. Zudem genügt aus Sicht der Translation Validation bereits der Nachweis der Gleichheit des extern beobachtbaren Verhaltens von Steuerung und Modell, was es unnötig macht, alle Zwischenzustände eines SPS-Programmes zu modellieren.

2.5 Analyse von Zeitsystemen

Dieser Abschnitt bezieht sich auf die Ermittlung der Zeitzustandsgraphen, also der Punkte (g) und (h) in Abbildung 1.3.

Zeitsysteme besitzen eine Menge diskreter Zustände, beispielsweise die Markierungen eines Zeitpetrinetzes, die Variablenwerte einer Speicherprogrammierbaren Steuerung oder die Zustände eines Zeitautomaten. Darüber hinaus besitzen Zeitsysteme eine oder mehrere Uhren, die unabhängig voranschreiten. Das Verhalten eines Zeitsystems hängt sowohl vom diskreten Zustand ab, in dem sich das Zeitsystem befindet, als auch vom Stand seiner Uhren.

Die Kombination eines diskreten Zustands mit dem Stand aller Uhren bildet einen Zeitzustand. Besitzt ein Zeitsystem reellwertige Uhren, dann ist die Menge der Zeitzustände des Systems überabzählbar unendlich, selbst wenn die Menge diskreter Zustände endlich ist. Jeder Zeitzustand eines Zeitsystems besitzt einen oder mehrere Nachfolger. Die Menge aller Zeitzustände eines Zeitsystems sowie die Nachfolgerrelation bildet einen Zeitzustandsgraphen, der die Semantik des Zeitsystems definiert.

Um die Zeitzustände praktisch aufzuzählen, ist es notwendig, Zeitzustände, die unter bestimmten Bedingungen als „äquivalent“ angesehen werden können, zusammenzufassen. Die so gebildeten Äquivalenzklassen werden als symbolische Zustände bezeichnet. Es existieren verschiedene Techniken, die Zeitzustände eines Zeitsystems durch symbolische Zustände zu repräsentieren, einen Überblick gibt [Yov98].

Während für die Modellprüfung eine symbolische Repräsentation der Menge der erreichbaren Zeitzustände meist ausreichend ist, wird für die in dieser Arbeit angestrebte Translation

Validation eine symbolische Repräsentation des Zeitzustandsgraphen benötigt. In Frage kommende Datenstrukturen müssen deshalb die Darstellung von Zeitzustandsgraphen erlauben.

Für n reellwertige Uhren kann man sich die Menge möglicher Uhrenstände auch als n-dimensionalen Raum vorstellen, in dem jede Uhr durch eine eigene Dimension repräsentiert wird. Die Dimension dieses Raumes entspricht der Anzahl der Uhren eines Zeitsystems, weshalb das starke kombinatorische Anwachsen der Anzahl möglicher Uhrenstände mit jeder zusätzlichen Uhr eine besondere Herausforderung für symbolische Zustände darstellt. Das Problem der Explosion des Raumvolumens mit jeder Dimension wurde bereits 1961 von Richard Bellman treffend als „Fluch der Dimensionalität“ (curse of dimensionality [Bel61]) bezeichnet. Die praktische Effizienz einer Analyse hängt maßgeblich von den symbolischen Zuständen und den dazu verwendeten Datenstrukturen ab.

2.5.1 Analysealgorithmen

Um zu prüfen, ob ein Menge von Zeitzuständen von einem Zeitautomaten (nicht) erreicht werden kann, genügt es aus Sicht eines Modellprüfers oft, ausgehend von der Menge der Zeitzustände deren Vorgängerzustände aufzuzählen bis ein Startzustand des Zeitautomaten erreicht wurde oder alle Vorgängerzustände aufgezählt wurden. Der Richtung der Analyse entsprechend wird dieses Vorgehen als Rückwärtsanalyse bezeichnet [Yov98]. Die Rückwärtsanalyse terminiert stets, zählt in der Regel aber nicht alle Zeitzustände eines Zeitautomaten auf.

Zur Darstellung aller erreichbaren Zeitzustände eines Zeitautomaten kann die Vorwärtsanalyse verwendet werden. Bei der Vorwärtsanalyse werden ausgehend von den Startzeitzuständen alle Nachfolgezustände aufgezählt. Damit die Menge symbolischer Zustände endlich ist und die Vorwärtsanalyse terminiert, existieren Operatoren, die symbolische Zustände oberhalb der größten in Zeitbedingungen vorkommenden Konstanten zusammenfassen [MP99, BBFL03]. Die Anwendung dieser Operatoren kann jedoch zu Fehlern in der Vorwärtsanalyse von Zeitautomaten mit diagonalen Beschränkungen der Form $u_1 - u_2 \diamond c$ mit $\diamond \in \{<, \leq\}$ führen [Bou03]. In [Bou03] wird die Korrektheit der Vorwärtsanalyse jedoch für Automaten bewiesen, die ausschließlich orthogonale Beschränkungen der Form $u \diamond c, \diamond \in \{<, \leq\}$ besitzen.

In [LT01] wird ein weiterer Weg zur Analyse von Zeitautomaten vorgeschlagen. Sogenannte *Splittrees* dienen dabei der Darstellung von Zeitzuständen und *Partition Refinement* wird als Algorithmus zur Aufzählung der Zeitzustände verwendet. Splittrees sind eine Datenstruktur zur räumlichen Teilung (Partitionierung) der Zeitzustände eines Zeitautomaten durch orthogonale und diagonale Beschränkungen. Beim Partition Refinement werden die Splittrees solange verfeinert, bis die entstandenen Partitionen durch Operationen des Zeitautomaten nicht weiter getrennt werden können. Beim praktischen Vergleich der Splittrees mit Zonen zeigte sich jedoch, dass die Vorwärtsanalyse effizienter ist [LT01].

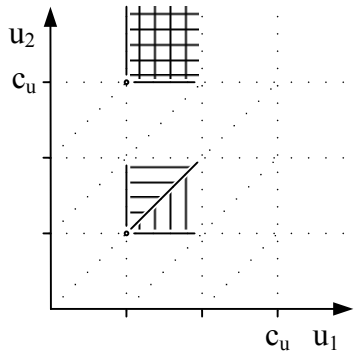


Abbildung 2.1: Beispiele für Regionen.

2.5.2 Regionen

Regionen sind einfache symbolische Zustände, die in [AD94] zur theoretischen Analyse von Zeitautomaten eingeführt und verwendet wurden. Regionen nutzen aus, dass die Uhren eines Zeitautomaten und vieler anderer Zeitsysteme nur mit natürlichen Konstanten verglichen werden. Werden gebrochene Zahlen verwendet, so können diese durch Multiplikation mit einer geeigneten Konstante in natürliche Zahlen umgewandelt werden.

Die Uhrenstände eines Zeitsystems können in einen integralen Teil und einen fraktionalen Teil aufgetrennt werden. Da ein Zeitautomat Uhren nur mit natürlichen Konstanten vergleicht, beschreibt der integrale Teil, welche Schaltbedingungen eines Zeitautomaten gelten, während der fraktionale Teil bei mehreren Uhren darüber entscheidet, welche Uhr zuerst ihren integralen Teil wechselt.

Zwei Zeitzustände sind in diesem Sinne deshalb gleich, wenn die integralen Uhrenstände aller Uhren jeweils gleich sind oder wenn sie größer als die größte in einer Schaltbedingung des Zeitautomaten vorkommende Konstante sind. Durch die letztgenannte Bedingung wird sichergestellt, dass die Anzahl an Regionen endlich bleibt. Ferner muss gelten, dass die fraktionalen Uhrenstände aller Uhren jeweils gleich geordnet sind. Eine exakte Definition findet man beispielsweise in [AD94].

Beispiel 2.1. *In Abbildung 2.1 ist ein Uhrenstandsdiagramm dargestellt. Die möglichen Uhrenstände zweier reellwertige Uhren u_1 und u_2 wurden dazu gegeneinander abgetragen. Die gestrichelten Linien dienen der Orientierung, c_u stellt die größte Konstante dar. In der Abbildung sind einige Regionen schraffiert bzw. schwarz markiert. Das betrifft zwei Eckpunkte, fünf Linien und drei Flächen, zusammen 10 Regionen. Insgesamt enthält das Diagramm* 16 *Eckpunkte,* 41 *Linien und* 25 *Flächen, zusammen* 82 *Regionen.*

Bewertung

Die Anzahl der Regionen eines Zeitautomaten ist endlich und repräsentiert den Zeitzustandsraum eines Zeitautomaten. Aufgrund ihrer einfachen Struktur eignen sich Regionen besonders für theoretische Beweise.

Die Anzahl an Regionen hängt von der größten Konstante eines Zeitautomaten ab und beträgt laut [AD94] $n!4^n(c_u + 1)^n$, wobei n die Anzahl der Uhren ist. Damit ist die Anzahl an Regionen für praktische Analysen zu groß.

2.5.3 Differenzgrenz-Matrizen

Eine *Differenzgrenz-Matrix* eines Zeitautomaten mit n Uhren ist eine quadratische $(n + 1)$-Matrix. Die Elemente der Matrix definieren für jede Uhr u eine orthogonale Uhrbeschränkung $u < (\leq)c$, wobei c eine Konstante darstellt, sowie für jedes Paar von Uhren u_1, u_2 eine diagonale Uhrbeschränkung $u_1 - u_2 < (\leq)c$. Geometrisch interpretiert, definiert eine Differenzgrenz-Matrix einen konvexen Hyperpolyeder, der eine Menge von Regionen einschließt.

Ein und derselbe konvexe Hyperpolyeder kann durch unterschiedliche Differenzgrenz-Matrizen dargestellt werden. Die Normalisierung, d.h. die Ermittlung einer eindeutigen Form einer gegebenen Differenzgrenz-Matrix, benötigt $\Theta(n^3)$, wobei n die Anzahl der Uhren ist.

Bewertung

Differenzgrenz-Matrizen sind die bei Modellprüfern für Zeitsysteme am weitesten verbreiteten symbolischen Zustände. Differenzgrenz-Matrizen sind vergleichsweise einfach zu verstehen und zu implementieren. Ihr wesentlicher Vorteil ist die Unabhängigkeit von Konstanten eines Zeitsystems, wodurch auch komplexere Systeme mit einer vergleichsweise geringen Anzahl von Differenzgrenz-Matrizen darstellbar ist. Alle Operationen, die zur Analyse eines Zeitsystems notwendig sind, lassen sich zudem durch Differenzgrenz-Matrizen einheitlich und effizient darstellen.

Zu den Nachteilen von Differenzgrenz-Matrizen insbesondere beim Abspeichern besuchter Zustände, gehört der konstant hohe Speicherbedarf einer Differenzgrenz-Matrix sowie die für viele Operationen notwendige Normalisierung. Aus diesen Gründen eignen sich Differenzgrenz-Matrizen nur bedingt als Indexstruktur für bereits besuchte Zustände [Ben01].

2.5.4 Weitere Ansätze

Während zur Aufzählung der Zustände von Zeitautomaten meist Differenzgrenz-Matrizen verwendet werden, wurden insbesondere zur Speicherung bereits besuchter Zustände verschiedene andere Ansätze verfolgt. Beispiele sind die graphische Repräsentation von Differenz-

grenz-Matrizen mit Hilfe von Gitterpunkten [BM00] sowie die Minimierung und kompakte Speicherung von Differenzgrenz-Matrizen [LLPY97, Ben01].

Angeregt durch die Erfolge bei der Modellprüfung diskreter Systeme mit Hilfe binärer Entscheidungsdiagramme (binary decision diagrams - BDDs) wurden ähnliche Techniken auch für Zeitsysteme adaptiert. Das betraf zuerst die Diskretisierung von Zeitzuständen und deren Speicherung mit BDDs. Beispielsweise wird in [BMPY97] die Erweiterung des Modellprüfers KRONOS um BDDs beschrieben. In weiteren Arbeiten wurde die Idee der binärer Entscheidungsdiagramme weiterentwickelt und auf Zahlen (Numerical Decision Diagrams - NDDs [ABK+97]) beziehungsweise Intervalle (Interval Diagrams [Str99]) übertragen. Andere Vorschläge basieren auf Differenzen von Uhrenständen (Clock Difference Diagrams - CDDs [BLP+99]) oder Zahlen (Difference Decision Diagrams - DDDs [MLAH99]). Ähnliche Ansätze verfolgen auch Data Decision Diagrams [Wan00, Wan01].

Bewertung

Entscheidungsdiagramme sowie kompakt dargestellte Differenzgrenz-Matrizen können den Platzbedarf zur Speicherung besuchter Zustände wesentlich senken, beispielsweise berichtet [BLP+99] von 46% bis 99% Speicherplatzersparnis.

Werden die Zustände eines Zeitautomaten mit Hilfe von Differenzgrenz-Matrizen aufgezählt, müssen diese in die zum Speichern der Zustände verwendeten Datenstruktur umgewandelt werden. Die Konvertierung zwischen Differenzgrenz-Matrizen und Speicherdatenstruktur ist meist sehr aufwändig und wiegt eine mögliche Zeitersparnis wieder auf. Das führt dazu, dass für viele Datenstrukturen keine Beschleunigung, sondern oft sogar eine Verlangsamung der Analyse beobachtet werden konnte.

Insbesondere Entscheidungsdiagramme sind teilweise sehr rechenaufwändig und haben dabei einen sehr unterschiedlichen Platzbedarf, der beispielsweise von der Variablenordnung abhängt. Eine wesentliche Beschleunigung der Analyse erscheint nur dann möglich, wenn Entscheidungsdiagramme auch zur Aufzählung der Zeitzustände verwendet werden. In diesem Fall wird von einer erheblichen Beschleunigung der Analyse von Zeitautomaten berichtet (etwa in [MLAH99] sowie [BN03]). Beide Arbeiten gehen jedoch auf die Adaption der Vorwärtsanalyse sowie der notwendigen Operationen auf Entscheidungsdiagramme nicht näher ein, was das Nachvollziehen der Ergebnisse erschwert. Eine ausführliche Analyse von BDD-ähnlichen Datenstrukturen in [Wan04] kommt ebenfalls zum Ergebnis, dass der Einsatz von BDDs bei der Vorwärtsanalyse langsamer als der Einsatz kompakter Differenzgrenz-Matrizen ist.

Für die in dieser Arbeit angestrebte Translation Validation zweier Zeitsyteme können Entscheidungsdiagramme nicht eingesetzt werden, da sie nicht zur Darstellung von Zeitzustandsgraphen geeignet sind.

2.6 Vergleich von Zeitsystemen

Dieser Abschnitt bezieht sich auf die Punkte (f) und (i) in Abbildung 1.3, den Korrektheitsbegriff und Korrektheitsnachweis.

Zwei Zeitautomaten sind gleich, wenn sie die selbe zeitbewertete Sprache erkennen. Nach Alur und Dill ist diese Frage im allgemeinen Fall für nichtdeterministische Zeitautomaten unentscheidbar [AD94]. Ebenso unentscheidbar ist im allgemeinen Fall, ob die von einem Zeitautomaten erkannte Sprache Teilmenge einer von einem anderen Zeitautomaten erkannten Sprache ist (Language Inclusion). Für Unterklassen von Zeitautomaten, darunter deterministische Zeitautomaten, sind die genannten Probleme theoretisch entscheidbar [AD94].

Strengere Begriffe von Ähnlichkeit sind ebenso entscheidbar. Ein Ähnlichkeitsbegriff ist dabei in dem Sinne strenger als ein anderer, als dass aus einem strengeren Ähnlichkeitsbegriff der allgemeinere folgt, aber nicht umgekehrt. Entscheidbar sind Bisimulation und Simulation auf den Zeitzuständen zweier Zeitautomaten [Yi90, Cer93, ACH94, WL97].

Bewertung

Die Entscheidbarkeit der Bisimulation für Zeitautomaten wurde von Karlis Cerans in [Cer93] theoretisch auf Regionen nachgewiesen. Der Nachweis basiert auf der Bildung des Produktzeitautomaten, der die Zustände, Uhren und Transitionen zweier Zeitautomaten kombiniert. Ein weiterer Ansatz [LY02] definiert eine Algebra für Zeitautomaten, die es erlaubt, symbolisch festzustellen, ob für zwei Automaten eine Bisimulation besteht. Eine praktische Implementierung dieses Ansatzes ist nicht bekannt.

Der in [Cer93] vorgeschlagene Algorithmus ist aus praktischer Sicht kaum anwendbar, da die Anzahl der Regionen von der Größe der Konstanten der Zeitautomaten abhängt und zu schnell wächst. In [WL97] wird deshalb ein Verfahren vorgeschlagen, das Differenzgrenz-Matrizen zur Darstellung der Zustände des Produktzeitautomaten verwendet und unabhängig von der tatsächlichen Größe von Konstanten ist („scaling-invariant“). Während der Nachweis einer Simulation für einfache Zeitautomaten mit diesem Verfahren praktisch durchführbar wird, ist die Konstruktion des Produktzeitautomaten für komplexere Zeitautomaten oft nicht mehr möglich, da die Anzahl symbolischer Zustände des Produktzeitautomaten weit über dem Produkt der symbolischen Zustände der verglichenen Zeitautomaten liegen kann.

Beispiel 2.2. *Für zwei Zeitautomaten mit je einer Uhr, einem Zustand und einer Transition Abbildung 2.2 stellt die symbolischen Zustände eines Produktzeitautomaten dar, der aus zwei sehr einfachen Zeitautomaten gebildet wurde. Jeder dieser Zeitautomaten enthält jeweils nur einen Zustand z, eine Uhr u und eine Transition, die die Uhr bei Erreichen von einer bzw. 100 Zeiteinheiten zurücksetzt. Beide Zeitautomaten besitzen lediglich drei symbolische Zustände, hier vereinfacht dargestellt:* $\langle z, u = 0\rangle$, $\langle z, 0 < u < 1(100)\rangle$ *und* $\langle z, u = 1(100)\rangle$.

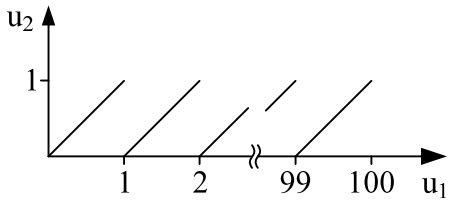


Abbildung 2.2: Symbolische Zustände eines Produktzeitautomaten.

Der Produktzeitautomat besitzt dem entgegen mindestens 100 symbolische Zustände, wobei die Anzahl der Zustände durch Ändern der Größe der Konstanten beliebig erhöht werden kann. Insofern ist der Ansatz von [WL97] auch nur bedingt „scaling-invariant“, nämlich nur genau dann, wenn die Konstanten beider Automaten mit dem gleichen Faktor skaliert werden.

2.7 Zusammenfassung

Die vorliegende Arbeit überschneidet viele aktuelle Forschungsgebiete. Das betrifft die Korrektheit von Übersetzern sowie die Translation Validation. Die bekannten Techniken wurden unseres Wissens vor allem auf den Korrektheitsnachweis von Programmen, weniger auf die Übersetzung von Modellen beispielsweise bei der modellgetriebenen Architektur angewendet. Beim Nachweis der Korrektheit von Übersetzern für Modelle werden in dieser Arbeit Techniken eingesetzt, die auch zur Prüfung von Modellen verwendet werden. Die bekannten, in Frage kommenden Modelle eignen sich gut zur Modellierung von Steuerungen oder zeitbewerteten Systemen, ihre vereinfachten Annahmen führen aber zu Problemen bei der Implementierung. Um die Übersetzung eines Modells zu validieren, werden in dieser Arbeit die Modellzustände aufgezählt. Dabei kann auf zahlreiche Arbeiten aus dem Bereich der Modellprüfung zurückgegriffen werden. Die Eigenschaften realer Zeitsysteme sowie die Notwendigkeit zur Aufzählung von Zustandsgraphen erfordern jedoch die Weiterentwicklung bekannter Techniken und Datenstrukturen.

3 Grundlagen

In diesem Kapitel werden die für das Verständnis der weiteren Arbeit notwendigen Grundlagen gelegt. Dazu werden im ersten Abschnitt die Eigenschaften und Annahmen dargelegt, die für Speicherprogrammierbare Steuerungen sowie den darauf implementierten Modellen gelten. Der zweite Abschnitt beschäftigt sich mit der symbolischen Repräsentation von Zeit.

3.1 Eigenschaften Speicherprogrammierbarer Steuerungen und verwandter Systeme

Modelle können den Umgang mit Zeit unterschiedlich abbilden. Unterschiede können beispielsweise in den *Eigenschaften der Uhren* (kontinuierlich, getaktet, synchron, asynchron), den möglichen *Zeitbedingungen* (Vergleiche mit Konstanten, Vergleiche von Uhren), den erlaubten *Operationen* (Setzen von Uhren auf 0 oder konstante Werte, Anhalten von Uhren) oder den *Rahmenbedingungen* (Reaktionszeit, Arbeitsweise) bestehen.

Aufgrund dieser Varianten ist bereits bei Zeitautomaten eine Vielfalt verschiedener Modelle mit unterschiedlichen Eigenschaften entstanden. Diese Eigenschaften haben Auswirkungen auf die verwendeten Algorithmen und Datenstrukturen.

In dieser Arbeit wird die Implementierung von Modellen mit Zeitbedingungen auf Speicherprogrammierbaren Steuerungen untersucht. Da die Eigenschaften von Speicherprogrammierbaren Steuerungen maßgebend für das Verhalten der Implementierung sind und nicht verändert werden können, ist es notwendig, diese Eigenschaften auf das Modell zu übertragen. Erfüllt das Modell eine dieser Eigenschaften nicht, kann es entweder nicht auf einer Speicherprogrammierbaren Steuerung implementiert werden oder muss angepasst werden.

Im Folgenden werden die Annahmen für die Implementierung eines Modells auf einer SPS formuliert, die sich aus den Vorgaben und Anforderungen dieser Arbeit ergeben. Diese Annahmen sind auf TNCES und SPS-Steuerprogramme übertragbar. Im ersten Unterabschnitt werden Annahmen über die Behandlung von Zeit, im zweiten über die Kommunikation der SPS mit ihrer Umwelt und im dritten über die Zustände und Zustandsübergänge der Speicherprogrammierbaren Steuerung dargelegt. Daran schließt sich ein vierter Unterabschnitt an, der Annahmen und Bemerkungen zur Implementierung von Modellen und Behandlung von Uhren enthählt.

3.1.1 Zeit

Zeit wird in dieser Arbeit als eine durch eine Uhr messbare kontinuierliche physikalische Größe verstanden. Der von der Uhr zu einem Zeitpunkt angezeigte Wert wird als Stand der Uhr, kurz *Uhrenstand* bezeichnet. Reale Uhren können Zeit nur mit *endlicher Genauigkeit* messen und sich der tatsächlichen Zeit nur annähern. Reale Uhrenstände können deshalb nur *rationale Werte* annehmen. Des Weiteren können Uhren eines Zeitsystems mit einer Konstante verglichen sowie auf den Uhrenstand 0 zurückgesetzt werden.

Folgende Annahmen leiten sich für die Uhren eines Zeitsystems aus diesen Überlegungen ab.

Endlichkeit Es werden nur Zeitsysteme betrachtet, die eine beliebige aber feste Anzahl von Uhren besitzen. Im Zeitsystem können Uhren weder erzeugt noch vernichtet werden. Beim Start des Zeitsystems sei der Uhrenstand aller Uhren 0.

Genauigkeit Um reale Zeitsysteme analysieren zu können, wird angenommen, dass die Genauigkeit der Uhren eines Zeitsystems endlich ist und eine kleinste messbare Zeiteinheit existiert. Die kleinste messbare Zeiteinheit eines Zeitsystems muss unter der im Weiteren definierten Zykluszeit des Zeitsystems liegen.

Operationen Zulässige Operationen, die ein Zeitsystem auf seinen Uhren ausführen kann, sind der Zeitvorlauf, d.h. das zeitliche Voranschreiten aller Uhren, sowie das Zurücksetzen einer oder mehrerer Uhren auf 0. Der Stand jeder Uhr kann zudem vom Zeitsystem mit rationalen Konstanten verglichen werden. Weitere Operationen werden nicht betrachtet.

3.1.2 Ein- und Ausgabe

Zur Steuerung eines technologischen Prozesses muss eine SPS mit ihrer Umwelt kommunizieren. Diese Kommunikation erfolgt durch Signale, die entweder durch die SPS selbst (Ausgang) oder die Umwelt der SPS (Eingang) aktiviert werden. Dabei gelten folgende Annahmen für die Kommunikation der SPS mit ihrer Umwelt:

Signale Ein Signal dient der Kommunikation der SPS mit ihrer Umwelt und wird mit σ bezeichnet. Eine Menge von Signalen wird mit Σ bezeichnet. Zu einem Zeitpunkt kann ein Signal entweder aktiv oder inaktiv sein. Ein Signal kann entweder ausschließlich durch die SPS oder ausschließlich durch dessen Umwelt aktiviert oder deaktiviert werden. Wird ein Signal durch die SPS aktiviert bzw. deaktiviert, so wird das Signal auch als *Ausgang* bezeichnet, andernfalls als *Eingang*. In einer realen SPS kann ein Signal in einem endlichen Zeitraum nur endlich oft aktiviert oder deaktiviert werden.

Eingänge Ein *Eingang* spiegelt das Auftreten eines Ereignisses in der Umwelt wider. Der Eingang ist genau dann aktiv, wenn das Ereignis in der Umwelt auftritt. Ein Ereignis kann zu einem Zeitpunkt oder über einen Zeitraum hinweg auftreten. Die Menge aller

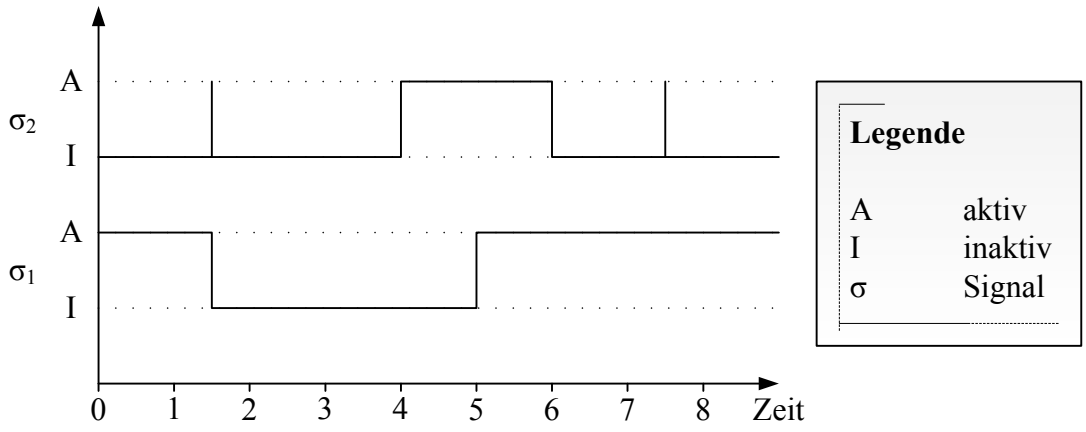


Abbildung 3.1: Pulsdiagramm zweier Signale σ_1 und σ_2.

Eingänge einer SPS wird mit $\Sigma^e := \{\sigma_1^e, \ldots, \sigma_n^e\}$ bezeichnet. Eine etwaige Verzögerung zwischen dem Auftreten eines Ereignisses in der Umwelt der SPS und dem Aktivieren des entsprechenden Eingangs kann durch die SPS nicht beeinflusst werden und wird nicht weiter betrachtet. Die zu einem Zeitpunkt aktiven Eingänge $\Sigma^{e'} \subseteq \Sigma^e$ definieren eine *Eingangsbelegung* der SPS.

Ausgänge Eine SPS besitzt eine endliche Menge Σ^a von *Ausgängen* σ^a, die durch die SPS aktiviert werden können. Die Ausgänge einer SPS sind stets nur zum Zeitpunkt ihrer Aktivierung aktiv und können nicht über Zeiträume aktiv sein. Durch das Aktivieren eines Ausgangs kann die SPS Prozesse in der Umwelt anstossen. Die etwaige Verzögerung zwischen Aktivierung eines Ausgangs und dem tatsächlichen Beginn eines Prozesses wird nicht weiter betrachtet. Ist $\Sigma^a := \{\sigma_1^a, \ldots, \sigma_m^a\}$ eine Menge von Ausgängen einer SPS, dann bildet die Menge der aktiven Ausgänge $\Sigma^{a'} \subseteq \Sigma^a$ eine *Ausgangsbelegung* der SPS.

Beispiel 3.1. *Signale können in Pulsdiagrammen veranschaulicht werden. Abbildung 3.1 stellt in einem* Pulsdiagramm *zwei Prozesse dar. Auf der horizontalen Achse des Diagramms ist die Zeit abgetragen. An der vertikalen Achse sind zwei Signale σ_1 und σ_2 dargestellt, die jeweils entweder aktiv („A“) oder inaktiv („I“) sein können. In den Zeiträumen, in denen ein Signal aktiv ist, wird die Linie „A“ durchgezogen, ansonsten gepunktet dargestellt, entsprechendes gilt für die Linie „I“. Zur besseren Veranschaulichung sind die Linien „A“ und „I“ zum Zeitpunkt der Aktivierung oder Deaktivierung eines Signals verbunden.*

Anmerkung 3.2. *Die betrachteten Modelle können nicht direkt mit kontinuierlichen Eingaben wie beispielsweise einer Temperatur umgehen. Modelle, die kontinuierliche Eingaben zulassen, werden als hybride Modelle bezeichnet. Es besteht für die betrachteten Modelle jedoch die Möglichkeit, kontinuierliche Umweltgrößen zu diskretisieren und als Eingabe nutzbar zu machen. Ein Beispiel dafür könnte die Einteilung einer Temperatur in Abschnitte und deren Repräsentation mit mehreren Signalen sein.*

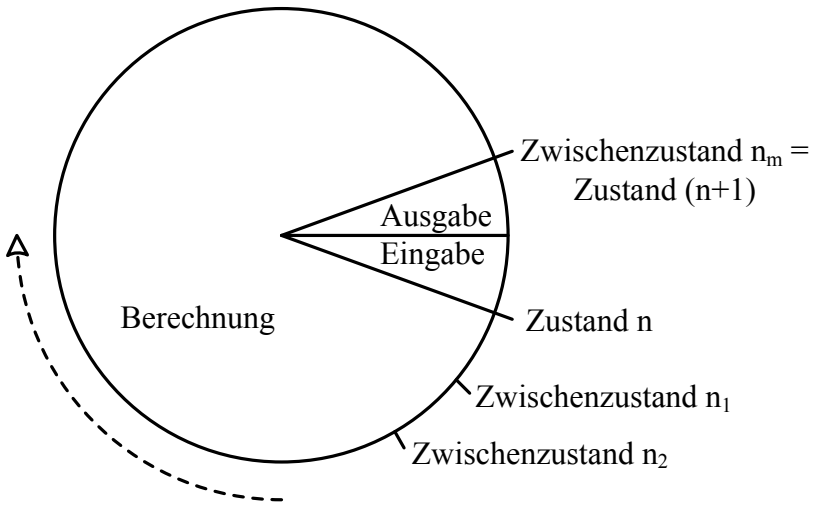


Abbildung 3.2: Zyklus einer Speicherprogrammierbaren Steuerung.

3.1.3 Arbeitsweise Speicherprogrammierbarer Steuerungen

Die folgenden Annahmen basieren auf der zyklischen Arbeitsweise Speicherprogrammierbarer Steuerungen, lassen sich aber auf andere zyklisch arbeitende Systeme übertragen. Entscheidendes Merkmal solcher Systeme ist die regelmäßige Abfrage der Eingänge und Uhren des Systems. Im Gegensatz dazu reagieren ereignisgesteuerte Systeme erst beim Auftreten eines Signals.

Die betrachteten Steuerungen besitzen eine endliche Anzahl von Uhren und Zuständen und wechseln einen Zustand aufgrund einer Eingangsbelegung, eines Uhrenstandes oder aufgrund von beidem. Da reale Steuerungen nicht unendlich schnell sein können, benötigt jeder Zustandsübergang Zeit.

Folgende Annahmen leiten sich aus diesen Überlegungen ab:

Zustände Ein *Zustand* ist die Belegung der Variablen eines SPS-Steuerprogramms. Da die Menge solcher Variablenbelegungen endlich ist, ist die Menge der Zustände des Steuerprogramms ebenfalls endlich.

Arbeitsweise Es wird davon ausgegangen, dass eine SPS in *Zyklen* arbeitet, die unmittelbar aufeinander folgen. Ein *Zyklus* ist in Abbildung 3.2 dargestellt und umfasst eine *Eingabephase*, eine *Berechnungsphase* und eine *Ausgabephase*.

In der *Eingabephase* befindet sich das Steuerprogramm in einem Zustand n_0 und die SPS ermittelt die Eingangsbelegung sowie den Uhrenstand.

Ausgehend vom Zustand n, der Eingangsbelegung und dem Uhrenstand ermittelt die SPS in der anschließenden *Berechnungsphase* einen Folgezustand $(n+1)$ sowie die Ausgangsbelegung und die Menge der zurückzusetzenden Uhren. In der Berechnungsphase

kann das Steuerprogramm eine endliche Anzahl von Zwischenzuständen n_i durchlaufen, die von außen nicht beobachtbar sind.

In der anschließenden *Ausgabephase* aktiviert die SPS die Ausgänge der berechneten Ausgangsbelegung und beginnt unmittelbar mit der Eingabephase des nächsten Zyklus.

Zykluszeit Die Ausführung eines Zyklus auf einer realen SPS benötigt die positive *Zykluszeit* τ, um welche die Uhren der SPS während des Zyklus vorlaufen. Die exakte Zykluszeit hängt von physikalischen Einflüssen ab und kann für einen einzelnen Zyklus nicht exakt vorhergesagt oder bestimmt werden. Die Zykluszeit bewegt sich nur in Grenzen, die als *minimale Zykluszeit* $\delta \in \mathbb{Q}_{>0}$ bzw. *maximale Zykluszeit* $\Delta \in \mathbb{Q}_{>0}$ bezeichnet werden, wobei $\delta < \tau < \Delta$ gilt.

Anmerkung 3.3. *Es besteht ein Unterschied zwischen der Reaktionszeit und Zykluszeit einer SPS. Da ein Ereignis theoretisch direkt am Ende der Eingabephase eines Zyklus auftreten kann, dauert es bis zum nächsten Zyklus, bis die SPS auf das Ereignis reagieren kann. Damit entspricht die Reaktionszeit maximal dem Doppelten der Zykluszeit. Diese Eigenschaft gilt erstaunlicherweise auch für Systeme mit* ereignisorientierter Arbeitsweise*, da ein Ereignis direkt auf ein zweites Folgen kann und das zweite Ereignis erst nach dem ersten bearbeitet werden kann.*

3.1.4 Implementierung

Es bestehen mehrere Möglichkeiten, eine SPS zu programmieren, unter anderem grafische Darstellungen wie Ladder Diagrams, assemblerähnliches Instruction List sowie PASCAL-ähnliches Structured Text. Ein Beispielprogramm einer stark vereinfachten Ampelsteuerung in Structured Text ist in Algorithmus 3.1 dargestellt.

Beispiel 3.4 (Algorithmus 3.1)**.** *Es gibt in dem Programm drei verschiedene Variablendeklarationen:* `VAR_IN` *definiert zwei binäre Eingangssignale (die nur der Anschaulichkeit dienen) und* `VAR_OUT` *zwei binäre Ausgangssignale* `Gruen_0` *und* `Gruen_1`*, die jeweils die Grünphasen zweier Ampeln steuern. Zudem definiert* `VAR` *zwei Uhren vom Typ Timer(*`TON`*). Das Programm soll alle 30 Sekunden die Grünphase der Ampeln umschalten. Gelbphasen oder andere Zwischenzustände wurden weggelassen.*

Mit dem Ausdruck `Uhr_0(IN:=1, T#24h).ET` *wird der Uhrenstand von* `Uhr_0` *ermittelt. Entspricht dieser Uhrenstand genau einer Minute, wird* `Uhr_0` *auf den Uhrenstand* 0 *mit der Anweisung* `Uhr_0(IN:=0, T#24h);` *zurückgesetzt. Ist der Uhrenstand von* `Uhr_0` *kleiner 30 Sekunden, wird in der folgenden Anweisung der Ausgang* `Gruen_0` *auf* `TRUE` *gesetzt, andernfalls auf* `FALSE`*. Entsprechend wird mit* `Uhr_1` *und* `Gruen_1` *verfahren, wobei* `Gruen_1` *nur dann auf* `TRUE` *gesetzt wird, wenn der Uhrenstand von* `Uhr_1` *größer oder gleich 30 Sekunden ist.*

```
1  FUNCTION_BLOCK Beispiel
2    VAR_IN     Ein_0, Ein_1      : BOOL; END_VAR
3    VAR_OUT    Gruen_0, Gruen_1  : BOOL; END_VAR
4    (* TON ist ein interner Timer *)
5    VAR        Uhr_0, Uhr_1      : TON;  END_VAR
6
7    (* Zeigt Uhr_0 genau eine Minute an? *)
8    IF Uhr_0(IN:=1, T#24h).ET = T#1m THEN
9      (* JA: Zurücksetzen *)
10     Uhr_0(IN:=0, T#24h);
11   END_IF;
12   IF Uhr_0(IN:=1, T#24h).ET < T#30s THEN
13     Gruen_0 := TRUE;
14   ELSE
15     Gruen_0 := FALSE;
16   END_IF;
17
18   IF Uhr_1(IN:=1, T#24h).ET = T#1m THEN
19     Uhr_1(IN:=0, T#24h);
20   END_IF;
21   IF Uhr_1(IN:=1, T#24h).ET >= T#30s THEN
22     Gruen_1 := TRUE;
23   ELSE
24     Gruen_1 := FALSE;
25   END_IF;
26 END_FUNCTION_BLOCK
```

Programm 3.1: Vereinfachte Ampelsteuerung in Structured Text.

Viele Zeitmodelle wie Zeitautomaten oder Zeitpetrinetze setzen voraus, dass alle Uhren eines Modells exakt gleich schnell laufen und Zustandswechsel unendlich schnell erfolgen [AD94, Thi02]. Überträgt man diese Annahmen auf SPS, dann hieße das, dass die SPS unendlich schnell liefe und dass die Uhren `Uhr_0` und `Uhr_1` exakt die gleiche Zeit messen und gleichzeitig starten würden. Ein Modellprüfer, der auf diesen Annahmen aufbaut, weist für ein Steuermodell das gewünschte Verhalten nach, d.h. aus Sicht des Modells funktioniert die Steuerung einwandfrei.

Reale Uhren können nur ihre eigene Zeit messen, da physikalische Effekte die Zeitmessung jeder Uhr unterschiedlich beeinflussen. Zwei physikalisch verschiedene reale Uhren besitzen demnach stets eine Abweichung in der Messung der Zeit, diese Abweichung wird in der Literatur als *Drift* bezeichnet. Uhren, die in der Zeitmessung voneinander abweichen, werden *asynchron* genannt. Besitzen Uhren einen einheitlichen Taktgeber und basieren damit physikalisch auf einer einheitlichen Zeitmessung, so besitzen diese Uhren keine Drift. Uhren, die bei der Messung der Zeit nicht voneinander abweichen, werden als *synchron* bezeichnet [SWL90, PSJ04]. Implementiert man das Steuermodell deshalb auf einer realen SPS mit asynchronen Uhren und *Reaktionszeit* ergeben sich folgende Probleme:

1. Die Uhren der SPS basieren möglicherweise nicht auf dem selben Taktgeber und weichen deshalb voneinander ab. Das führt zu einem Auseinanderlaufen der Zeitpunkte, an denen die Grünphasen umgeschaltet werden und die Uhren zurückgesetzt werden.
2. Eine reale SPS benötigt Zeit zur Ausführung von Befehlen. Werden zwei Timer im selben Zyklus zurückgesetzt, liegt dennoch eine Zeitspanne zwischen den realen Befehlen, d.h. zwei Timer können nie gleichzeitig zurückgesetzt werden. Diese Zeitspanne summiert sich im obigen Beispiel und führt zu einer wachsenden Abweichung der Grünphasen.
3. Das Auslesen von Uhren erfolgt aus den gleichen Gründen im Beispiel zu unterschiedlichen Zeitpunkten, wodurch eine scheinbare Abweichung im Gang der Uhren entsteht.
4. Da jeder Zyklus Zeit benötigt, liegt zwischen dem wiederholten Auslesen einer Uhr eine Zeitspanne. Im Beispiel könnte die Uhr `Uhr_0` in einem Zyklus 59 Sekunden und 932 Millisekunden und im darauffolgenden Zyklus eine Minute und 114 Millisekunden anzeigen. Es ist sehr unwahrscheinlich, dass eine Uhr einen bestimmten Zeitwert anzeigt, weshalb exakte Vergleiche von Uhren praktisch kaum möglich sind.

In der Summe würden die genannten Punkte zu einer *Fehlfunktion der realen Steuerung* führen, obwohl das Steuermodell verifiziert wurde. Um das zu vermeiden muss das Steuermodell oder die Implementierung angepasst werden.

Durch eine veränderte Implementierung lassen sich einige der genannten Probleme beseitigen. Dazu wird zur realen Zeitmessung nur eine Uhr der SPS verwendet, alle anderen Uhren leiten sich logisch von dieser Uhr ab. Eine solche Implementierung ist in Algorithmus 3.2 dargestellt.

Beispiel 3.5 (Algorithmus 3.2)**.** *Dieses Beispiel benutzt keine expliziten Timer zur Zeitmessung*

```
1  FUNCTION_BLOCK Beispiel
2    VAR_IN     Ein_0, Ein_1       : BOOL; END_VAR
3    VAR_OUT    Gruen_0, Gruen_1   : BOOL; END_VAR
4    VAR        Now, Uhr_0, Uhr_1  : DATE_AND_TIME; END_VAR
5
6    (* Weise Now den Wert der Systemuhr zu. *)
7    Now := RTC(IN:=1,PDT:=DT#0s).CDT;
8
9    (* Wenn Uhrenstand der Uhr_0 = 1 Min., dann setze Uhr_0 zurück *)
10   IF Now - Uhr_0 =  T#1m  THEN Uhr_0 := Now; END_IF;
11   IF Now - Uhr_0 <  T#30s THEN
12     Gruen_0 := TRUE;
13   ELSE
14     Gruen_0 := FALSE;
15   END_IF;
16
17   IF Now - Uhr_1 =  T#1m  THEN Uhr_1 := Now; END_IF;
18   IF Now - Uhr_1 >= T#30s THEN
19     Gruen_1 := TRUE;
20   ELSE
21     Gruen_1 := FALSE;
22   END_IF;
23 END_FUNCTION_BLOCK
```

Programm 3.2: Verbesserte Implementierung der Ampelschaltung aus Programm 3.1.

sondern leitet die Uhrenstände logischer Uhren vom Uhrenstand der Echzeituhr einer SPS ab. Mit der Anweisung `RTC(IN:=1,PDT:=DT#0s).CDT` *wird der Uhrenstand der Echtzeituhr am Anfang jedes Zykluses ermittelt und in der Variablen* `Now` *gespeichert.*

Die Variablen `Uhr_0` *und* `Uhr_1` *speichern den Uhrenstand der Echtzeituhr, an dem die jeweilige Uhr zuletzt zurückgesetzt wurde. Der Uhrenstand der logischen Uhr* 0 *ergibt sich dann aus der Differenz zwischen dem gegenwärtigen und dem gespeicherten Uhrenstand, also* `Now - Uhr_0`. *Um eine Uhr x zurückzusetzen, wird die Variable* `Uhr_x` *auf* `Now` *gesetzt, womit* `Uhr_x - Now = T#0s` *gilt.*

Mit dieser Implementierung wird gewährleistet, dass

- Uhren im Gang nicht voneinander abweichen (Punkt 1 von Seite 33),
- mehrere Uhren in einem Zyklus logisch zum gleichen Zeitpunkt zurückgesetzt werden (Punkt 2) und
- mehrere Uhren in einem Zyklus logisch zum gleichen Zeitpunkt ausgelesen werden (Punkt 3).

Die Implementierung bietet den Vorteil, dass Zwischenzustände nicht explizit modelliert werden müssen, da alle Operationen auf Uhren in jedem Zyklus logisch gleichzeitig stattfinden. Ein weiterer wesentlicher Vorteil dieser Implementierung ist die Nähe zum Modell, d.h. es ist *nicht notwendig*, an den genannten Stellen *die Semantik der Modelle und Modellprüfer der Realität anzupassen*.

Punkt 4 der Auflistung von Seite 33, d.h. das Vorhandensein von *Reaktionszeiten*, bleibt von dieser Implementierung unbeeinflusst. Da sich Reaktionszeiten in einer realen Steuerung nicht vermeiden lassen, muss an dieser Stelle das *Modell angepasst* werden.

Aus diesen Überlegungen leiten sich folgende Anforderungen an die Implementierung einer Modells ab:

Synchronität Es wird angenommen, dass alle Uhren eines Zeitsystems synchron sind. Diese Forderung kann durch das Zeitsystem oder durch eine angepasste Implementierung erfüllt werden.

Zurücksetzen Es wird angenommen, dass beim Zurücksetzen der Uhrenstand vom Anfang des Zyklus maßgebend ist und das Zurücksetzen logisch am Zyklusanfang erfolgt. Ferner werden alle Uhren in einem Zyklus logisch gleichzeitig zurückgesetzt.

3.2 Symbolische Zustände

Dieser Abschnitt widmet sich der Definition und Darstellung von Uhrenständen. Dazu werden im ersten Unterabschnitt Basisdefinitionen des verwendeten Zeitbegriffs sowie notwendiger Operationen vorgestellt. Beschränkungen von Uhrenständen bilden die Grundlage für die meisten symbolischen Zustände wie Regionen, Differenzgrenz-Matrizen, Entscheidungsdiagrammen und Splittrees. Sie werden im zweiten Unterabschnitt eingeführt. Im letzten Unterabschnitt werden Differenzgrenz-Matrizen und die wichtigsten Operationen auf ihnen definiert.

3.2.1 Zeitbegriff

Im weiteren werden Uhren mit u bezeichnet. $\mathbb{Q}_{>0}$ bezeichne die Menge der positiven gebrochenen Zahlen, $\mathbb{Q}_{\geq 0}$ die Menge der positiven gebrochenen Zahlen und 0. Eine endliche Menge Uhren $\{u_1, u_2, \ldots, u_n\}$ wird mit U bezeichnet. Ein Uhrenstand $\upsilon : U \to \mathbb{Q}_{\geq 0}$ weist jeder Uhr in U einen positiven Zeitwert oder 0 zu. Für einen Uhrenstand υ wird der von einer Uhr $u \in U$ angezeigte Zeitwert mit $\upsilon(u) \in \mathbb{Q}_{\geq 0}$ bezeichnet. Da $\mathbb{Q}_{\geq 0}$ unendlich ist, enthält die Menge aller Uhrenstände $\mathbb{Q}_{\geq 0}^{U}$ unendlich viele Uhrenstände υ.

Aus den Annahmen in Abschnitt 3.1.1 ergibt sich, dass die folgenden Operationen durch ein Zeitsystem auf Uhrenständen ausgeführt werden können: Für einen beliebigen Uhrenstand υ und ein beliebiges $\tau \in \mathbb{Q}_{>0}$ bezeichnet $\upsilon + \tau$ den Uhrenstand, für die für alle Uhren $u \in U$ gilt, dass $(\upsilon + \tau)(u) = \upsilon(u) + \tau$. Die Operation $\upsilon + \tau$ wird als *Zeitvorlauf* von υ um τ bezeichnet. Für die Analyse von Zeitsystemen wird zusätzlich die Operation $\upsilon - \tau$, wobei $\tau \leq \min\{\upsilon(u) : u \in U\}$ und $(\upsilon - \tau)(u) = \upsilon(u) - \tau$, als *Zeitrücklauf* definiert.

Eine weitere Operation ist das *Zurücksetzen* einzelner Uhren auf 0. Für eine Teilmenge von Uhren $U' \subseteq U$ bezeichnet $\upsilon[U' := 0]$ den Uhrenstand, für den gilt: Alle Uhren $u \in U'$ zeigen den Zeitwert $\upsilon[U' := 0](u) = 0$ an und alle anderen Uhren $u \in U \setminus U'$ stimmen mit υ überein: $\upsilon[U' := 0](u) = \upsilon(u)$.

Beispiel 3.6. *In Abbildung 3.3 sind die Uhrenstände zweier Uhren u_1 und u_2 in einem* Uhrenstandsdiagramm *aufgetragen. Der Uhrenstand der Uhr u_1 kann an der Abszisse, der Uhrenstand von u_2 an der Ordinate abgelesen werden. Für einen Uhrenstand $\upsilon = \{u_1 \mapsto 1.5, u_2 \mapsto 2.5\}$ ist $\upsilon + 1.5 = \{u_1 \mapsto 3, u_2 \mapsto 4\}$ der Zeitvorlauf um $1,5$ Zeiteinheiten, $\upsilon - 1 = \{u_1 \mapsto 0.5, u_2 \mapsto 1.5\}$ der Zeitrücklauf um eine Zeiteinheit und $\upsilon[\{u_2\} := 0] = \{u_1 \mapsto 1.5, u_2 \mapsto 0\}$ der Uhrenstand, der durch das Zurücksetzen der Uhr u_2 aus υ entsteht.*

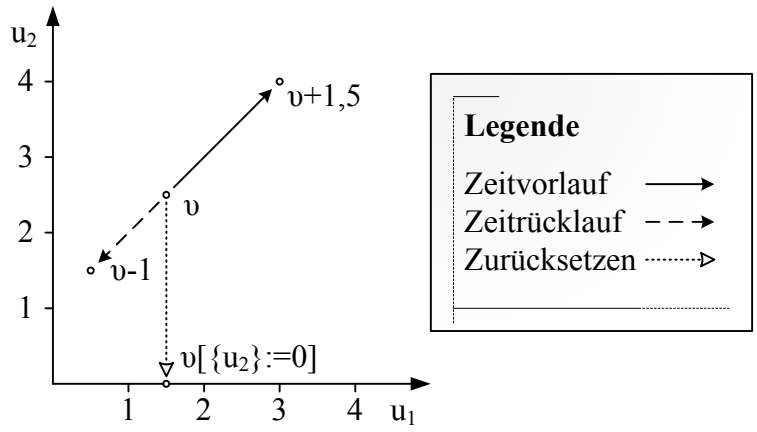


Abbildung 3.3: Operationen auf Uhrenständen.

3.2.2 Beschränkungen

Die meisten symbolischen Zustände für Zeitsysteme basieren auf Ungleichungen über Uhren. Obwohl bei der Definition der meisten symbolischen Zustände diese Ungleichungen nicht explizit betrachtet werden, ergeben sich dennoch Übereinstimmungen in Form und Behandlung, die zu vergleichbaren Problemstellungen führen. Zur Unterscheidung von allgemeinen Ungleichungen werden solche Ungleichungen im Weiteren als Beschränkungen bezeichnet.

Dieser Unterabschnitt definiert Beschränkungen sowie wesentliche Operationen explizit, um eine einheitliche Basis für die Definition weiterer symbolischen Zustände zu legen.

Definition 3.7 (Beschränkung)**.** *Sei* $U = \{u_1, \ldots, u_n\}$ *eine Menge von Uhren. Ferner sei* u_0 *eine Uhr, die stets* 0 *anzeigt, d.h. für alle Uhrenstände* $\upsilon \in \mathbb{Q}_{\geq 0}^{U \cup \{u_0\}}$ *gilt* $\upsilon(u_0) = 0$*. Seien* $i, j \in \{0, \ldots, n\}$ *mit* $i \neq j$ *die Indizes zweier Uhren,* $\lozenge \in \{<, \leq\}$ *ein Relationssymbol und* $c \in \mathbb{Q}$ *eine beliebige Konstante. Die Ungleichung*

$$\beta^{i-j} := u_i - u_j \lozenge c$$

wird als Beschränkung *über U bezeichnet. Die Menge aller Uhrenstände, die durch* β^{i-j} *beschränkt werden, wird mit*

$$[\![\beta^{i-j}]\!] := \{\upsilon \in \mathbb{Q}_{\geq 0}^{U \cup \{u_0\}} : \upsilon(u_0) = 0 \wedge \upsilon(u_i) - \upsilon(u_j) \lozenge c\}$$

bezeichnet.

Anmerkung 3.8. *Da* $\upsilon(u_0)$ *stets* 0 *gilt, werden Beschränkungen der Form* $\beta^{i-0} = u_i - u_0 \lozenge c = u_i \lozenge c$ *und Beschränkungen der Form* $\beta^{0-i} = u_0 - u_i \lozenge c = -u_i \lozenge c$ *mit* $i \in \{1, \ldots, n\}$ orthogonale Beschränkungen *genannt. Alle anderen Beschränkungen heißen* diagonale Beschränkungen.

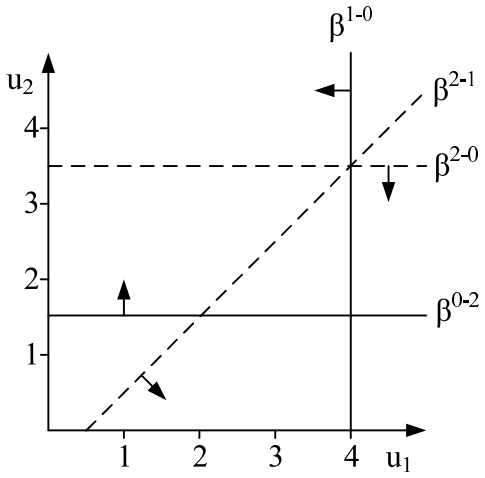


Abbildung 3.4: Beispiele für Beschränkungen.

In der im Weiteren verwendeten Schreibweise bezeichnet β eine beliebige Beschränkung, β^{i-j} bezeichnet eine Beschränkung der Uhren i und j mit $i, j \in \{0, \ldots, |U|\}$ und $i \neq j$. Verschiedene Beschränkungen werden rechts unten indiziert, beispielsweise $\beta_1^{i-j} = u_i - u_j < 3$ und $\beta_2^{i-j} = u_i - u_j \leq 0$.

Beispiel 3.9. *In Abbildung 3.4 sind für zwei Uhren u_1 und u_2 die Beschränkungen*

$$\beta^{1-0} = u_1 - u_0 \leq 4,0 \qquad \beta^{0-2} = u_0 - u_2 \leq -1,5$$
$$\beta^{2-0} = u_2 - u_0 < 3,5 \qquad \beta^{2-1} = u_2 - u_1 < -0,5$$

dargestellt.

Wird eine Gerade durchgezogen gezeichnet, so ist sie Teilmenge der beschränkten Uhrenstände, andernfalls nicht. Die Richtung der Pfeile an den Geraden im Diagramm zeigt an, welche Uhrenstände beschränkt werden.

Im Folgenden wird das Komplement einer Beschränkung definiert. Jede beliebige Beschränkung besitzt genau ein Komplement, welches ebenfalls als Beschränkung dargestellt werden kann.

Definition 3.10 (Komplement)**.** *Sei U eine Menge von Uhren und β eine beliebige Beschränkung über U. Eine komplementäre Beschränkung von β, geschrieben $\complement\beta$, ist eine Beschränkung, für die gilt: $[\![\complement\beta]\!] = \mathbb{Q}_{\geq 0}^U \setminus [\![\beta]\!]$.*

Korollar 3.11. *Aus Definition 3.10 kann sofort abgeleitet werden, dass für jede beliebige Beschränkung $\beta^{i-j} = u_i - u_j \diamond c$ das Komplement von β^{i-j} als*

$$\complement\beta^{i-j} = u_j - u_i \diamond' -c$$

dargestellt werden kann, wobei $\diamond' =<$ gdw. $\diamond =\leq$ und $\diamond' =\leq$ gdw. $\diamond =<$.

Schärfere Beschränkungen

Beschränkungen können bezüglich der von ihnen beschränkten Uhrenstände miteinander verglichen werden.

Definition 3.12. *Sei $U = \{u_1, \ldots, u_n\}$ eine Menge von Uhren und β_1 und β_2 zwei Beschränkungen über U. Die Beschränkung β_1 ist genau dann schärfer als die Beschränkung β_2, geschrieben $\beta_1 \lhd \beta_2$, wenn β_2 echt mehr Uhrenstände als β_1 beschränkt, d.h. $[\![\beta_1]\!] \subset [\![\beta_2]\!]$. Die Beschränkung β_1 ist mindestens so scharf wie die Beschränkung β_2, geschrieben $\beta_1 \trianglelefteq \beta_2$, genau dann wenn β_2 mindestens die Uhrenstände von β_1 beschränkt, d.h. $[\![\beta_1]\!] \subseteq [\![\beta_2]\!]$.*

Anmerkung 3.13. *Eine Beschränkung β mit $[\![\beta]\!] = \varnothing$ ist laut Definition 3.12 mindestens so scharf wie jede beliebige Beschränkung. Aus Definition 3.12 lässt sich ableiten, dass eine Beschränkung β_1^{i-j}, die mindestens einen Uhrenstand beschränkt, nur schärfer als eine Beschränkung β_2^{i-j} der gleichen Uhren sein kann.*

Beschränkungen verschiedener Uhren sind somit nicht vergleichbar, weshalb die Menge aller Beschränkungen für ein festes U mit $\trianglelefteq$ eine Halbordnung *und $\lhd$ eine* strenge Halbordnung *über allen Beschränkungen bildet. Werden nur Beschränkungen der gleichen Uhren betrachtet, bilden $\trianglelefteq$ und $\lhd$ (strenge)* Totalordnungen.

Ob eine Beschränkung schärfer als eine andere ist, kann mit dem folgenden Satz festgestellt werden:

Lemma 3.14 (Schärfere Beschränkung)**.** *Sei $U = \{u_1, \ldots, u_n\}$ eine Menge von Uhren und $i, j \in \{0, \ldots, n\}$ mit $i \neq j$ zwei Uhrenindizes. Ferner seien β_1^{i-j} und β_2^{i-j} zwei Beschränkungen mit*

$$\beta_1^{i-j} = u_i - u_j \diamond_1 c_1 \qquad \diamond_1 \in \{<, \leq\}, c_1 \in \mathbb{Q}$$
$$\beta_2^{i-j} = u_i - u_j \diamond_2 c_2 \qquad \diamond_2 \in \{<, \leq\}, c_2 \in \mathbb{Q}$$

Die Beschränkung β_1^{i-j} ist genau dann schärfer als die Beschränkung β_2^{i-j}, wenn (i) $c_1 < c_2$ oder (ii) $c_1 = c_2$ und $\diamond_1 = <$ und $\diamond_2 = \leq$ gilt.

Beweis. Die Aussage von Satz 3.14 ergibt sich aus der Definition der Relationssymbole $<$ und $\leq$. □

Beispiel 3.15. *Die Beschränkung $u_1 - u_0 < 9$ ist schärfer als die Beschränkung $u_1 - u_0 \leq 9$. Die Beschränkung $u_3 - u_1 \leq -3$ ist schärfer als die Beschränkung $u_3 - u_1 < 1$.*

Die Beschränkung $u_1 < 9$ ist nicht schärfer als die Beschränkung $u_1 \geq 5$ und umgekehrt, damit besteht zwischen den beiden Beschränkungen keine „ist schärfer als“-Beziehung, die beiden Beschränkungen sind nicht vergleichbar. Die Beschränkungen $u_1 \leq 4, u_2 < 3$ sowie $u_1 - u_2 \leq -2$ und $u_2 - u_1 \leq -2$ sind ebenfalls nicht vergleichbar.

Abgeleitete Beschränkungen

In Abbildung 3.4 werden alle Uhrenstände, die sowohl durch β^{1-0} als auch durch β^{0-2} beschränkt werden, ebenfalls durch $\beta^{1-2} = u_1 - u_2 \leq 2$ beschränkt. Eine Beschränkung β^{1-2} muss nicht explizit gegeben sein, sondern kann aus β^{1-0} und β^{0-2} gewonnen werden und wird als abgeleitete Beschränkung bezeichnet. Allgemein kann eine abgeleitete Beschränkung aus einer Menge von Beschränkungen folgendermaßen gewonnen werden:

Definition 3.16 (Abgeleitete Beschränkung)**.** *Gegeben seien n paarweise verschiedene Uhren* $u_{i_1}, \ldots, u_{i_n} \in U \cup \{u_0\}$*, wobei* $2 \leq n \leq |U|$ *und* $i_1, \ldots, i_n \in \{0, \ldots, |U|\}$*. Ferner seien* $n - 1$ *Beschränkungen der Form*

$$\begin{aligned} \beta^{i_1-i_2} &= u_{i_1} - u_{i_2} \diamond_{i_1,i_2} c_{i_1,i_2} \\ \beta^{i_2-i_3} &= u_{i_2} - u_{i_3} \diamond_{i_2,i_3} c_{i_2,i_3} \\ &\vdots \\ \beta^{i_{n-1}-i_n} &= u_{i_{n-1}} - u_{i_n} \diamond_{i_{n-1},i_n} c_{i_{n-1},i_n} \end{aligned}$$

gegeben.

Eine abgeleitete Beschränkung, geschrieben $\beta^{i_1,\ldots,i_n}$*, ist definiert als*

$$\beta^{i_1,\ldots,i_n} := u_{i_1} - u_{i_n} \diamond' \sum_{1 \leq k < n} c_{i_k,i_{k+1}}$$

wobei

$$\diamond' := \begin{cases} \leq & \text{wenn alle } \diamond_{i_k,i_{k+1}} = \leq \\ < & \text{sonst} \end{cases}$$

Beispiel 3.17. *In Abbildung 3.4 kann aus den Beschränkungen*

$$\beta^{2-1} = u_2 - u_1 < -0,5 \qquad \beta^{1-0} = u_1 - u_0 \leq 4,0$$

die Beschränkung

$$\beta^{2,1,0} = u_2 - u_0 < -0,5 + 4,0$$

abgeleitet werden. $\beta^{2,1,0}$ *entspricht der Beschränkung* β^{2-0} *aus Abbildung 3.4.*

Eine abgeleitete Beschränkung $\beta^{i_1,\ldots,i_n}$ enthält alle Uhrenstände, die in $\bigcap_{1 \leq k < n} [\![\beta^{i_k - i_{k+1}}]\!]$ enthalten sind.

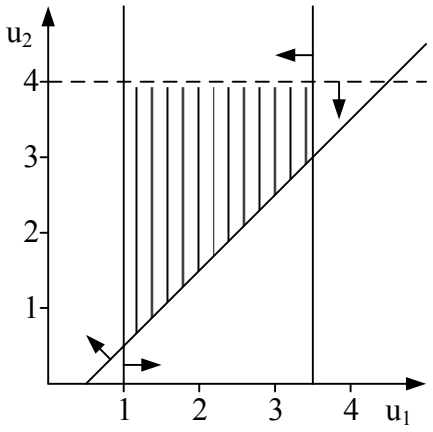


Abbildung 3.5: Beispiel für einen Hyperpolyeder.

3.2.3 Hyperpolyeder

Eine konjunktive Verknüpfung von Beschränkungen wird im Weiteren als Hyperpolyeder bezeichnet.

Definition 3.18 (Hyperpolyeder)**.** *Sei $U = \{u_1, \ldots, u_n\}$ eine Menge von Uhren und $\beta_1, \ldots, \beta_m$ eine beliebige Anzahl orthogonaler und diagonaler Beschränkungen über U. Eine Konjunktion von Beschränkungen*

$$\hbar := \beta_1 \wedge \cdots \wedge \beta_m$$

wird als Hyperpolyeder $\hbar$ über U bezeichnet.

Der Ausdruck $[\![\hbar]\!] = [\![\beta_1]\!] \cap \ldots \cap [\![\beta_m]\!]$ bezeichnet die Menge aller Uhrenstände, die in $\hbar$ enthalten sind.

Zur Vereinfachung wird im Weiteren angenommen, dass ein Hyperpolyeder $\hbar$ eine Beschränkung β enhält, geschrieben $\beta \in \hbar$, wenn $\hbar = \beta^1 \wedge \ldots \wedge \beta^n$ und ein $i \in \{1, \ldots, n\}$ existiert, so dass $\beta = \beta^i$ gilt. Ein Hyperpolyeder $\hbar$, für den $[\![\hbar]\!] = \varnothing$ gilt, wird als *leerer Hyperpolyeder* bezeichnet.

Beispiel 3.19. *In Abbildung 3.5 sind für zwei Uhren u_1 und u_2 und einen Hyperpolyeder*

$$\hbar = u_0 - u_1 \leq -1 \wedge u_1 - u_0 \leq 3,5 \wedge u_2 - u_0 < 4 \wedge u_1 - u_2 \leq 0,5$$

die Uhrenstände schraffiert dargestellt, die in $\hbar$ enthalten sind.

Operationen auf Hyperpolyedern

Im Folgenden werden einige Operationen auf Hyperpolyedern definiert, die für die weitere Arbeit notwendig sind. Diese Operationen sind mit Differenzgrenz-Matrizen implementierbar [BY04].

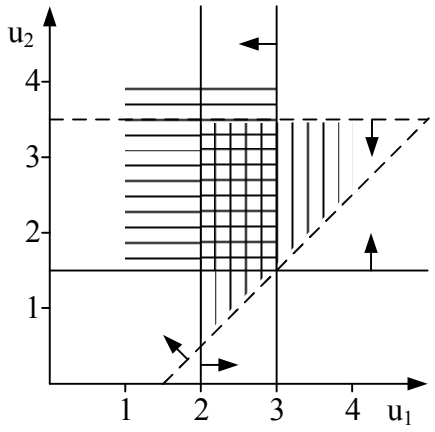


Abbildung 3.6: Schnittmenge zweier Hyperpolyeder.

Schärfere Hyperpolyeder

Wie für Beschränkungen kann auch für Hyperpolyeder eine „schärfer als“-Beziehung definiert werden.

Definition 3.20 (Schärfer)**.** *Ein Hyperpolyeder $\hbar_1$ ist echt schärfer als ein Hyperpolyeder $\hbar_2$, geschrieben $\hbar_1 \lhd \hbar_2$, gdw. $[\![\hbar_1]\!] \subset [\![\hbar_2]\!]$. Entsprechend ist $\hbar_1$ schärfer als $\hbar_2$ ($\hbar_1 \trianglelefteq \hbar_2$) gdw. $[\![\hbar_1]\!] \subseteq [\![\hbar_2]\!]$.*

Anmerkung 3.21. *Die Menge aller Hyperpolyeder bildet für ein festes U mit $\trianglelefteq$ eine* Halbordnung *und mit $\lhd$ eine* strenge Halbordnung.

Schnitt

Die Schnittmenge zweier Hyperpolyeder ist wie folgt definiert:

Definition 3.22 (Schnitt)**.** *Seien $\hbar$ und $\hbar'$ zwei Hyperpolyeder. Ein Hyperpolyeder wird als Schnitt von $\hbar$ und $\hbar'$, geschrieben $\hbar \cap \hbar'$, bezeichnet, wenn gilt: $[\![\hbar \cap \hbar']\!] = [\![\hbar]\!] \cap [\![\hbar']\!]$.*

Beispiel 3.23. *In Abbildung 3.6 sind zwei horizontal und vertikal schraffierte Hyperpolyeder $\hbar_1$ und $\hbar_2$ sowie ihr gekreuzt schraffierter Schnitt*

$$\begin{aligned}\hbar_1 \cap \hbar_2 =& u_1 - u_0 \leq 3 \wedge u_2 - u_0 < 3,5 \wedge u_0 - u_1 \leq -2 \wedge \\ & u_0 - u_2 \leq -1,5 \wedge u_1 - u_2 < 1,5\end{aligned}$$

dargestellt.

Zur besseren Übersicht sind nur die Beschränkungen von $\hbar_1 \cap \hbar_2$ in der Abbildung eingezeichnet.

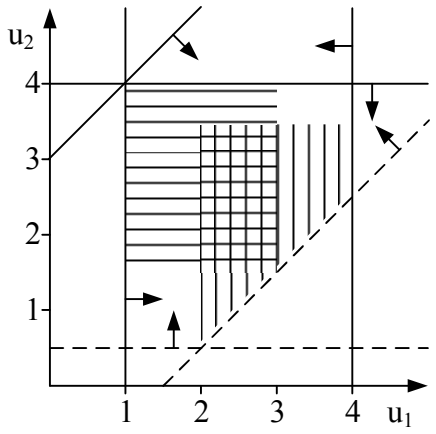


Abbildung 3.7: Abschluss zweier Hyperpolyeder.

Anmerkung 3.24. *In Satz 5.12 wird gezeigt, dass der Schnitt zweier Hyperpolyeder ebenfalls als Hyperpolyeder darstellbar ist. Damit bildet für ein festes U die Menge aller Hyperpolyeder mit* $\cap$ *einen* Halbverband, *wobei der leere Hyperpolyeder die größte untere Schranke aller Hyperpolyeder bildet.*

Abschluss

Im Gegensatz zum Schnitt kann die Vereinigung zweier Hyperpolyeder nicht immer durch einen einzelnen Hyperpolyeder dargestellt werden. So kann etwa die Vereinigung der Hyperpolyeder aus Beispiel 3.23 nicht durch einen einzigen Hyperpolyeder dargestellt werden. Es ist möglich, die Vereinigung der Hyperpolyeder durch einen Hyperpolyeder nach oben zu beschränken. Der schärfste beschränkende Hyperpolyeder wird als Abschluss bezeichnet:

Definition 3.25 (Abschluss)**.** *Seien* $\hbar$ *und* $\hbar'$ *zwei beliebige, nicht leere Hyperpolyeder. Ein Hyperpolyeder* $\hbar_1 \oplus \hbar_2$ *wird als Abschluss von* $\hbar_1$ *und* $\hbar_2$ *bezeichnet, wenn* (*i*) $[\![\hbar_1]\!] \cup [\![\hbar_2]\!] \subseteq [\![\hbar_1 \oplus \hbar_2]\!]$ *und* (*ii*) *es keinen Hyperpolyeder* $\hbar_3$ *gibt, so dass* $[\![\hbar_3]\!] \subset [\![\hbar_1 \oplus \hbar_2]\!]$ *und* $[\![\hbar_1]\!] \cup [\![\hbar_2]\!] \subseteq [\![\hbar_3]\!]$ *gilt.*

Beispiel 3.26. *In Abbildung 3.7 ist der Abschluss*

$$\begin{aligned}\hbar_1 \oplus \hbar_2 =& u_1 - u_0 \leq 4 \wedge u_2 - u_0 \leq 4 \wedge u_0 - u_1 \leq -1 \wedge \\ & u_0 - u_2 < -0,5 \wedge u_1 - u_2 < 1,5 \wedge u_2 - u_1 \leq 3\end{aligned}$$

der Hyperpolyeder $\hbar_1$ *und* $\hbar_2$ *aus Beispiel 3.23 dargestellt.*

Anmerkung 3.27. *Der Abschluss ist laut Satz 5.14 als Hyperpolyeder darstellbar. Damit bildet für ein festes U die Menge aller Hyperpolyeder mit* $\cap$ *und* $\oplus$ *einen* Verband, *wobei der* leere Hyperpolyeder *die größte untere Schranke und ein* Hyperpolyeder ohne Beschränkungen *die kleinste obere Schranke aller Hyperpolyeder bildet.*

3.3 Zusammenfassung

Folgende Annahmen über die verwendeten Steuermodelle und -programme sind für die weitere Arbeit wichtig:

- Speicherprogrammierbare Steuerungen besitzen mehrere Ein- und Ausgänge. Bei der Analyse von Modellen muss beachtet werden, dass mehrere Eingänge gleichzeitig aktiv sein können. Ferner können die Ausgänge einer SPS z.B. zeitgesteuert aktiviert werden, auch wenn keine Änderung an den Eingängen stattgefunden hat.
- Speicherprogrammierbare Steuerungen besitzen eine zyklische Arbeitsweise.
- Annahmen wie die Synchronität oder das gleichzeitige Zurücksetzen mehrerer Uhren können durch eine geeignete Implementierung gewährleistet werden. Das ermöglicht die Verwendung von Standardalgorithmen und -software zur Analyse der Modelle.
- Reaktionszeiten lassen sich auch durch geeignete Implementierung nicht vermeiden und müssen deshalb explizit im Modell beachtet werden.

Beschränkungen bilden die Grundlage der meisten zur Analyse von Zeitsystemen verwendeten Datenstrukturen wie Differenzgrenz-Matrizen, Regionen, Entscheidungsdiagrammen oder Splittrees. Mit Hyperpolyedern sowie den Operationen „Schärfer“ und „Abschluss“ wurde ferner ein Verband definiert, welcher Beschränkungen, Differenzgrenz-Matrizen und Regionen verallgemeinert. Hyperpolyeder bilden damit im Gegensatz zu anderen bekannten Strukturen eine einheitliche Grundlage für die in der weiteren Arbeit verwendeten Datenstrukturen.

4 Definition, Analyse und Vergleich von SPS-Zeitautomaten

Im folgenden Kapitel wird gezeigt, wie eine Translation Validation für Steuermodelle und -programme mit Zeitbedingungen durchgeführt werden kann.

Eine besondere Herausforderung für die *praktische Durchführung* der Translation Validation ist die *komplexe Semantik* der vorgegebenen Steuermodelle und -programme. Dabei müssen die *Eigenschaften Speicherprogrammierbarer Steuerungen*, d.h. die zyklischen Arbeitsweise, Reaktionszeiten sowie mehrerer Ein- und Ausgaben beachtet werden.

Um die komplexe Semantik zu bewältigen, werden für die Translation Validation *einfachere Modelle* verwendet. Ein etabliertes Modell mit Zeitbedingungen sind Zeitautomaten. *Diskrete Zustände* der Steuermodelle und -programme wie Petrinetzmarkierungen oder Variablenbelegungen können mit Zeitautomaten direkt *einheitlich dargestellt* werden, was ein wesentlicher Vorteil von Zeitautomaten ist. Klassische Zeitautomaten nach Alur/Dill haben aber den Nachteil, die Eigenschaften Speicherprogrammierbarer Steuerungen nicht abzubilden.

Um dem zu begegnen wird bei der Translation Validation intern ein modifiziertes Zeitautomatenmodell, sogenannte *SPS-Zeitautomaten*, verwendet, das die genannten Eigenschaften besitzt. Damit wird vermieden, dass die Anzahl der Zustände des Zeitautomaten durch explizite Modellierung der SPS-Eigenschaften ansteigt. Dies ermöglicht eine effiziente und damit praktisch durchführbare Translation Validation.

Das Kapitel gliedert sich in vier Abschnitte:

- Im ersten Abschnitt werden SPS-Zeitautomaten und deren Semantik definiert.
- Im zweiten Abschnitt werden Zonen als symbolische Repräsentation der Zeitzustände eines SPS-Zeitautomaten eingeführt. Zudem werden die zur Nachbildung der Semantik und Analyse von SPS-Zeitautomaten notwendigen Operationen eingeführt.
- Im dritten Abschnitt wird gezeigt, wie mit Hilfe dieser Operationen die für die Translation Validation notwendigen Zonengraphen erstellt werden können.
- Im letzten Abschnitt wird schließlich gezeigt, wann eine Implementierung als korrekt verstanden wird und wie die Korrektheit durch den Vergleich zweier Zonengraphen nachgewiesen werden kann.

4.1 SPS-Zeitautomaten

In diesem Abschnitt werden SPS-Zeitautomaten eingeführt, die die Modellierung von Zeitsystemen unter den im Kapitel 3 genannten Annahmen ermöglichen. Nach der Definition von Schaltbedingungen und SPS-Zeitautomaten wird die Semantik von SPS-Zeitautomaten mit Hilfe von Transitionssystemen formalisiert.

4.1.1 Schaltbedingungen

Eingangsbedingungen

Ein Zeitsystem soll feststellen können, ob eine Menge von Eingängen aktiv oder inaktiv ist. Zu diesem Zweck, werden Eingangsbedingungen definiert.

Definition 4.1 (Eingangsbedingung)**.** *Sei Σ^e eine Menge von Eingängen. Eine atomare Bedingung B eines Eingangs $\sigma^e \in \Sigma^e$ hat die Form $B = \sigma^e$ oder $B = \neg\sigma^e$. Eine Eingangsbelegung $\Sigma^{e'} \subseteq \Sigma^e$ erfüllt eine atomare Bedingung $B = \sigma^e$, geschrieben $\Sigma^{e'} \models B$, gdw. $\sigma^e \in \Sigma^{e'}$. Für $B = \neg\sigma^e$ gilt entsprechend $\Sigma^{e'} \models B$ gdw. $\sigma^e \notin \Sigma^{e'}$.*

Seien $B_1, \ldots, B_n$ mit $n \geq 0$ eine endliche Anzahl atomarer Bedingungen der Eingänge Σ^e. Dann wird

$$B^e := B_1 \wedge \ldots \wedge B_n$$

als Eingangsbedingung *bezeichnet. Eine Eingangsbelegung $\Sigma^{e'} \subseteq \Sigma^e$ erfüllt eine Eingangsbedingung B^e, geschrieben $\Sigma^{e'} \models B^e$, gdw. $\Sigma^{e'}$ jede atomare Bedingung B_i von B^e erfüllt. Für eine Eingangsbedingung B^e enthält die Menge*

$$\Sigma^{e'}(B^e) := \{\Sigma^{e'} \in \Sigma^e : \Sigma^{e'} \models B^e\}$$

die Eingangsbelegungen, die B^e erfüllen.

Korollar 4.2. *Die Negation einer atomaren Bedingung B kann ebenfalls als atomare Bedingung dargestellt werden:*

$$\neg(B) = \begin{cases} \neg\sigma^e & \text{wenn } B = \sigma^e \\ \sigma^e & \text{wenn } B = \neg\sigma^e \end{cases}$$

Zeitbedingungen

Als Zeitbedingungen werden ausschließlich Vergleiche einzelner Uhren mit Konstanten, d.h. orthogonale Beschränkungen, zugelassen. Diese Einschränkung ist in der Praxis ausreichend,

da TNCES ausschließlich orthogonale Beschränkungen verwenden. Darüber hinaus erschwert die Verwendung diagonaler Beschränkungen in Modellen deren Analyse [Bou03].

Diese Einschränkung trifft nicht auf die Datenstrukturen und Operationen zu, die zur Analyse und Translation Validation von Zeitautomaten benötigt werden. Diese Datenstrukturen und Operationen verwenden sowohl orthogonale als auch diagonale Beschränkungen.

Definition 4.3 (Zeitbedingung)**.** *Eine* Zeitbedingung B^u *ist ein Hyperpolyeder, der ausschließlich orthogonaler Beschränkungen enthält. Ein Uhrenstand* υ *erfüllt eine Zeitbedingung* B^u*, geschrieben* $\upsilon \models B^u$ *gdw.* $\upsilon \in [\![B^u]\!]$.

Anmerkung 4.4. *Zur Vereinfachung wird in Modellen* $u_i \diamond c, \diamond \in \{<, \leq\}$ *statt* $u_i - u_0 \diamond c$ *geschrieben sowie* $u_i \diamond c, \diamond \in \{>, \geq\}$ *statt* $u_0 - u_i < -c$ *bzw.* $u_0 - u_i \leq -c$.

Schaltbedingungen

Ein Zeitsystem soll auf die Belegung seiner Eingänge und auf den Uhrenstand seiner Uhren reagiern können. Schaltbedingungen setzen sich aus Eingangsbedingungen und Zeitbedingungen zusammen, damit ein Zeitsystem auf die Belegung seiner Eingänge und auf den Uhrenstand seiner Uhren reagieren kann.

Definition 4.5 (Schaltbedingung)**.** *Sei* B^e *eine Eingangsbedingung über den Eingängen* Σ^e *und* B^u *eine Zeitbedingung über den Uhren* U*. Dann wird*

$$B := B^e \wedge B^u$$

als Schaltbedingung *über* Σ^e *und* U *bezeichnet. Die Menge aller Schaltbedingungen über* Σ^e *und* U *wird mit* $\mathcal{P}(B)$ *bezeichnet.*

Analog zu Eingangsbedingungen und Zeitbedingungen erfüllt eine Eingangsbelegung $\Sigma^{e'} \in \Sigma^e$ *und ein Uhrenstand* $\upsilon \in \mathbb{Q}^U_{\geq 0}$ *eine Schaltbedingung* B*, geschrieben* $\Sigma^{e'}, \upsilon \models B$*, gdw.* $\Sigma^{e'} \models B^e$ *und* $\upsilon \models B^u$.

Anmerkung 4.6. *Die Disjunktion* $B_1 \vee B_2$ *zweier Schaltbedingungen kann durch das Einführen von zwei Zustandsübergängen in einem Zeitsystem modelliert werden. Beide Zustandsübergänge sind bis auf ihre Schaltbedingung identisch, wobei ein Zustandsübergang die Schaltbedingung* B_1 *besitzt und der andere die Schaltbedingung* B_2.

Gemäß Korollar 4.2 und Korollar 3.11 können die Negationen atomarer Bedingungen bzw. Beschränkungen ebenfalls als atomare Bedingungen bzw. Beschränkungen dargestellt werden. Somit kann jede aussagenlogische Formel über atomaren Bedingungen in disjunktiver Normalform dargestellt werden. Daraus folgt, dass beliebige aussagenlogische Formeln über atomaren Bedingungen für Zeitsysteme mit Schaltbedingungen darstellbar sind.

Weitere Operationen können mit Schaltbedingungen leicht nachgebildet werden:

true	$\equiv$	$\sigma^e \vee \neg\sigma^e$
false	$\equiv$	$\sigma^e \wedge \neg\sigma^e$
$u = c$	$\equiv$	$u \leq c \wedge u \geq c$
$B_1 \Leftrightarrow B_2$	$\equiv$	$(B_1 \wedge B_2) \vee (\neg B_1 \wedge \neg B_2)$
$\sigma_1^e = \sigma_2^e$	$\equiv$	$\sigma_1^e \Leftrightarrow \sigma_2^e$

Beispiel 4.7. *Sei $U = \{u_1\}$ eine Menge von Uhren und $\Sigma^e = \{\sigma_1^e, \sigma_2^e\}$ eine Menge von Eingängen. Die Aussage $(\sigma_1^e = \sigma_2^e) \wedge (u_1 = 1,25)$ kann folgendermaßen dargestellt werden:*

$$
\begin{aligned}
&(\sigma_1^e = \sigma_2^e) \wedge (u_1 = 1,25) \\
&= ((\sigma_1^e \wedge \sigma_2^e) \vee (\neg\sigma_1^e \wedge \neg\sigma_2^e)) \wedge u_1 \leq 1,25 \wedge u_1 \geq 1,25 \\
&= (\sigma_1^e \wedge \sigma_2^e \wedge u_1 \leq 1,25 \wedge u_1 \geq 1,25) \vee (\neg\sigma_1^e \wedge \neg\sigma_2^e \wedge u_1 \leq 1,25 \wedge u_1 \geq 1,25)
\end{aligned}
$$

Es ergeben sich die Schaltbedingungen

$$
\begin{aligned}
B_1 &= (\sigma_1^e \wedge \sigma_2^e) \wedge (u_1 \leq 1,25 \wedge u_1 \geq 1,25) \text{ und} \\
B_2 &= (\neg\sigma_1^e \wedge \neg\sigma_2^e) \wedge (u_1 \leq 1,25 \wedge u_1 \geq 1,25)
\end{aligned}
$$

4.1.2 Unbeschränkte SPS-Zeitautomaten

Im Folgenden werden SPS-Zeitautomaten definiert, die keine Beschränkungen ihrer Reaktionszeit besitzen.

Definition 4.8 (SPS-Zeitautomat)**.** *Ein* SPS-Zeitautomat $\mathcal{A}$ *ist ein Tupel* $\langle \Sigma^e, \Sigma^a, Z, z^0, U, T \rangle$ *wobei*

- $\Sigma^e = \{\sigma_1^e, \ldots, \sigma_k^e\}$, $k \in \mathbb{N}$ *eine endliche Menge von Eingängen,*
- $\Sigma^a = \{\sigma_1^a, \ldots, \sigma_l^a\}$, $l \in \mathbb{N}$ *eine endliche Menge von Ausgängen mit* $\Sigma^e \cap \Sigma^a = \emptyset$,
- $Z = \{z_1, \ldots, z_m\}$, $m \in \mathbb{N}$ *eine endliche Menge von Zuständen,*
- $z^0 \in Z$ *ein Anfangszustand,*
- $U = \{u_1, \ldots, u_n\}$, $n \in \mathbb{N}$ *eine endliche Menge von Uhren und*
- $T \subseteq Z \times \mathcal{P}(B) \times \mathcal{P}(U) \times \mathcal{P}(\Sigma^a) \times Z$ *eine endliche Menge von Zustandsübergängen ist.*

Beispiel 4.9. *Anhand des Beispiels einer Industriepresse wird im Folgenden eine intuitive Vorstellung von der Semantik von SPS-Zeitautomaten gegeben: Um Unfälle zu vermeiden, muss ein Arbeiter, nachdem er ein Werkstück auf eine Presse gelegt hat, solange links und rechts an der Presse gleichzeitig auf je einen Tastschalter drücken, bis sich das Sicherheitsgitter der Presse geschlossen hat. Der Arbeiter muss seine Hände für mindestens* 5 *Zeiteinheiten an den Sicherheitsschaltern lassen, um zu garantieren, dass das Gitter vollständig geschlossen wurde. Sollte er in dieser Zeit seine Hände von einem der Schalter entfernen, wird der Pressvorgang*

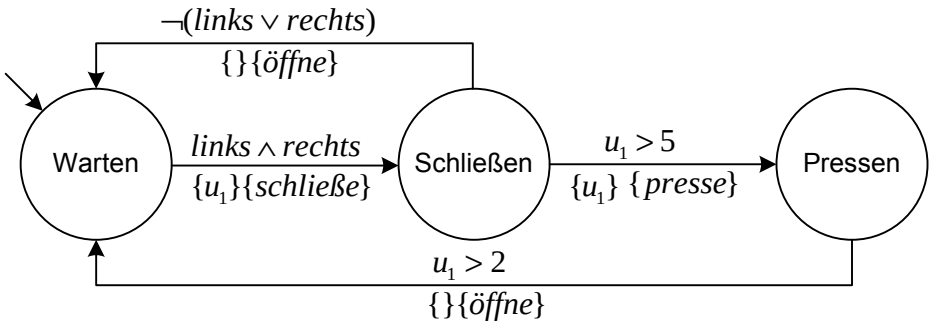


Abbildung 4.1: Beispiel eines SPS-Zeitautomaten.

abgebrochen und das Sicherheitsgitter öffnet sich. Nach dem Verschließen des Sicherheitsgitters wird der Pressvorgang ausgelöst und ist nach 2 *Zeiteinheiten abgeschlossen. Anschließend gibt die Steuerung den Befehl, das Sicherheitsgitter zu öffnen.*

Abbildung 4.1 veranschaulicht einen SPS-Zeitautomaten, der die Steuerung der Industriepresse modelliert. Der SPS-Zeitautomat besitzt die drei Zustände $Z = \{Warten, Schließen, Pressen\}$. *Die Eingänge* $\Sigma^e = \{links, rechts\}$ *repräsentieren das Betätigen des linken bzw. rechten Schalters. Zusätzlich löst die Aktivierung der Ausgänge* $\Sigma^a = \{schließe, öffne, presse\}$ *das Schließen und Öffnen des Sicherheitsgitters sowie das Pressen aus. Die Zustandsübergänge des SPS-Zeitautomaten sind durch Pfeile vom Start- zum Endzustand markiert. Die Schaltbedingung eines Zustandsüberganges befindet sich oberhalb des Pfeils. Unterhalb des Pfeiles befindet sich links eine Menge von Uhren, die während des Zustandsübergangs zurückgesetzt werden. Rechts daneben befindet sich die Menge der Ausgänge, die während des Zustandsübergangs aktiviert werden. Beispielsweise kann der SPS-Zeitautomat den Zustand „Warten" nur verlassen, wenn die Eingänge „links" und „rechts" aktiv sind. Beim Übergang in den Zustand „Schließen" setzt der SPS-Zeitautomat die Uhr* u_1 *zurück und aktiviert den Ausgang„schließe".*

Eingabe

Die tatsächliche Zykluszeit eines realen Zeitsystems, das durch einen SPS-Zeitautomaten modelliert wird, hängt von der verwendeten Hardware und physikalischen Einflüssen, allgemeiner der Umwelt des Zeitsystems, ab und ist für jeden Zyklus verschieden. Da sowohl die Belegung der Eingänge als auch die tatsächliche Zykluszeit von der Umwelt eines SPS-Zeitautomaten abhängen, werden Eingangsbelegung und Zykluszeit als Eingabe des SPS-Zeitautomaten modelliert:

Definition 4.10 (Eingabesymbol, Eingabewort)**.** *Sei* $\mathcal{A} = \langle \Sigma^e, \Sigma^a, Z, z_0, U, T \rangle$ *ein SPS-Zeitautomat. Ein Paar* $\langle \Sigma^{e'}, \tau \rangle$ *wird als Eingabesymbol von* $\mathcal{A}$ *bezeichnet, wenn* $\Sigma^{e'} \subseteq \Sigma^e$ *eine Eingangsbelegung und* $\tau \in \mathbb{Q}_{>0}$ *eine Zykluszeit ist. Eine unendliche Folge von Eingabesymbolen* $w := \langle \Sigma_0^{e'}, \tau_0 \rangle \langle \Sigma_1^{e'}, \tau_1 \rangle \langle \Sigma_2^{e'}, \tau_2 \rangle \ldots$ *bildet ein Eingabewort von* $\mathcal{A}$.

Zeitzustände

Neben der endlichen Menge Z von Zuständen besitzt ein SPS-Zeitautomat eine unendliche Menge $\mathbb{Q}_{\geq 0}^{U}$ möglicher Uhrenstände seiner Uhren. Ein diskreter Zustand und ein Uhrenstand bilden gemeinsam einen Zeitzustand:

Definition 4.11 (Zeitzustand)**.** *Für einen SPS-Zeitautomaten $\mathcal{A}$ ist ein Zeitzustand zz ein Paar $\langle z, \upsilon \rangle \in Z \times \mathbb{Q}_{\geq 0}^{U}$, wobei $z \in Z$ einen diskreten Zustand von $\mathcal{A}$ und $\upsilon \in \mathbb{Q}_{\geq 0}^{U}$ eine Uhrzeit der Uhren von $\mathcal{A}$ beschreibt.*

Semantik

Ein *beschriftetes Transitionssystem* $\mathcal{T}$ ist ein Tupel $\mathcal{T} = \langle S, S_0, \Sigma, T \rangle$, wobei

- S eine beliebige Menge von Zuständen,
- $S_0 \subseteq S$ eine Menge von Startzuständen,
- Σ eine beliebige Menge und
- $T \in S \times \Sigma \times S$ eine Transitionsrelation ist.

Die Semantik von $\mathcal{A}$ wird im Folgenden als zeitbeschriftetes Transitionssystem (Time Labeled Transition System - TLTS) $\mathcal{T} \coloneqq \langle ZZ, ZZ^0, M, TR \rangle$ über den Zeitzuständen von $\mathcal{A}$ definiert. Für $\mathcal{T}$ gilt:

- $ZZ \coloneqq Z \times \mathbb{Q}_{\geq 0}^{U}$ die Zustände des Transitionssystems sind die Zeitzustände des SPS-Zeitautomaten $\mathcal{A}$;
- $ZZ^0 \coloneqq \{\langle z^0, \upsilon[U \coloneqq 0] \rangle\} \in ZZ$ das Transitionssystem besitzt nur einen Startzeitzustand, den Startzustand z^0 von $\mathcal{A}$, wobei alle Uhren in U auf 0 gesetzt sind,
- $M \coloneqq \mathcal{P}(\Sigma^e) \times \mathbb{Q}_{>0} \times \mathcal{P}(\Sigma^a)$ die Transitionen werden mit einer Eingangsbelegung, einer Zykluszeit und einer Ausgangsbelegung markiert und
- $TR \subseteq ZZ \times M \times ZZ = ZZ \times \mathcal{P}(\Sigma^e) \times \mathbb{Q}_{>0} \times \mathcal{P}(\Sigma^a) \times ZZ$ die Transitionen des Transitionssystems seien wie folgt definiert:

Eine Transition $\langle zz, \Sigma^{e'}, \tau, \Sigma^{a'}, zz' \rangle \in TR$ wird im Folgenden auch als $zz \xrightarrow{\Sigma^{e'}, \tau, \Sigma^{a'}} zz'$ geschrieben. Für einen SPS-Zeitautomaten können zwei Arten von Zeitzustandsübergängen unterschieden werden:

Zeitvorlauf: Wechselt ein SPS-Zeitautomat seinen Zustand nicht, so benötigt er eine Zykluszeit τ bis er den nächsten Zeitzustand erreicht. Die Menge der durch einen Zeitvorlauf von einem Zeitzustand $zz = \langle z, \upsilon \rangle$ erreichbaren Zeitzustände ist:

$$\mathbf{post}(zz) \coloneqq \{\langle z, \upsilon + \tau \rangle : \tau \in \mathbb{Q}_{>0}\}$$

Ein SPS-Zeitautomat wechselt seinen Zustand genau dann nicht, wenn für die Eingangsbelegung des SPS-Zeitautomaten kein Zustandsübergang existiert, dessen Schaltbedingung durch die Eingangsbelegung und den Uhrenstand des Zeitzustands erfüllt wird. Werden der Zeitzustand zz und seine Nachfolger mit der Eingabebelegung und Zykluszeit markiert, ergeben sich die Transitionen von $\mathcal{T}$, die einen Zeitvorlauf wiederspiegeln.

$$TR\nearrow(zz) := \left\{ zz \xrightarrow{\Sigma^{e'},\tau,\varnothing} zz' : zz' \in \mathbf{post}(zz) \wedge \forall \langle z, B, U_R, \Sigma^{e'}, z' \rangle \in T.\ \Sigma^{e'}, \upsilon \not\models B \right\}$$

Zustandswechsel: Führt ein SPS-Zeitautomat einen Zustandswechsel aus, so kann er dabei einige seiner Uhren zurücksetzen und benötigt eine Zykluszeit τ bis zum Erreichen des nächsten Zeitzustands. Für einen Zeitzustand $zz = \langle z, \upsilon \rangle$ und einen Zustandsübergang $t = \langle z, B, U_R, \Sigma^{a'}, z' \rangle \in T$, wobei $\upsilon \models B^u$, ergibt sich die Menge der erreichbaren Zeitzustände als:

$$\mathbf{post}(zz, t) := \{ \langle z', \upsilon[U_R := 0] + \tau \rangle : \tau \in \mathbb{Q}_{>0} \}$$

Ein Zustandsübergang t wird von einem SPS-Zeitautomaten nur dann ausgeführt, wenn die Eingabebelegung und der Uhrenstand des Zeitzustandes die Schaltbedingung von t erfüllen. Damit ergeben sich die aufgrund von Zustandsübergängen von zz ausgehenden Transitionen als

$$TR(zz) := \left\{ zz \xrightarrow{\Sigma^{e'},\tau,\Sigma^{a'}} zz' : \exists t = \langle z, B, U_R, \Sigma^{a'}, z' \rangle \in T.\ \Sigma^{e'}, \upsilon \models B \wedge zz' \in \mathbf{post}(zz, t) \right\}$$

Die Menge aller Transitionen TR von T ergibt sich aus der Vereinigung der Transitionen aller Zeitzustände:

$$TR := \bigcup_{zz \in ZZ} TR(zz) \cup TR\nearrow(zz)$$

Definition 4.12 (Lauf)**.** *Sei $\mathcal{A}$ ein beliebiger SPS-Zeitautomat und*

$$w := \langle \Sigma_0^{e'}, \tau_0 \rangle \langle \Sigma_1^{e'}, \tau_1 \rangle \ldots \langle \Sigma_i^{e'}, \tau_i \rangle \ldots$$

ein beliebiges Eingabewort mit $\Sigma_i^{e'} \subseteq \Sigma^e$ und $\tau_i \in \mathbb{Q}_{>0}$ für $i \in \mathbb{N}$. Ein Lauf

$$r := zz_0 \xrightarrow{\Sigma_0^{e'},\tau_0,\Sigma_0^{a'}} zz_1 \xrightarrow{\Sigma_1^{e'},\tau_1,\Sigma_1^{a'}} zz_2 \ldots zz_i \xrightarrow{\Sigma_i^{e'},\tau_i,\Sigma_i^{a'}} zz_{i+1} \ldots$$

ist eine unendliche Folge von Zeitzuständen und Transitionen, die $\mathcal{A}$ über w beginnend mit dem Startzustand durchläuft.

Existieren für ein Eingabewort w und einen SPS-Zeitautomaten $\mathcal{A}$ mehrere Läufe r, so heißt $\mathcal{A}$ *nichtdeterministisch*, andernfalls *deterministisch*. Für einen SPS-Zeitautomaten $\mathcal{A}$ heißt ein Zeitzustand zz *erreichbar*, wenn ein $i \in \mathbb{N}$ und ein Lauf $r = zz_0 \xrightarrow{\Sigma_0^{e'},\tau_0,\Sigma^{a'}_0} zz_1 \ldots zz_i \ldots$ existiert, so dass $zz = zz_i$. Die Menge aller vom Startzeitzustand zz_0 erreichbaren Zeitzustände von $\mathcal{A}$ wird mit $Reach(\mathcal{A}) \subseteq ZZ$ bezeichnet.

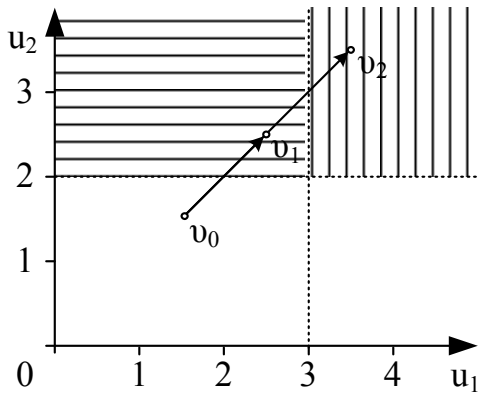


Abbildung 4.2: Scheinbarer Nichtdeterminismus durch Zykluszeiten.

Zeitnichtdeterminismus

Die exakte Zykluszeit jedes einzelnen Zyklus einer SPS hängt von technischen Gegebenheiten ab und ist *im Einzelfall nicht vorhersagbar*. Innerhalb von Grenzen verbleibt für die Zykluszeit deshalb ein Spielraum, der durch die SPS nicht beeinflusst werden kann.

Diskrete Zustandswechsel können von der tatsächlichen Zykluszeit abhängen, was zu unterschiedlichem Verhalten einer Steuerung ausgehend vom selben Zeitzustand führt. Dadurch kann auch ein deterministischer SPS-Zeitautomat ein scheinbar nichtdeterministisches Verhalten aufweisen.

Dieser Effekt kann sowohl bei deterministischen als auch bei nichtdeterministischen SPS-Zeitautomaten auftreten und wird als in Abgrenzung zu gewolltem Nichtdeterminismus als *Zeitnichtdeterminismus* bezeichnet. Das folgende Beispiel verdeutlicht die Entstehung von Zeitdeterminismus.

Beispiel 4.13. *In Abbildung 4.2 ist ein Uhrenstandsdiagramm für einen Zustand z eines SPS-Zeitautomaten $\mathcal{A}$ abgebildet. Der SPS-Zeitautomat $\mathcal{A}$ wechselt vom Zustand z in den Zustand z′, wenn die Schaltbedingung* $u_1 < 3 \wedge u_2 \geq 2$ *gilt. Die Uhrenstände, die diese Schaltbedingung erfüllen, wurden in Abbildung 4.2 horizontal schraffiert dargestellt. $\mathcal{A}$ wechselt aus z in den Zustand z″, wenn* $u_1 \geq 3 \wedge u_2 \geq 2$ *gilt. Die entsprechenden Uhrenstände wurden vertikal schraffiert.*

Befindet sich der SPS-Zeitautomat $\mathcal{A}$ im Zeitzustand $zz_0 = \langle z, \upsilon_0 \rangle$, so ist bei einer Zykluszeit $\tau = 1,0$ von exakt einer Zeiteinheit der Zeitzustand $zz_1 = \langle z, \upsilon_1 \rangle$ Nachfolger von zz_0, bei einer Zykluszeit $\tau = 2,0$ von exakt zwei Zeiteinheiten der Zeitzustand $zz_2 = \langle z, \upsilon_2 \rangle$. Unter der Annahme, dass keine Uhren zurückgesetzt werden, wechselt $\mathcal{A}$ vom Zeitzustand zz_1 in einen Zeitzustand $\langle z', \upsilon_1 + \tau \rangle$ bei einer Zykluszeit $\tau \in \mathbb{Q}_{>0}$ und vom Zeitzustand zz_2 in einen Zeitzustand $\langle z'', \upsilon_2 + \tau \rangle$.

Ausgehend vom Zeitzustand zz_0 bestimmt die tatsächliche Zykluszeit, ob der SPS-Zeitautomat

$\mathcal{A}$ den diskreten Zustand z' oder z'' erreicht. Somit können diskrete Zustandswechsel von der Zykluszeit abhängen.

Zeitdeterminismus ist eine Eigenschaft aller realen SPS und damit der von einer SPS ausgeführten Modelle. Diese Eigenschaft unterscheidet SPS-Zeitautomaten von Zeitautomaten nach Alur/Dill, die keine Zykluszeit besitzen.

4.1.3 Beschränkte SPS-Zeitautomaten

Bisher wurden SPS-Zeitautomaten betrachtet, deren Zykluszeit beliebig groß oder, solange positiv, beliebig klein sein kann. In diesem Unterabschnitt wird untersucht, welche Auswirkungen eine Beschränkung der Zykluszeit auf das Verhalten von SPS-Zeitautomaten hat.

Definition 4.14 (Beschränkter SPS-Zeitautomat)**.** *Für eine beliebige untere Schranke $\delta \in \mathbb{Q}_{>0}$, eine obere Schranke $\Delta \in \mathbb{Q}_{>0}$ mit $\delta < \Delta$ und einen SPS-Zeitautomaten $\mathcal{A} = \langle \Sigma^e, \Sigma^a, Z, z_0, U, T \rangle$ wird ein Tupel $\mathcal{A}_\delta^\Delta := \langle \mathcal{A}, \delta, \Delta \rangle$ als* beschränkter SPS-Zeitautomat *bezeichnet.*

Die Semantik eines beschränkten SPS-Zeitautomaten $\mathcal{A}_\delta^\Delta$ wird durch ein Transitionssystem $\mathcal{T}_\delta^\Delta := \langle ZZ, ZZ_0, M, TR_\delta^\Delta \rangle$ definiert, wobei die Zeitzustandsmengen ZZ, ZZ_0 sowie die Markierungen M den Zeitzustandsmengen und Markierungen des Transitionssystems $\mathcal{T}$ von $\mathcal{A}$ entsprechen. Der Zeitvorlauf eines Zeitzustandes $zz \in ZZ$ und die Transitionen von $\mathcal{T}_\delta^\Delta$ werden analog zu unbeschränkten SPS-Zeitautomaten definiert, wobei die Zykluszeit durch δ und Δ beschränkt wird. Für eine untere Schranke δ und eine obere Schranke Δ und für einen Zeitzustand $zz = \langle z, \upsilon \rangle$ ist die Menge der Zeitnachfolger von zz als

$$\mathbf{post}_\delta^\Delta(zz) := \{\langle z, \upsilon + \tau \rangle : \tau \in \mathbb{Q}_{>0} \wedge \delta < \tau < \Delta\}$$

definiert. Die von zz ausgehenden Transitionen, die Zeitnachfolger beschreiben, sind:

$$TR_\delta^\Delta\nearrow(zz) := \left\{ zz \xrightarrow{\Sigma^{e'}, \tau, \varnothing} zz' : zz' \in \mathbf{post}_\delta^\Delta(zz) \wedge \forall \langle z, B, U_R, \Sigma^{a'}, z' \rangle \in T.\ \Sigma^{e'}, \upsilon \not\models B \right\}$$

Entsprechend ist für einen Zustandsübergang $t = \langle z, B, U_R, \Sigma^{a'}, z' \rangle$ die Menge der Nachfolger als

$$\mathbf{post}_\delta^\Delta(zz, t) := \{\langle z', \upsilon[U_R := 0] + \tau \rangle : \tau \in \mathbb{Q}_{>0} \wedge \delta < \tau < \Delta\}$$

definiert. Alle von zz ausgehenden Transitionen, die Zustandsübergänge beschreiben, sind:

$$TR_\delta^\Delta(zz) := \left\{ zz \xrightarrow{\Sigma^{e'}, \tau, \Sigma^{a'}} zz' : \exists t = \langle z, B, U_R, \Sigma^{a'}, z' \rangle \in T.\ \Sigma^{e'}, \upsilon \models B \wedge zz' \in \mathbf{post}_\delta^\Delta(zz, t) \right\}$$

Die Menge aller Transitionen TR_δ^Δ von T_δ^Δ ergibt sich aus der Vereinigung der Transitionen aller Zeitzustände:

$$TR_\delta^\Delta := \bigcup_{zz \in ZZ} TR_\delta^\Delta(zz) \cup TR_\delta^\Delta\nearrow(zz)$$

Satz 4.15. *Sei $\mathcal{A}$ ein beliebiger SPS-Zeitautomat. Für die beschränkten SPS-Zeitautomaten $\mathcal{A}_{\delta_1}^{\Delta_1}$ und $\mathcal{A}_{\delta_2}^{\Delta_2}$ gilt: Wenn $\delta_1 > \delta_2$ und $\Delta_1 < \Delta_2$ gilt, dann gilt $Reach(\mathcal{A}_{\delta_1}^{\Delta_1}) \subset Reach(\mathcal{A}_{\delta_2}^{\Delta_2})$.*

Beweis. Aus der Definition von TR_δ^Δ und $\delta_1 > \delta_2$ sowie $\Delta_1 < \Delta_2$ folgt $TR_{\delta_1}^{\Delta_1} \subset TR_{\delta_2}^{\Delta_2}$ und damit Satz 4.15. □

Anmerkung 4.16. *Die Umkehrung von Satz 4.15 gilt nicht. Ein einfaches Gegenbeispiel kann folgendermaßen konstruiert werden: Sei $\mathcal{A} = \langle \varnothing, \varnothing, \{z_0\}, z_0, \{u_0\}, \varnothing \rangle$ ein unbeschränkter SPS-Zeitautomat, der einen Zustand, eine Uhr und keine Zeitbeschränkungen besitzt. Sind $\mathcal{A}_1^2 = \langle \mathcal{A}, 1, 2 \rangle$ und $\mathcal{A}_2^4 = \langle \mathcal{A}, 2, 4 \rangle$ zwei beschränkte SPS-Zeitautomaten, so gilt*

$$Reach(\mathcal{A}_2^4) = \{\langle z_0, \upsilon \rangle : \upsilon(u_0) > 2\} \subset Reach(\mathcal{A}_1^2) = \{\langle z_0, \upsilon \rangle : \upsilon(u_0) > 1\},$$

aber nicht $4 < 2$. Im folgenden Satz wird deshalb eine abgeschwächte Variante der Umkehrung von Satz 4.15 bewiesen.

Satz 4.17. *Sei $\mathcal{A}$ ein beliebiger SPS-Zeitautomat. Für die beschränkten SPS-Zeitautomaten $\mathcal{A}_{\delta_1}^{\Delta_1}$ und $\mathcal{A}_{\delta_2}^{\Delta_2}$ gilt: Wenn $Reach(\mathcal{A}_{\delta_1}^{\Delta_1}) \subset Reach(\mathcal{A}_{\delta_2}^{\Delta_2})$ gilt, dann folgt $\delta_1 > \delta_2$ oder $\Delta_1 < \Delta_2$.*

Beweis. Sei $\mathcal{A}$ ein beliebiger SPS-Zeitautomat und es gelte für die beschränkten SPS-Zeitautomaten $\mathcal{A}_{\delta_1}^{\Delta_1}$ und $\mathcal{A}_{\delta_2}^{\Delta_2}$: $Reach(\mathcal{A}_{\delta_1}^{\Delta_1}) \subset Reach(\mathcal{A}_{\delta_2}^{\Delta_2})$. Dann folgt: Es gibt einen Zeitzustand zz mit $zz \in Reach(\mathcal{A}_{\delta_2}^{\Delta_2})$ und $zz \notin Reach(\mathcal{A}_{\delta_1}^{\Delta_1})$.

Aus $zz \in Reach(\mathcal{A}_{\delta_2}^{\Delta_2})$ folgt: Es gibt einen Lauf $r = zz_0 \xrightarrow{\Sigma_0^{e'}, \tau_0, \Sigma_0^{a'}} zz_1 \ldots zz_k \ldots$ für $\mathcal{A}_{\delta_2}^{\Delta_2}$ und ein $k \in \mathbb{N}$, so dass $zz = zz_k$. Da $zz \notin Reach(\mathcal{A}_{\delta_1}^{\Delta_1})$ kann der Lauf r nicht für $\mathcal{A}_{\delta_1}^{\Delta_1}$ gelten und eine Transition tr_i in r muss existieren, so dass $tr_i \in TR_{\delta_2}^{\Delta_2}$, aber $tr_i \notin TR_{\delta_1}^{\Delta_1}$.

Nach der Definition von TR_δ^Δ gilt $tr_i \in TR_\delta^\Delta \iff tr_i \in TR \wedge \delta < \tau_i < \Delta$. Demnach gilt $tr_i \notin TR_{\delta_1}^{\Delta_1} \iff \neg(tr_i \in TR) \vee \neg(\delta_1 < \tau_i < \Delta_1)$ und $tr_i \in TR_{\delta_2}^{\Delta_2} \iff tr_i \in TR \wedge \delta_2 < \tau_i < \Delta_2$.

Da $tr_i \in TR$ und $tr_i \notin TR_{\delta_1}^{\Delta_1}$ muss $\delta \geq \tau_i$ oder $\tau_i \geq \Delta$ gelten. Da $tr_i \in TR_{\delta_2}^{\Delta_2}$ gilt $\delta_2 < \tau_i < \Delta_2$. Somit muss $\delta_1 > \delta_2$ oder $\Delta_1 < \Delta_2$ gelten.

Damit ist $Reach(\mathcal{A}_{\delta_1}^{\Delta_1}) \subset Reach(\mathcal{A}_{\delta_2}^{\Delta_2}) \Longrightarrow \delta_2 < \delta_1 \vee \Delta_1 < \Delta_2$ bewiesen. □

Es wird angenommen, dass die *Validierung* eines SPS-Zeitautomaten feststellt, ob der SPS-Zeitautomat unerwünschte (Zeit-)Zustände nicht erreichen kann. Satz 4.15 beschreibt einen

für die weitere Arbeit wichtigen Aspekt beschränkter SPS-Zeitautomaten: Wurde ein SPS-Zeitautomat unter Annahme zweier Zeitschranken δ und Δ validiert, so erreicht eine reale Implementierung auch dann keine weiteren Zeitzustände, wenn die tatsächlichen Zykluszeiten schärfer als mit δ und Δ beschränkt werden können. Stellt eine Implementierung eines validen beschränkten SPS-Zeitautomaten sicher, dass ihre tatsächlichen Zykluszeiten zwischen δ und Δ liegen, so ist die Implementierung ebenfalls valid.

4.2 Zonen: Endliche Darstellung von Zeitzuständen

Die Zeitzustände eines SPS-Zeitautomaten setzen sich aus einem (diskreten) Zustand des SPS-Zeitautomaten sowie einem Uhrenstand zusammen. Während die Anzahl diskreter Zustände stets endlich ist, existieren unendlich viele Uhrenstände und somit unendlich viele Zeitzustände.

Zur Analyse von SPS-Zeitautomaten können Uhrenstände symbolisch mit Hilfe von Hyperpolyedern repräsentiert werden. Hyperpolyeder sind bezüglich aller von einer SPS auf Zeitzuständen ausführbarer Operationen abgeschlossen und dabei im Gegensatz zu Regionen unabhängig von der Größe in Schaltbedingungen verwendeter Konstanten.

Die Verbindung eines (diskreten) Zustandes eines Zeitautomaten mit einer Differenzgrenz-Matrix oder Region wird in der Literatur häufig als Zone bezeichnet. Analog dazu wird eine Zone wie folgt definiert:

Definition 4.18 (Zone)**.** *Sei $\mathcal{A} = \langle \Sigma^e, \Sigma^a, Z, z_0, U, T \rangle$ ein SPS-Zeitautomat. Ein Paar $\overline{z} := \langle z, \hbar \rangle$, wobei $z \in Z$ ein Zustand von $\mathcal{A}$ und $\hbar$ ein beliebiger Hyperpolyeder über U ist, wird als* Zone *bezeichnet. Die Menge $[\![\overline{z}]\!] = \{\langle z, \upsilon \rangle : \upsilon \in [\![\hbar]\!]\}$ enthält alle durch $\overline{z}$ dargestellten Zeitzustände von $\mathcal{A}$.*

$\overline{\mathcal{Z}} = \{\overline{z}_1, \ldots, \overline{z}_n\}$ bezeichnet im Weiteren eine Menge von Zonen. Dabei gilt $[\![\overline{\mathcal{Z}}]\!] := \bigcup_{\overline{z} \in \overline{\mathcal{Z}}} [\![\overline{z}]\!]$.

4.2.1 Zustandsübergänge

Um die Zeitzustände eines beschränkten SPS-Zeitautomaten aufzuzählen, werden im folgenden Abschnitt Operationen auf Hyperpolyedern vorgestellt, die es ermöglichen, Zustandsübergänge von SPS-Zeitautomaten nachzubilden.

Zurücksetzen

SPS-Zeitautomaten und die meisten realen Zeitsysteme können ihre Uhren auf 0 zurücksetzen. Die entsprechende Operation für Hyperpolyeder kann direkt aus der Operation für Differenzgrenz-Matrizen abgeleitet werden. Dabei wird angenommen, dass das Zurücksetzen einer Uhr

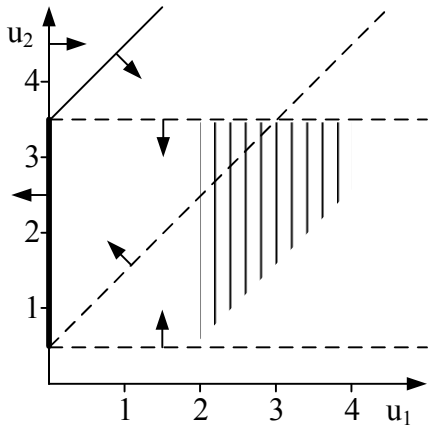


Abbildung 4.3: Zurücksetzen von Uhr u_1.

selbst keine Zeit benötigt. Diese Annahme erleichtert die Abbildung des Zurücksetzens durch Hyperpolyeder und ist für Speicherprogrammierbare Steuerungen implementierbar (vgl. Anmerkung 4.21), trifft aber nicht auf jedes Zeitsystem zu. Bei anderen Zeitsystemen muss gegebenenfalls die für das Zurücksetzen benötigte Zeit und bei mehreren zurückzusetzenden Uhren deren Reihenfolge beachtet werden.

Definition 4.19 (Zurücksetzen)**.** *Sei $U = \{u_1, \ldots, u_n\}$ eine Menge von Uhren, $u_R \in U$ eine beliebige Uhr und $\hbar$ ein beliebiger, nicht leerer Hyperpolyeder über U. Die Uhr u_R wurde in $\hbar$ zurückgesetzt, geschrieben $\hbar[u_R := 0]$, wenn für den Hyperpolyeder $\hbar[u_R := 0]$ und alle Uhrenstände υ gilt: $\upsilon \in [\![\hbar]\!]$ gdw. $\upsilon[u_R := 0] \in [\![\hbar[u_R := 0]]\!]$.*

Beispiel 4.20. *In Abbildung 4.3 wurde für den schraffiert dargestellten Hyperpolyeder $\hbar$ die Uhr u_1 zurueckgesetzt. Es ergeben sich die Beschränkungen*

$$B := \{u_0 - u_2 < -0,5; u_2 - u_0 < 3,5\} \qquad \text{sowie}$$

$$\beta^{0-1} := u_0 - u_1 \leq 0 \qquad \beta^{1-2} := u_1 - u_2 < -0,5$$

$$\beta^{1-0} := u_1 - u_0 \leq 0 \qquad \beta^{2-1} := u_2 - u_1 < 3,5$$

und daraus

$$\hbar[u_1 := 0] = u_0 - u_2 < -0,5 \wedge u_2 - u_0 < 3,5 \wedge \beta^{0-1} \wedge \beta^{1-0} \wedge \beta^{1-2} \wedge \beta^{2-1}$$

Die Uhrenstände, die in $\hbar[u_1 := 0]$ enthalten sind, sind fett gekennzeichnet.

Um eine beliebige Menge $U_R = \{u'_1, \ldots, u'_m\} \subseteq U$ von Uhren in einem Hyperpolyeder $\hbar$ zurückzusetzen, geschrieben $\hbar[U_R := 0]$, kann $\hbar[U_R := 0]$ als

$$\hbar[U_R := 0] := \hbar[u'_1 := 0][u'_2 := 0] \ldots [u_m := 0]$$

dargestellt werden, da $\upsilon[U_R := 0] = \upsilon[u'_1 := 0][u'_2 := 0] \ldots [u_m := 0]$. Die Reihenfolge, in der die Uhren zurückgesetzt werden, ist für das Ergebnis des Zurücksetzens nicht von Bedeutung.

Anmerkung 4.21. *Besitzt eine SPS eine einheitliche physikalische Uhr und mehrere logische Uhren, so können diese gemäß Abschnitt 3.1.4 zeitgleich zurückgesetzt werden.*

Zeitschritt

SPS-Zeitautomaten durchschreiten die Zeit nicht kontinuierlich, sondern schrittweise. Diesem Umstand wird Rechnung getragen, indem der Zeitfortschritt für SPS-Zeitautomaten als *Zeitschritt* als Operation auf Hyperpolyedern modelliert wird.

Das Einführen einer expliziten Operation auf Hyperpolyedern hat den Vorteil, dass keine weiteren Uhren zur Modellierung des Zeitfortschritts für Steuerungen mit zyklischer Arbeitsweise benötigt werden. Das wäre notwendig, um den Zeitfortschritt solcher Systeme beispielsweise mit Zeitautomaten nach Alur/Dill zu modellieren.

Durch das Einführen einer Operation für den Zeitfortschritt ist es zudem denkbar, durch geeignetes Zusammenfassen aufeinanderfolgender Zeitschritte die Analyse zu optimieren. Dieser Ansatz wird im Folgenden allerdings nicht weiter verfolgt.

Schließlich sind im Gegensatz zu Zeitautomaten nach Alur/Dill nicht alle Zeitnachfolger eines Zeitzustandes für SPS-Zeitautomaten stets erreichbar. Die Partitionierung von Hyperpolyeder dient der Zerlegung eines Hyperpolyeders in mehrere Teile, wenn einige dieser Teile keine Zeitnachfolger haben könnten.

Der Zeitschritt eines Hyperpolyeders $\hbar$ umfasst alle Uhrenstände, die innerhalb eines Schaltzyklus von einem Uhrenstand in $\hbar$ erreicht werden können:

Definition 4.22 (Zeitschritt)**.** *Sei $\hbar$ ein beliebiger, nicht leerer Hyperpolyeder, $\delta \in \mathbb{Q}_{>0}$ eine untere Zykluszeitschranke und $\Delta \in \mathbb{Q}$ eine obere Zykluszeitschranke, wobei $\delta < \Delta$. Ein Hyperpolyeder $\hbar\nearrow_{\delta}^{\Delta}$ wird als Zeitschritt von $\hbar$ bezeichnet, wenn gilt:*

$$[\![\hbar\nearrow_{\delta}^{\Delta}]\!] = \left\{\upsilon : \exists \tau \in \mathbb{Q}.\ \delta < \tau < \Delta \wedge \upsilon - \tau \in [\![\hbar]\!]\right\}$$

Beispiel 4.23. *In Abbildung 4.4 sind die Uhrenstände des Hyperpolyeders*

$$\hbar = u_0 - u_1 \leq -1 \wedge u_2 - u_0 < 1,5 \wedge u_1 - u_2 \leq 0,5$$

vertikal schraffiert dargestellt. Für $\delta = 1,5$ und $\Delta = 2$ sind die Uhrenstände des zugehörigen Zeitschritts

$$\hbar\nearrow_{\delta}^{\Delta} = u_0 - u_1 < -2,5 \wedge u_0 - u_2 < -2 \wedge u_1 - u_0 < 4 \wedge u_2 - u_0 < 3,5 \wedge u_1 - u_2 \leq 0,5 \wedge u_2 - u_1 \leq 0,5$$

horizontal schraffiert.

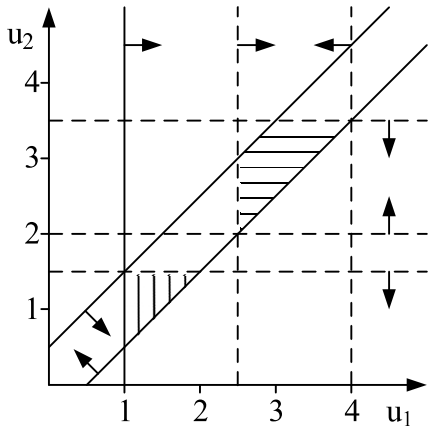


Abbildung 4.4: Zeitschritt eines Hyperpolyeders mit $\delta = 1,5$ und $\Delta = 2$.

Mit den Operationen Zurücksetzen und Zeitschritt wurden die Operationen definiert, die zur Aufzählung der erreichbaren Zeitzustände eines SPS-Zeitautomaten notwendig sind. Durch die Operation Zeitschritt werden die Uhrzeiten ermittelt, die von einem SPS-Zeitautomaten in einem Zyklus erreichbar sind. Durch die Kombination von Zurücksetzen und Zeitschritt kann im folgenden Abschnitt der Zustandsübergang eines SPS-Zeitautomaten nachgebildet werden.

Folgezonen

Die Zeitzustände, die durch die Ausführung eines Zustandsüberganges eines SPS-Zeitautomaten erreicht werden, bilden eine Folgezone im neuen Zustand. Existiert für eine Zone außerdem eine Belegung der Eingänge, so dass keine gültige Schaltbedingung existiert, besitzt die Zone zudem eine Folgezone im gleichen Zustand.

Definition 4.24 (Zeitfolgezone)**.** *Sei $\mathcal{A} = \langle \Sigma^e, \Sigma^a, Z, z_0, U, T, \delta, \Delta \rangle$ ein beschränkter SPS-Zeitautomat und $\overline{z} = \langle z, \hbar \rangle$ eine Zone. Die Zone $\overline{z}\nearrow_\delta^\Delta := \langle z, \hbar \nearrow_\delta^\Delta \rangle$ wird als Zeitfolgezone von $\overline{z}$ bezeichnet.*

Laut der Definition der Semantik beschränkter SPS-Zeitautomaten ist die Menge der Nachfolger eines Zeitzustandes als $\mathbf{post}_\delta^\Delta(zz) = \{\langle z, \upsilon + \tau \rangle : \tau \in \mathbb{Q}_{>0} \wedge \delta < \tau < \Delta\}$ definiert. Es gilt folgender Satz:

Satz 4.25 (Korrektheit Zeitfolgezone)**.** *Sei $\mathcal{A} = \langle \Sigma^e, \Sigma^a, Z, z_0, U, T, \delta, \Delta \rangle$ ein beschränkter SPS-Zeitautomat und $\overline{z} = \langle z, \hbar \rangle$ eine Zone. Dann gilt*

$$[\![\overline{z}\nearrow_\delta^\Delta]\!] = \bigcup_{zz \in [\![\overline{z}]\!]} \mathbf{post}_\delta^\Delta(zz)$$

Beweis. Satz 4.25 folgt direkt aus Definition 4.22. □

Die Folgezone für einen Zustandsübergang ist entsprechend definiert.

Definition 4.26 (*t*-Folgezone)**.** *Sei $\mathcal{A} = \langle \Sigma^e, \Sigma^a, Z, z_0, U, T, \delta, \Delta \rangle$ ein beschränkter SPS-Zeitautomat und $t = \langle z, B^u \wedge B^e, U_R, E', z' \rangle \in T$ eine Transition. Für eine Zone $\langle z, \hbar \rangle$, wobei $\hbar \trianglelefteq B^u$, wird die Zone $\overline{z}_t := \langle z, \hbar[U_R := 0]\nearrow_\delta^\Delta \rangle$ als t-Folgezone bezeichnet.*

Die Menge $\mathbf{post}_\delta^\Delta(zz, t) = \{\langle z', \upsilon[U_R := 0] + \tau \rangle : \tau \in \mathbb{Q}_{>0} \wedge \delta < \tau < \Delta\}$ enthält die Nachfolger eines Zeitzustands *zz* eines beschränkten SPS-Zeitautomaten. Die Korrektheit von *t*-Folgezonen wird durch den folgenden Satz sichergestellt.

Satz 4.27 (Korrektheit *t*-Folgezone)**.** *Sei $\mathcal{A} = \langle \Sigma^e, \Sigma^a, Z, z_0, U, T, \delta, \Delta \rangle$ ein beschränkter SPS-Zeitautomat und $\overline{z} = \langle z, \hbar \rangle$ eine Zone. Dann gilt*

$$[\![\overline{z}_t]\!] = \bigcup_{zz \in [\![\overline{z}]\!]} \mathbf{post}_\delta^\Delta(zz, t)$$

Beweis. Die Behauptung folgt aus den Definitionen 4.19 und 4.22. □

4.2.2 Partitionierung

Auf die Zeitzustände einer Zone können möglicherweise verschiedene, sich eventuell überschneidende Schaltbedingungen zutreffen. Dabei können einzelne Schaltbedingungen oder Kombinationen von Schaltbedingungen dazu führen, dass manche Zeitzustände einen Zeitnachfolger besitzen, andere jedoch nicht. Um zu gewährleisten, dass die Zeitzustände einer Zone, die unterschiedlichen Schaltbedingungen genügen, auch unterschiedlich behandelt werden, ist es notwendig, Zonen zu zerlegen.

Definition 4.28 (Partitionierung)**.** *Seien $\hbar$ und $\hbar'$ zwei Hyperpolyeder. Eine endliche Menge $\mathcal{H} = \{\hbar_1, \ldots, \hbar_n\}$ von Hyperpolyedern wird als Partitionierung von $\hbar$ durch $\hbar'$ bezeichnet, wenn*

1. $[\![\hbar]\!] = \bigcup_{\hbar_i \in \mathcal{H}} [\![\hbar_i]\!]$ *gilt und*
2. *für alle $i, j \in \{1, \ldots, n\}$ mit $i \neq j$ die Aussage $\hbar_i \cap \hbar_j = \varnothing$ gilt und*
3. *für jedes $\hbar_i \in \mathcal{H}$ entweder $[\![\hbar_i]\!] \subseteq [\![\hbar']\!]$ oder $[\![\hbar_i]\!] \cap [\![\hbar']\!] = \varnothing$ gilt.*

Es bestehen verschiedene Möglichkeiten, einen Hyperpolyeder zu partitionieren. Die Art der Partitionierung bestimmt die Anzahl entstehender Hyperpolyeder, die bei ungünstigen Verfahren exponentiell in der Anzahl der Beschränkungen der Hyperpolyeder wachsen können. Die im Folgenden vorgeschlagene Partitionierung erzeugt höchstens so viele Hyperpolyeder wie $\hbar'$ Beschränkungen besitzt.

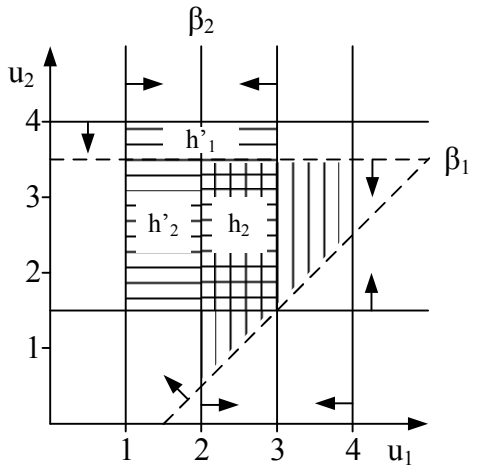


Abbildung 4.5: Partitionierung eines Hyperpolyeders.

Beispiel 4.29. *In Abbildung 4.5 ist der Hyperpolyeder*

$$\hbar = u_0 - u_1 \leq -1 \wedge u_1 - u_0 \leq 3 \wedge u_0 - u_2 \leq -1,5 \wedge u_2 - u_0 \leq 4$$

von links unten nach recht oben schraffiert und der Hyperpolyeder

$$\hbar' = u_0 - u_1 \leq -2 \wedge u_1 - u_0 \leq 4 \wedge u_2 - u_0 < 3,5 \wedge u_1 - u_2 < 1,5$$

von links oben nach rechts unten schraffiert dargestellt. Der Hyperpolyeder $\hbar$ wird von den in der Abbildung markierten Beschränkungen $B = \{\beta_1, \beta_2\}$ geschnitten. Bei der Partitionierung von $\hbar$ durch $\hbar'$ entstehen folgende Hyperpolyeder:

$$\hbar'_1 = u_0 - u_1 \leq -1 \wedge u_1 - u_0 \leq 3 \wedge u_0 - u_2 \leq -3,5 \wedge u_2 - u_0 \leq 4$$
$$\hbar'_2 = u_0 - u_1 \leq -1 \wedge u_1 - u_0 < 2 \wedge u_0 - u_2 \leq -1,5 \wedge u_2 - u_0 < 3,5$$
$$\hbar_2 = u_0 - u_1 \leq -2 \wedge u_1 - u_0 \leq 3 \wedge u_0 - u_2 \leq -1,5 \wedge u_2 - u_0 < 3,5$$

Satz 4.30 (Partitionierung)**.** *Seien $\hbar$ und $\hbar'$ zwei Hyperpolyeder. Sei ferner*

$$B := \{\beta_i \in \hbar' : \beta_i \cap \hbar \neq \hbar \wedge \beta_i \cap \hbar \neq \varnothing\}$$

die Menge der Beschränkungen von $\hbar'$, die $\hbar$ schneiden, wobei die Beschränkungen von B von β_1 bis β_n durchnummeriert seien.
Sei ferner $\hbar_0 = \hbar, \hbar_i = \hbar_{i-1} \cap \beta_i$ und $\hbar'_i = \hbar_{i-1} \cap \complement\beta_i, i = 1, \ldots, n$. Dann ist die Menge

$$\mathbf{part}_{\hbar'}(\hbar) = \{\hbar'_1, \ldots, \hbar'_n, \hbar_n\}$$

eine Partitionierung.

Beweis. Es werden die drei Aussagen von Definition 4.28 bewiesen:

1. Es gilt (*i*): $[\![\hbar_{i-1}]\!] = [\![\hbar'_i]\!] \cup [\![\hbar_i]\!]$, da

$$\begin{aligned} & [\![\hbar'_i]\!] \cup [\![\hbar_i]\!] \\ =& [\![\hbar_{i-1} \cap \complement\beta^i]\!] \cup [\![\hbar_{i-1} \cap \beta^i]\!] && \text{gemäß Satz 4.30} \\ =& [\![\hbar_{i-1}]\!] && \text{gemäß Definition 3.22 und Definition 3.10} \end{aligned}$$

   Durch Einsetzen von $[\![\hbar_1]\!] = [\![\hbar'_2]\!] \cup [\![\hbar_2]\!]$ in $[\![\hbar_0]\!] = [\![\hbar'_1]\!] \cup [\![\hbar_1]\!]$ erhält man die Gleichung $[\![\hbar_0]\!] = [\![\hbar'_1]\!] \cup [\![\hbar'_2]\!] \cup [\![\hbar_2]\!]$. Mit Induktion folgt

$$[\![\hbar_0]\!] = [\![\hbar'_1]\!] \cup [\![\hbar'_2]\!] \cup \ldots \cup [\![\hbar'_n]\!] \cup [\![\hbar_n]\!]$$

2. Laut der Definition des Komplements (Def. 3.10) gilt $[\![\beta]\!] \cap [\![\complement\beta]\!] = \varnothing$ und somit (*ii*): $[\![\hbar'_i]\!] \cap [\![\hbar_i]\!] = \varnothing$. Mit (*i*) und (*ii*) folgt die Behauptung für alle Hyperpolyeder in $\mathbf{part}_{\hbar'}(\hbar)$.
3. Für alle $\hbar'_i$ gilt $[\![\hbar'_i]\!] \cap [\![\hbar']\!] = \varnothing$, da β^i Beschränkungen von $\hbar'$ sind. Der Hyperpolyeder $\hbar_n$ ist schärfer als alle Beschränkungen in B, da er nur mit Beschränkungen von $\hbar'$ geschnitten wurde. Laut der Definition von B gilt für alle Beschränkungen $\beta \in \hbar'$, die nicht in B enthalten sind entweder $[\![\hbar_n]\!] \subseteq [\![\beta]\!]$ oder $[\![\hbar_n]\!] \cap [\![\beta]\!] = \varnothing$. Somit gilt $[\![\hbar_n]\!] \subseteq [\![\hbar']\!]$ gdw. für alle Beschränkungen $\beta \in \hbar', \beta \notin B$ gilt: $[\![\hbar_n]\!] \subseteq [\![\beta]\!]$ ansonsten gilt $[\![\hbar_n]\!] \cap [\![\beta]\!] = \varnothing$.

□

Anmerkung 4.31. *Die Partitionierung eines Hyperpolyeders $\hbar$ bezüglich $\hbar'$ enthält mindestens einen Hyperpolyeder, nämlich $\hbar$ und höchstens $n(n+1)$ Hyperpolyeder, wobei n die Anzahl von Uhren ist. Das ist dadurch zu erklären, dass die Partitionierung genau soviele Hyperpolyeder enthält wie Beschränkungen von $\hbar'$ den Hyperpolyeder $\hbar$ schneiden. Es kann deshalb von Vorteil sein, den minimalen Hyperpolyeder von $\hbar'$ vor der Partitionierung zu bestimmen, um so die Anzahl von Beschränkungen und entstehender Hyperpolyeder zu reduzieren.*

Aus der Partitionierung eines Hyperpolyeders kann die Partitionierung einer Zone abgeleitet werden.

Definition 4.32 (Zonen-Partitionierung)**.** *Für einen SPS-Zeitautomaten $\mathcal{A}$ sei $\overline{z} = \langle z, \hbar \rangle$ eine Zone und $\mathcal{H} = \{\hbar_1, \ldots, \hbar_n\}$ eine endliche Menge von Hyperpolyedern. Eine endliche Menge $\mathcal{Z} = \{\langle z, \hbar'_1 \rangle, \ldots, \langle z, \hbar'_m \rangle\}$ wird als Partitionierung von $\overline{z}$ bezüglich $\mathcal{H}$ bezeichnet, wenn die Menge $\mathcal{H}' = \{\hbar'_1, \ldots \hbar'_m\}$ eine Partitionierung von $\hbar$ für jedes $\hbar_i$ mit $1 < i < n$ ist.*

Für den folgenden Satz wird angenommen, dass die Partition einer Menge von Hyperpolyedern als $\mathbf{part}_{\hbar'}(\mathcal{H}) \coloneqq \bigcup_{\hbar'' \in \mathcal{H}} \mathbf{part}_{\hbar'}(\hbar'')$ definiert ist.

Satz 4.33. *Für eine Zone $\overline{z} = \langle z, \hbar \rangle$ und eine Menge von Hyperpolyedern $\mathcal{H} = \{\hbar_1, \ldots, \hbar_n\}$ ist die Menge*

$$\mathbf{part}_{\{\hbar_1,\ldots,\hbar_n\}}(\langle z, \hbar \rangle) = \{\langle z, \hbar' \rangle : \hbar' \in \mathbf{part}_{\hbar_1}(\mathbf{part}_{\hbar_2}(\ldots \mathbf{part}_{\hbar_n}(\hbar) \ldots))\}$$

eine Partitionierung von $\overline{z}$ bezüglich $\mathcal{H}$.

Beweis. Es ist zu zeigen, dass die Menge $\mathcal{H}' = \mathbf{part}_{\hbar_1}(\mathbf{part}_{\hbar_2}(\ldots \mathbf{part}_{\hbar_n}(\hbar)\ldots))$ Partitionierung für jeden Hyperpolyeder in $\mathcal{H}$ ist. Es werden die drei Anforderungen von Definition 4.28 bewiesen.

1. Folgt aus $A \cup B = C \wedge B = B' \cup B'' \Longrightarrow A \cup (B' \cup B'') = C$.
2. Folgt aus $A \cap B = \varnothing \wedge B = B' \cup B'' \Longrightarrow A \cap B' = \varnothing$.
3. Laut Definition 4.28 gilt für jeden $\hbar' \in \mathbf{part}_{\hbar_n}(\hbar)$ entweder $[\![\hbar']\!] \subseteq [\![\hbar_n]\!]$ oder $[\![\hbar']\!] \cap [\![\hbar_n]\!] = \varnothing$. Für jeden $\hbar'' \in \mathbf{part}_{\hbar_{n-1}}(\hbar')$ gilt laut 1. $[\![\hbar'']\!] \subseteq [\![\hbar']\!]$ und es folgt entweder $[\![\hbar'']\!] \subseteq [\![\hbar_n]\!]$ oder $[\![\hbar'']\!] \cap [\![\hbar_n]\!] = \varnothing$. Induktiv kann die Forderung für alle Hyperpolyeder in $\mathcal{H}$ gefolgert werden.

□

Anmerkung 4.34. *Eine Partitionierung einer Zone $\langle z, \hbar\rangle$ bezüglich aller Zeitbedingungen $\mathcal{H} = \{B^u : \langle z, B^u \wedge B^e, U_R, E', z'\rangle \in T\}$ eines Zeitautomaten wird als* $\mathbf{part}(\langle z, \hbar\rangle)$ *geschrieben.*

4.2.3 Zonengraphen

Für einen Modellprüfer ist es meist ausreichend, zu ermitteln, welche symbolisch durch Zonen repräsentierte Zeitzustände ein SPS-Zeitautomat erreichen kann, um festzustellen, ob einer der erreichten Zeitzustände eine Modellanforderung verletzt.

Beim Validieren einer Implementierung liegen mit Modell und Implementierung zwei Zeitsysteme vor, die miteinander verglichen werden sollen. Um das Verhalten dieser Zeitsysteme symbolisch abzubilden, müssen neben den Zeitzuständen auch die Zustandsübergänge erfasst werden. Zu diesem Zweck werden Zonengraphen definiert und um eine Markierung für die Ein- und Ausgabe eines SPS-Zeitautomaten ergänzt.

Definition 4.35 (Zonengraph)**.** *Sei $\mathcal{A} = \langle \Sigma^e, \Sigma^a, Z, z_0, U, T\rangle$ ein SPS-Zeitautomat. Als Zonengraph wird ein Tupel $zg := \langle \underline{\mathcal{Z}}, \Sigma^e, \Sigma^a, \rightarrow \rangle$ bezeichnet, wobei $\underline{\mathcal{Z}}$ eine endliche Menge von Zonen von $\mathcal{A}$, Σ^e und Σ^a die Ein- und Ausgänge von $\mathcal{A}$ und $\rightarrow \subseteq \underline{\mathcal{Z}} \times \mathcal{P}(\Sigma^e) \times \mathcal{P}(U) \times \mathcal{P}(\Sigma^a) \times \underline{\mathcal{Z}}$ eine Menge mit Eingangs- und Ausgangsbelegungen beschrifteter Kanten ist.*

4.3 Analyse von SPS-Zeitautomaten

Die Aufzählung der Zeitzustände eines Zeitsystems kann unterschiedlichen Zielstellungen folgen. Es kann eine Minimierung oder Optimierung, beispielsweise durch die Elimination nicht erreichbarer Zustände, angestrebt werden. Die Aufzählung kann ferner dem Vergleich von Zeitsystemen oder der Prüfung eines Modells dienen.

Je nach Zweck unterscheiden sich die zur Aufzählung der Zeitzustände verwendeten Algorithmen. Für das letztgenannte Ziel der Prüfung eines Modells genügt es meist, festzustellen,

ob ein vorgegebener unerwünschter Zeitzustand erreicht werden kann. Dieses Ziel ist am einfachsten mit der Rückwärtsanalyse zu erreichen.

4.3.1 Rückwärtsanalyse

Die Rückwärtsanalyse zählt ausgehend von einem vorgegebenen Zeitzustand solange die Vorgänger des Zeitzustandes auf, bis entweder der Startzuständ eines Zeitsystems erreicht oder alle Vorgänger aufgezählt wurden. Oft besitzt ein Zeitzustand im Vergleich zur Anzahl erreichbarer Zustände eines Zeitsystems wenige Vorgänger, wodurch die Rückwärtsanalyse effizient ist.

Dem entgegen werden zur Minimierung oder zum Vergleich von Zeitsystemen meist alle erreichbaren Zeitzustände eines Zeitsystems benötigt. Diese können mit der Vorwärtsanalyse ermittelt werden.

4.3.2 Vorwärtsanalyse

Bei der Vorwärtsanalyse werden beginnend mit dem Startzeitzustand eines Zeitautomaten alle Folgezeitzustände aufgezählt. Die Zeitzustände werden symbolisch repräsentiert. Die Aufzählung endet, wenn ein gesuchter Zeitzustand erreicht oder alle erreichbaren Zeitzustände aufgezählt wurden. Die Vorwärtsanalyse ist ein Standardalgorithmus, der in vielen Modellprüfern Verwendung findet [BY04].

Der Algorithmus der Vorwärtsanalyse, welcher für Zeitautomaten nach Alur/Dill eingesetzt wird, muss modifiziert werden, da beschränkte SPS-Zeitautomaten folgende, für die Vorwärtsanalyse wichtige Besonderheiten aufweisen:

1. Obere und untere Schranken begrenzen die Menge der Nachfolger eines beschränkten SPS-Zeitautomaten.
2. Die Eingabe eines SPS-Zeitautomaten ist eine Belegung der Eingänge des SPS-Zeitautomaten. Damit kann für einige Zeitzustände gelten, dass ein SPS-Zeitautomat für jede Belegung seiner Eingänge einen Zustandsübergang ausführt. Derartige Zeitzustände besitzen keine Nachfolger im gleichen (diskreten) Zustand, was der Einführung von ϵ-Transitionen in Zeitautomaten ähnelt [BGP96, DGP97, BPDG98].

Beispiel 4.36. *In Abbildung 4.6 ist ein SPS-Zeitautomat dargestellt, der drei Zustände, eine Uhr und einen Eingang σ_1^e besitzt. Die Schaltbedingungen der Zustandsübergänge des SPS-Zeitautomaten sind so konstruiert, dass der SPS-Zeitautomat vom Zustand* 0 *in den Zustand* 1 *wechselt, wenn der Eingang σ_1^e aktiv ist und in den Zustand* 2 *wechselt, wenn σ_1^e inaktiv ist. Demnach hat der Startzeitzustand im Zustand* 0 *keinen Nachfolger.*

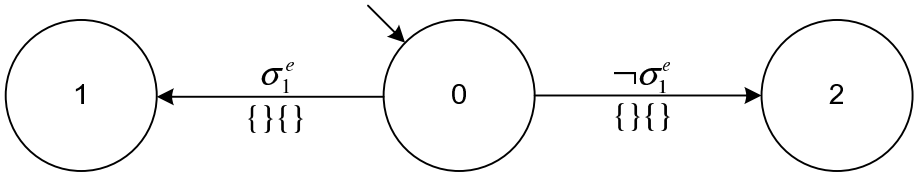


Abbildung 4.6: Ein SPS-Zeitautomat.

Algorithmus 1 stellt eine Variante der Vorwärtsanalyse vor, die für einen gegebenen beschränkten SPS-Zeitautomaten $\mathcal{A} = \langle \Sigma^e, \Sigma^a, Z, z_0, U, T, \delta, \Delta \rangle$ einen Zonengraph *zg* ermittelt, der das zu $\mathcal{A}$ gehörende Transitionssystem $\mathcal{T}$ repräsentiert.

Funktionsweise

Der Algorithmus baut zwei Zonenmengen auf:

$\bar{\mathcal{Z}}_B$ Enthält erreichbare Zonen, die bereits analysiert wurden („besuchte Zonen“).

$\bar{\mathcal{Z}}_W$ Enthält erreichbare Zonen, die noch nicht analysiert wurden („wartende Zonen“).

Die Kanten des Zonengraphen werden zusätzlich in der Relation $\rightarrow$ abgelegt.

Die Menge $\bar{\mathcal{Z}}_W$ enthält anfangs eine den Startzeitzustand repräsentierende Zone (Zeile 5). Solange Zonen existieren, die noch nicht analysiert wurden, wird eine dieser Zonen $\bar{z}$ aus $\bar{\mathcal{Z}}_W$ entnommen (Zeile 8), analysiert und in $\bar{\mathcal{Z}}_B$ abgelegt. Der Algorithmus endet, sobald keine Zonen mehr existieren, die nicht bereits analysiert wurden. Es wird also der *Fixpunkt* der Menge $\bar{\mathcal{Z}}_B$ gesucht.

Nach der Entnahme von $\bar{z}$ aus $\bar{\mathcal{Z}}_W$ wird geprüft, ob alle Zeitzustände von $\bar{z}$ zwischenzeitlich bereits besucht worden sind (Zeile 10). Danach weicht die Vorwärtsanalyse für SPS-Zeitautomaten von der Vorwärtsanalyse für Zeitautomaten nach Alur/Dill ab: Erfolgt für einen Zeitzustand *zz* und jede Eingangsbelegung ein Zustandswechsel, besitzt *zz keinen* Zeitnachfolger im gleichen Zustand.

Um das zu gewährleisten, wird in Zeile 11 die Menge $\bar{\Sigma}$ aller Eingangsbelegungen ermittelt. Anschließend werden in den Zeilen 13-21 alle diskreten Zustandsübergänge ermittelt und in $\bar{\mathcal{Z}}_W$ abgelegt (Zeile 20). Dabei werden Eingangsbelegungen, für die ein Zustandsübergang existiert aus $\bar{\Sigma}$ gestrichen (Zeile 15). Nach Beenden der Schleife enthält $\bar{\Sigma}$ nur noch Eingangsbelegungen, bei denen kein Zustandsübergang erfolgt. Es wird deshalb nur dann ein Zeitnachfolger ermittelt, wenn $\bar{\Sigma}$ nicht leer ist (Zeilen 23-28). Schließlich wird $\bar{z}$ durch Einfügen in $\bar{\mathcal{Z}}_B$ als besucht markiert (Zeile 29).

Im Folgenden wird gezeigt, dass die Aufzählung korrekt und vollständig ist, d.h. dass alle aufgezählten Zeitzustände erreichbar sind und alle erreichbaren Zeitzustände aufgezählt werden.

Algorithmus 1 Vorwärtsanalyse für einen beschränkte SPS-Zeitautomaten $\langle \Sigma^e, \Sigma^a, Z, z_0, U, T \rangle$

1 *// Besuchte Zonen und Folgezonenrelation.*
2 $\overline{\mathcal{Z}}_B := \emptyset$
3 $\rightarrow \; := \emptyset$
4 *// Erreichbare Zonen: Startzone.*
5 $\overline{\mathcal{Z}}_W := \{\langle z_0, \hbar[U := 0]\rangle\}$
6 **while** $\overline{\mathcal{Z}}_W \neq \emptyset$ **do**
7 *// Wähle eine erreichbare Zone.*
8 **choose** $\overline{z} = \langle z, \hbar\rangle \in \overline{\mathcal{Z}}_W$
9 *// Ist die Zone noch unbesucht?*
10 **if** $[\![\overline{z}]\!] \not\subseteq [\![\overline{\mathcal{Z}}_B]\!]$ **then**
11 $\overline{\Sigma} := \mathcal{P}(\Sigma^e)$
12 *// Für alle gültigen Zustandsübergänge.*
13 **forall** $t = \langle z, B^u \wedge B^e, U_R, \Sigma^{a'}, z' \rangle : \hbar \trianglelefteq B^u$ **do**
14 *// Streiche Eingabebelegungen.*
15 $\overline{\Sigma} := \overline{\Sigma} \setminus \{\Sigma^{e'} : \Sigma^{e'} \models B^e\}$
16 *// Bestimme t-Folgezonen.*
17 $\overline{\mathcal{Z}}_t := \mathbf{part}(\overline{z}_t)$
18 $\rightarrow \; := \rightarrow \cup \{\langle \overline{z}, \Sigma^{e'}, U_R, \Sigma^{a'}, \overline{z}' \rangle : \overline{z}' \in \overline{\mathcal{Z}}_t \wedge \Sigma^{e'} \models B^e\}$
19 *// Merke Folgezonen.*
20 $\overline{\mathcal{Z}}_W := \overline{\mathcal{Z}}_W \cup \overline{\mathcal{Z}}_t$
21 **end forall**
22 *// Existieren Eingangsbelegungen ohne Zustandsübergänge?*
23 **if** $\overline{\Sigma} \neq \emptyset$ **then**
24 *// Bestimme und merke Zeitfolgezonen.*
25 $\overline{\mathcal{Z}}\nearrow_\delta^\Delta := \mathbf{part}(\overline{z}\nearrow_\delta^\Delta)$
26 $\rightarrow \; := \rightarrow \cup \{\langle \overline{z}, \Sigma^{e'}, \emptyset, \emptyset, \overline{z}' \rangle : \overline{z}' \in \overline{\mathcal{Z}}\nearrow_\delta^\Delta \wedge \Sigma^{e'} \in \overline{\Sigma}\}$
27 $\overline{\mathcal{Z}}_W := \overline{\mathcal{Z}}_W \cup \overline{\mathcal{Z}}\nearrow_\delta^\Delta$
28 **end if**
29 $\overline{\mathcal{Z}}_B := \overline{\mathcal{Z}}_B \cup \{\overline{z}\}$
30 **end if**
31 **end while**
32 *// Gib Zonengraph zurück.*
33 **return** $\langle \overline{\mathcal{Z}}_B, \Sigma^e, \Sigma^a, \rightarrow \rangle$

Lemma 4.37 (Korrektheit)**.** *Für einen SPS-Zeitautomaten* $\mathcal{A} = \langle \Sigma^e, \Sigma^a, Z, z_0, U, T \rangle$ *gilt während der Vorwärtsanalyse die Invariante:*

$$[\![\overline{\mathcal{Z}}_B]\!] \cup [\![\overline{\mathcal{Z}}_W]\!] \subseteq \mathit{Reach}(\mathcal{A})$$

Beweis. Der Beweis erfolgt induktiv über der Schleife in Zeile 6:

Induktionsanfang Die Zone $\overline{z}_0 = \langle z_0, \hbar[U \coloneqq 0] \rangle$ enthält für einen beliebigen Hyperpolyeder $\hbar \neq \varnothing$ ausschließlich den Startzeitzustand. Da $\overline{\mathcal{Z}}_W = \{\overline{z}_0\}$ und $\overline{\mathcal{Z}}_B = \varnothing$ folgt die Behauptung.

Induktionsbehauptung Nach jedem Durchlauf der Schleife in Zeile 6 gilt: $[\![\overline{\mathcal{Z}}_B]\!] \cup [\![\overline{\mathcal{Z}}_W]\!] \subseteq \mathit{Reach}(\mathcal{A})$

Induktionsbeweis $\overline{\mathcal{Z}}_W$ enthält nur Zonen, die bezüglich aller Zeitbedingungen B^u aller Zustandsübergänge $\langle z, B^u \wedge B^e, U_R, \Sigma^{a'}, z' \rangle \in T$ von $\mathcal{A}$ partitioniert wurden (Zeilen 17, 20, 25 und 27). Somit gilt für alle Zonen $\langle z, \hbar \rangle \in \overline{\mathcal{Z}}_W$ entweder $\hbar \trianglelefteq B^u$ oder $[\![\hbar]\!] \cap [\![B^u]\!] = \varnothing$ (**Invariante 1**).

Aus Invariante 1 und Zeile 13 folgt, dass alle Uhrenstände $\upsilon \in [\![\hbar]\!]$ der Zone $\overline{z}$ die Zeitbedingung B^u von t erfüllen. Mit der Induktionsbehauptung $[\![\overline{z}]\!] \subseteq \mathit{Reach}(\mathcal{A})$ sowie der Definition von $\mathbf{post}^{\Delta}_{\delta}(zz, t)$ und Satz 4.27 folgt $\overline{z}_t \subseteq \mathit{Reach}(\mathcal{A})$. Mit $[\![\overline{\mathcal{Z}}_t]\!] = [\![\overline{z}_t]\!]$ (Satz 4.33) folgt schließlich $[\![\overline{\mathcal{Z}}_t]\!] \subseteq \mathit{Reach}(\mathcal{A})$ (**Invariante 2**).

Aus Zeile 11 und Zeile 15 folgt, dass $\overline{\Sigma}$ in Zeile 23 ausschließlich Eingangsbelegungen enthält, für die für jeden Zeitzustand aus $\overline{z}$ kein Zustandsübergang $t \in T$ erfolgt. Gemäß der Definition von $TR^{\Delta}_{\delta}\nearrow$ besitzt $\overline{z}$ genau für alle Eingangsbelegungen $\Sigma^{e'} \in \overline{\Sigma}$ einen Zeitnachfolger. Mit Satz 4.25 und Satz 4.33 folgt analog zu $\overline{\mathcal{Z}}_t$ die Invariante $[\![\overline{\mathcal{Z}}\nearrow^{\Delta}_{\delta}]\!] \subseteq \mathit{Reach}(\mathcal{A})$ (**Invariante 3**).

Die einzigen Zuweisungen zu den Mengen $\overline{\mathcal{Z}}_B$ und $\overline{\mathcal{Z}}_W$ erfolgen in den Zeilen 20, 27 und 29. Mit Invariante 2, Invariante 3 und $\overline{z} \in \overline{\mathcal{Z}}_W$ (Zeile 8) folgt die Behauptung.

□

Lemma 4.38 (Vollständigkeit)**.** *Für einen SPS-Zeitautomaten* $\mathcal{A} = \langle \Sigma^e, \Sigma^a, Z, z_0, U, T \rangle$ *und die mit der Vorwärtsanalyse ermittelte Menge* $\overline{\mathcal{Z}}_B$ *gilt:*

$$[\![\overline{\mathcal{Z}}_B]\!] \supseteq \mathit{Reach}(\mathcal{A})$$

Beweis. Angenommen, es gäbe einen Zeitzustand $zz'' \in \mathit{Reach}(\mathcal{A})$ für den gilt: $zz'' \notin \overline{\mathcal{Z}}_B$.

Es gilt $zz_0 \in [\![\overline{\mathcal{Z}}_B]\!]$, da die einzige Zuweisung zur Menge $\overline{\mathcal{Z}}_B$ in Zeile 29 vorgenommen und $\overline{z} \in \overline{\mathcal{Z}}_W$ nicht verändert wird.

Demnach müsste im Lauf $r = zz_0 \ldots zz \xrightarrow{\Sigma^{e'}, \tau, \Sigma^{a'}} zz' \ldots zz''$ ein Zustand zz existieren, so dass $zz \in [\![\overline{\mathcal{Z}}_B]\!]$ und $zz' \notin [\![\overline{\mathcal{Z}}_B]\!]$. Da alle Zonen aus $\overline{\mathcal{Z}}_B$ auch in $\overline{\mathcal{Z}}_W$ waren, folgt daraus $zz' \notin [\![\overline{\mathcal{Z}}_W]\!]$.

Laut Satz 4.27 gilt $[\![\overline{z_t}]\!] = \bigcup_{zz \in [\![\overline{z}]\!]} \mathbf{post}^{\Delta}_{\delta}(zz, t)$ und laut Satz 4.25 $[\![\overline{z}\nearrow^{\Delta}_{\delta}]\!] = \bigcup_{zz \in [\![\overline{z}]\!]} \mathbf{post}^{\Delta}_{\delta}(zz)$. Mit Invariante 1 folgt, dass alle Nachfolger von zz in $[\![\overline{z_t}]\!] \cup [\![\overline{z}\nearrow^{\Delta}_{\delta}]\!]$ und damit in $\mathcal{Z}_W$ enthalten sind. Damit kann kein zz' existieren. $\Rightarrow\Leftarrow$ Widerspruch! □

Satz 4.39. *Für einen SPS-Zeitautomaten $\mathcal{A} = \langle \Sigma^e, \Sigma^a, Z, z_0, U, T \rangle$ und den durch die Vorwärtsanalyse aufgezählten Zonengraphen $zg = \langle \overline{\mathcal{Z}}_B, \Sigma^e, \Sigma^a, \rightarrow \rangle$ gilt:*

$$[\![\overline{\mathcal{Z}}_B]\!] = \mathit{Reach}(\mathcal{A})$$

Beweis. Die Behauptung folgt aus Lemma 4.37 sowie $\overline{\mathcal{Z}}_W = \varnothing$ (Zeile 6) und Lemma 4.38. □

Anmerkung 4.40. *Um die Terminierung von Algorithmus 1 zu gewährleisten, kann eine Operation zum Abschätzen von Zonen für Zeitautomaten nach Alur/Dill verwendet werden. Eine ausführliche Diskussion findet man in [Bou03].*

Bei der Abschätzung werden Zeitzustände zusammengefasst, deren Uhrenstände größer als die größte im Zeitautomaten vorkommende Zeitbedingung sind. Die Abschätzung beeinflusst das Ergebnis der Vorwärtsanalyse innerhalb der größten Zeitbedingung *nicht. Das Abschätzen von Zonen* außerhalb der größten Zeitbedingung *hat keinen Einfluss auf das Zurücksetzen von Uhren. Weitere Operationen, die von einem Zeitzustand ausserhalb der größten Zeitbedingung zu einem Zeitzustand innerhalb führen, existieren für Zeitautomaten nach Alur/Dill nicht.*

Diese Argumentation ist auf SPS-Zeitautomaten übertragbar, da

1. *das Zurücksetzen von Uhren dem von Zeitautomaten nach Alur/Dill gleicht und*
2. *Zeitschritte von Zeitzuständen, die außerhalb der größten Zeitbedingung liegen zu Zeitzuständen führt, die ebenfalls außerhalb der größten Zeitbedingung liegen.*

Sofern beim Vergleich zweier SPS-Zeitautomaten der Abschluss bezüglich der größten Zeitbedingung beider Automaten vorgenommen wird, bleibt der Korrektheitsnachweis ebenfalls unberührt. Das ist darauf zurückzuführen, dass Zonen der Ergebnismenge der Vorwärtsanalyse innerhalb der größten Zeitbedingung nicht und außerhalb der größten Zeitbedingung identisch verändert werden.

4.4 Translation Validation

Im folgenden Abschnitt wird gezeigt, wie die Korrektheit einer Implementierung durch den Vergleich zweier SPS-Zeitautomaten sichergestellt werden kann.

4.4.1 Korrektheitsbegriff

Der Begriff der Ähnlichkeit zweier SPS-Zeitautomaten kann unterschiedlich definiert werden. Meist gilt, dass strengere Definitionen von Ähnlichkeit algorithmisch schneller entschieden

werden können. Da die Anzahl bei der Analyse entstehender Zonen sehr hoch sein kann, ist die Effizienz eines Enscheidungsalgorithmus wichtig für die praktische Durchführbarkeit der Translation Validation. Aus diesem Grund wird in diesem Abschnitt ein Korrektheitsbegriff entwickelt, der Freiheitsgrade bezüglich der Implementierung von Nichtdeterminismus bietet und schnell entscheidbar ist.

Modelle

Viele Modelle sind nichtdeterministisch, d.h. sie besitzen im gleichen Zustand und bei gleicher Eingabe mehrere mögliche Folgezustände. Beispiele dafür sind:

Petrinetze: Haben zwei Transitionen eines Petrinetzes Ausgangskanten zur gleichen Stelle, können möglicherweise nicht beide Transitionen gleichzeitig schalten, wenn die Kapazität der Stelle nicht ausreicht. Ein solcher Fall wird als Kontakt bezeichnet und es ist nicht festgelegt, welche der beiden Transitionen schalten darf.

Klassische Zeitautomaten: Treten bei einem klassischen Zeitautomaten zwei Symbole zur selben Zeit auf, ist nicht festgelegt, welches der beiden Symbole zuerst durch den Zeitautomaten verarbeitet wird. Von der Reihenfolge der Symbole hängt gegebenenfalls der Folgezustand ab.

Implementierung

Eine Implementierung eines Modells, beispielsweise ein SPS-Programm, hat meist ein eindeutiges Verhalten. Von der Reihenfolge, in der beispielsweise Symbole oder Transitionen im Programm ausgewertet werden, hängt ab, wie sich die Implementierung in den oben beschriebenen Fällen verhält. Da aus Sicht des Modells in nichtdeterministischen Fällen mehrere Folgezustände korrekt sind, ist die Auswahl eines konkreten Folgezustandes eine *Übersetzerfreiheit*, d.h. der Übersetzer kann beispielsweise unter Optimierungsgesichtspunkten den günstisten Folgezustand auswählen. In diesem Sinne ist die Implementierung eines Modells deterministischer als das Modell.

Simulation

Ein auch für Zeitsysteme entscheidbarer Korrektheitsbegriff ist die *Simulationsrelation* [Par81, Cer93, WL97].

Definition 4.41 ((Bi-)Simulation)**.** *Gegeben seien ein Alphabet* Σ *sowie zwei Transitionssysteme* $\mathcal{T}_M := \langle S_M, S_{M,0}, \Sigma, T_M \rangle$ *und* $\mathcal{T}_I := \langle S_I, S_{I,0}, \Sigma, T_I \rangle$.

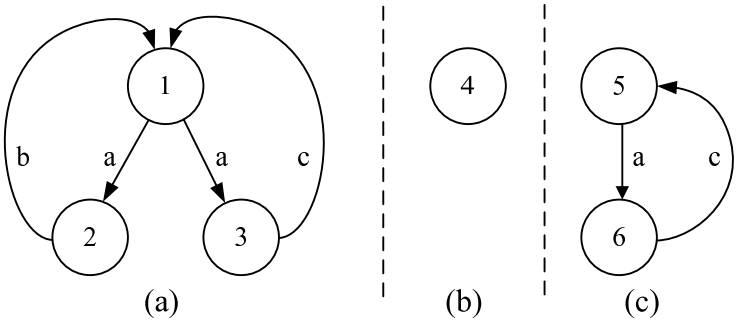


Abbildung 4.7: Drei Transitionssysteme. Sowohl Transitionssystem (b) als auch Transitionssystem (c) simuliert (a).

Eine Relation $\sim \subseteq S_M \times S_I$ *heißt Simulation gdw. für alle Zustände* $s_M \in S_M$ *und* $s_I \in S_I$ *sowie für alle Symbole* $\sigma \in \Sigma$ *gilt:*

$$s_M \sim s_I \wedge s_I \xrightarrow{\sigma} s_I' \Longrightarrow \exists s_M' \in S_M.\ s_M \xrightarrow{\sigma} s_M' \wedge s_M' \sim s_I'$$

Gilt zusätzlich

$$s_M \sim s_I \wedge s_M \xrightarrow{\sigma} s_M' \Longrightarrow \exists s_I' \in S_I.\ s_I \xrightarrow{\sigma} s_I' \wedge s_M' \sim s_I'$$

wird $\sim$ *als Bisimulation bezeichnet.*

$\mathcal{T}_I$ *(bi-)simuliert* $\mathcal{T}_M$ *gdw. eine (Bi-)Simulation* $\sim$ *existiert, die jedem Zustand von* $\mathcal{T}_I$ *mindestens einen Zustand von* $\mathcal{T}_M$ *zuordnet.*

Der Umstand, dass die Implementierung deterministischer als das Modell sein kann, wird bei Korrektheitbegriffen wie der Simulation nur ungenügend beachtet. Folgendes Beispiel verdeutlicht dies:

Beispiel 4.42. *In Abbildung 4.7 sind drei Transitionssysteme dargestellt, deren Kanten beschriftet wurden. Transitionsystem (b) simuliert Transitionssystem (a), da der Zustand 4 nach Definition jeden Zustand von (a) simuliert. Es besteht allerdings keine Bisimulation, da keiner der Zustände von (a) Zustand 4 simuliert.*

Zustand 5 von Transitionssystem (c) simuliert Zustand 1 und Zustand 6 Zustand 3, weshalb (c) Transitionssystem (a) ebenfalls simuliert. Zustand 1 simuliert allerdings Zustand 5 nicht, weshalb auch in diesem Fall keine Bisimulation vorliegt.

Unter der Annahme, dass die Transitionssysteme (b) und (c) das Verhalten zweier Implementierungen und Transitionssystem (a) das Verhalten eines Modells beschreibt, ist das Verhalten von (b) nicht korrekt, da die Implementierung nicht auf das Symbol *a* reagiert. Dem entgegen ist Implementierung (c) korrekt unter der Bedingung, dass eine Implementierung deterministischer als ein Modell sein darf. In diesem Sinne gilt, dass eine *Simulationsrelation als Korrektheitsbegriff zu schwach* ist, da sie (b) akzeptiert, während die *Bisimulation zu stark* ist, da sie (c) ablehnt.

Es wird deshalb ein verschärfter Simulationsbegriff verwendet:

Definition 4.43 (vollst. Simulation)**.** *Für zwei Transitionssysteme* $\mathcal{T}_M := \langle S_M, S_{M,0}, \Sigma, T_M \rangle$ *und* $\mathcal{T}_I := \langle S_I, S_{I,0}, \Sigma, T_I \rangle$ *wird eine Relation* $\sim \in S_M \times S_I$ *als* vollständige Simulation *bezeichnet, wenn gilt:*

- $\sim$ *ist eine Simulationsrelation und*
- *für alle* $s_M \sim s_I$ *gilt* $s_M \xrightarrow{\sigma} s'_M \Rightarrow \exists s'_I \in S_I.\ s_I \xrightarrow{\sigma} s'_I$

Eine vollständige Simulation fordert neben der Simulationsbeziehung zusätzlich, dass für jede Transition, die von einem Zustand s_M in S_M ausgeht, auch eine Transition vom simulierenden Zustand s_I in S_I mit dem gleichen Symbol ausgehen muss. Dadurch wird im Gegensatz zur Simulation sichergestellt, dass zu jeder Transition in S_M mindestens eine Transition in S_I gehört. Ist S_M nichtdeterministisch, d.h. führen von einem Zustand mehrere Transitionen mit gleichem Symbol zu unterschiedlichen Nachfolgezuständen, muss im Gegensatz zur Bisimulation nicht jeder der Nachfolgezustände simuliert werden. Demnach ist die Simulationsrelation (c) $\sim$ (a) in Beispiel 4.42 vollständig, während die Simulationsrelation (b) $\sim$ (a) nicht vollständig ist.

4.4.2 Algorithmische Bestimmung der Simulationsrelation

Die Bestimmung einer Bisimulation ist algorithmisch weniger komplex als die Bestimmung einer Simulation. Es sind Algorithmen bekannt, die eine Bisimulation in $O(m \log n)$ bestimmen, wobei m die Gesamtzahl der Zustände und n die Gesamtzahl der Transitionen der Transitionssysteme sind [Fer89, DPP01].

Die effizientesten bekannten Algorithmen zur Bestimmung einer Simulation haben eine Laufzeitkomplexität von $O(nm)$ [HHK95, BP95]. Die Grundidee dieser Algorithmen ist es, ausgehend von einer Simulationsrelation, die alle Paare von Ecken enthält, schrittweise die Paare zu eliminieren, die die Simulationsrelation verletzen. Dazu werden zuerst die Paare entfernt, die die Simulationsrelation offensichtlich verletzen, da für Ausgangskanten in der Implementierung keine Kante im Modell existiert. Anschließend werden all die Paare entfernt, deren Nachfolger nicht in der Simulationsrelation enthalten sind.

Algorithmus 2 setzt diese Idee sehr einfach zur Prüfung einer vollständigen Simulation um. Im ersten Schritt werden alle Paare von Ecken bestimmt, die die Kriterien der vollständigen Simulation nicht erfüllen. Für jedes dieser Paare $\langle e'_I, e'_M \rangle$ wird anschließend geprüft, welche Vorgänger ebenfalls aus der Simulationsrelation entfallen.

Dazu werden all die Vorgänger e_M von e'_M bestimmt, die keinen alternativen Nachfolger e''_M besitzen, der durch e'_I simuliert wird. Damit ist die Simulationsbedingung für diese Ecken e_M sowie die Vorgänger e_I von e'_I nicht mehr erfüllt, und die Paare $\langle e_I, e_M \rangle$ entfallen aus der Simulation.

Die Laufzeit von Algorithmus 2 kann mit $O(n^2 m)$ abgeschätzt werden.

Algorithmus 2 Entscheidungsprozedur für vollständige Simulation

$\overline{\sim} := \{\langle e_I, e_M\rangle \in E_I \times E_M : \neg(e_M \xrightarrow{\sigma} e'_M \Longrightarrow e_I \xrightarrow{\sigma} e'_I \wedge e_I \xrightarrow{\sigma} e'_I \Longrightarrow e_M \xrightarrow{\sigma} e'_M)\}$
for all $\langle e'_I, e'_M\rangle \in \overline{\sim}$ **do**
  $E'_M := \{e_M : e_M \xrightarrow{\sigma} e'_M \wedge \forall e''_M.\ e_M \xrightarrow{\sigma} e''_M \wedge e'_I \overline{\sim} e''_M\}$
  for all $e_I : e_I \xrightarrow{\sigma} e'_I$ **do**
    for all $e_M \in E'_M$ **do**
      $\overline{\sim} := \overline{\sim} \cup \{\langle e_I, e_M\rangle\}$
    end for
  end for
end for
$\sim := (E_I \times E_M) \setminus \overline{\sim}$

Simulation auf Zuständen

Um eine Simulation auf den *Zeitzuständen* zweier Zeitautomaten nachzuweisen, ist es notwendig, den Produktautomaten der Zeitautomaten zu konstruieren [Cer93, WL97]. Die Anzahl der Zustände des Produktautomaten wächst in der Realität meist derart stark an, dass ein Nachweis der Simulation praktisch unmöglich wird.

Um dennoch einen Nachweis für die Korrektheit einer Implementierung führen zu können, wird ein strengerer Simulationsbegriff vorgeschlagen, der auf den diskreten und nicht auf den Zeitzuständen zweier Zeitautomaten definiert ist:

Definition 4.44 (Zustandssimulation)**.** *Gegeben seien eine Menge von Eingängen Σ^e, eine Menge von Ausgängen Σ^a und eine Menge von Uhren U sowie zwei beschränkte SPS-Zeitautomaten $\mathcal{A}_I = \langle \Sigma^e, \Sigma^a, Z_I, z_I^0, U, T_I, \delta_I, \Delta_I\rangle$ und $\mathcal{A}_M = \langle \Sigma^e, \Sigma^a, Z_M, z_M^0, U, T_M, \delta_M, \Delta_M\rangle$. Die durch $\mathcal{A}_I$ und $\mathcal{A}_M$ definierten Transitionssysteme seien $\mathcal{T}_I := \langle ZZ_I, ZZ_I^0, M, TR_I\rangle$ und $\mathcal{T}_M := \langle ZZ_M, ZZ_M^0, M, TR_M\rangle$. Ferner gelte $\delta_M < \delta_I$ sowie $\Delta_I < \Delta_M$.*

Eine Relation $\sim_Z \subseteq Z_I \times Z_M$ wird als Zustandssimulation bezeichnet, wenn für jeden erreichbaren Zeitzustand $\langle z_I, \upsilon\rangle \in Reach(\mathcal{A}_I)$ und jedes Paar $z_I \sim_Z z_M$ gilt:

1. *Für jede Transition $\langle z_I, \upsilon\rangle \xrightarrow{\Sigma^{e'}, \tau, \Sigma^{a'}} \langle z'_I, \upsilon'\rangle$ existiert eine Transition $\langle z_M, \upsilon\rangle \xrightarrow{\Sigma^{e'}, \tau, \Sigma^{a'}} \langle z'_M, \upsilon'\rangle$ mit $z'_I \sim_Z z'_M$ und*
2. *für jede Transition $\langle z_M, \upsilon\rangle \xrightarrow{\Sigma^{e'}, \tau, \Sigma^{a'}} \langle z'_M, \upsilon'\rangle$ existiert eine Transition $\langle z_I, \upsilon\rangle \xrightarrow{\Sigma^{e'}, \tau, \Sigma^{a'}} \langle z'_I, \upsilon'\rangle$.*

Bestimmung einer Zustandssimulation

Die Bestimmung einer Zustandsrelation erfolgt in zwei Schritten. Im ersten Schritt werden bis auf die Relation der Nachfolger alle Bedingungen der Zeitsimulation geprüft. Dazu wird der Begriff der Zustandsähnlichkeit eingeführt:

Definition 4.45 (Zustandsähnlichkeit)**.** *Es gelten die Bedingungen einer Zustandssimulation.*

Zwei Zustände z_I und z_M werden als ähnlich ($z_I \simeq_Z z_M$) bezeichnet, wenn für jeden erreichbaren Zeitzustand $\langle z_I, \upsilon \rangle \in Reach(\mathcal{A}_I)$ gilt:

1. *Für jede Transition $\langle z_I, \upsilon \rangle \xrightarrow{\Sigma^{e'}, \tau, \Sigma^{a'}} \langle z'_I, \upsilon' \rangle$ existiert eine Transition $\langle z_M, \upsilon \rangle \xrightarrow{\Sigma^{e'}, \tau, \Sigma^{a'}} \langle z'_M, \upsilon' \rangle$ und*
2. *für jede Transition $\langle z_M, \upsilon \rangle \xrightarrow{\Sigma^{e'}, \tau, \Sigma^{a'}} \langle z'_M, \upsilon' \rangle$ existiert eine Transition $\langle z_I, \upsilon \rangle \xrightarrow{\Sigma^{e'}, \tau, \Sigma^{a'}} \langle z'_I, \upsilon' \rangle$.*

Die Zustandsähnlichkeit ist notwendig für eine Zustandssimulation und kann mit Algorithmus 3 bestimmt werden:

Lemma 4.46. *Für zwei Zonengraphen zg_I und zg_M und die Zustände z_I und z_M bestimmt Algorithmus 3, ob $z_I \simeq_Z z_M$ gilt.*

Beweis. Aus Satz 4.39 und Zeile 2 folgt, dass Algorithmus 3 alle erreichbaren Zeitzustände von z_I betrachtet.

Gemäß der Definition der Transitionssysteme $\mathcal{T}_I$ und $\mathcal{T}_M$ besitzt jeder erreichbare Zeitzustand Nachfolger.

Angenommen für einen Uhrenstand υ existiert ein Zeitzustand $\langle z_I, \upsilon \rangle \in Reach(\mathcal{A}_I)$, jedoch kein Zeitzustand $\langle z_M, \upsilon \rangle \in Reach(\mathcal{A}_M)$.

Da gemäß der Definition der Transitionssysteme $\mathcal{T}_I$ und $\mathcal{T}_M$ jeder erreichbare Zeitzustand Nachfolger besitzt, existiert mindestens eine Transition $\langle z_I, \upsilon \rangle \xrightarrow{\Sigma^{e'}, \tau, \Sigma^{a'}} \langle z'_I, \upsilon' \rangle$. Es existiert jedoch keine Transition $\langle z_M, \upsilon \rangle \xrightarrow{\Sigma^{e'}, \tau, \Sigma^{a'}} \langle z'_M, \upsilon' \rangle$, da $\langle z_M, \upsilon \rangle \notin Reach(\mathcal{A}_M)$, womit die Simulationsbedingung verletzt ist (Zeile 4).

In Zeile 6 werden alle Zonen $\overline{z}_M$ bestimmt, die Uhrenstände von $\overline{z}_I$ enthalten. Für alle Zeitzustände $\langle z_I, \upsilon \rangle \in [\![\overline{z}_I]\!]$ und $\langle z_M, \upsilon \rangle \in [\![\overline{z}_M]\!]$ gilt: Existiert eine Transition $\overline{z}_I \xrightarrow{\Sigma^{e'}, U_R, \Sigma^{a'}} \overline{z}'_I$ sowie $\overline{z}_M \xrightarrow{\Sigma^{e'}, U_R, \Sigma^{a'}} \overline{z}'_M$, folgt für alle Zykluszeiten τ aus $\delta_M < \delta_I < \tau < \Delta_I < \Delta_M$ (Voraussetzung für Zustandsähnlichkeit) und aus der Definition von $\mathcal{T}_I$ und $\mathcal{T}_M$, dass für jede Transition $\langle z_I, \upsilon \rangle \xrightarrow{\Sigma^{e'}, \tau, \Sigma^{a'}} \langle z'_I, \upsilon' \rangle$ eine Transition $\langle z_M, \upsilon \rangle \xrightarrow{\Sigma^{e'}, \tau, \Sigma^{a'}} \langle z'_M, \upsilon' \rangle$ existiert.

In den Zeilen 7 und 8 wird für jede Zone geprüft, ob für jede Transition $\overline{z}_I \xrightarrow{\Sigma^{e'}, U_R, \Sigma^{a'}} \overline{z}'_I$ eine entsprechende Transition $\overline{z}_M \xrightarrow{\Sigma^{e'}, U_R, \Sigma^{a'}} \overline{z}'_M$ existiert. Gilt dies nicht, sind die Zustände z_I und z_M nicht ähnlich.

Die zweite Bedingung für die Ähnlichkeit zweier Zustände wird analog in den Zeilen 10 und 11 geprüft. □

Algorithmus 3 Entscheidungsalgorithmus für die Ähnlichkeit $z_I \simeq_Z z_M$

```
1  // Für jede Zone von z_I.
2  forall z̄_I = ⟨z_I, ħ_I⟩ ∈ Z̄_I do
3          // Wird z̄_I vollständig überdeckt?
4          if [[ħ_I]] ⊈ ⋃_{⟨z_M,ħ_M⟩∈Z̄_M} [[ħ_M]] then return false; end if
5          // Prüfe Ähnlichkeitsbedingung.
6          forall z̄_M = ⟨z_M, ħ_M⟩ ∈ Z̄_M : [[ħ_I]] ∩ [[ħ_M]] ≠ ∅ do
7                  forall z̄_I --Σ^e',U_R,Σ^a'--> z̄'_I do
8                          if ∄ z̄_M --Σ^e',U_R,Σ^a'--> z̄'_M then return false; end if
9                  end forall
10                 forall z̄_M --Σ^e',U_R,Σ^a'--> z̄'_M do
11                         if ∄ z̄_I --Σ^e',U_R,Σ^a'--> z̄'_I then return false; end if
12                 end forall
13         end forall
14 end forall
15 return true
```

Die Laufzeit von Algorithmus 3 hängt von der Effizienz ab, mit der für eine gegebene Zone überprüft werden kann, ob diese vollständig überdeckt wird (Zeile 8) bzw. mit der alle überdeckenden Zonen ermittelt werden (Zeile 11).

Unter Verwendung einer geeigneten Datenstruktur können beide Operationen linear in der Anzahl von Zonen ausgeführt werden. Ist z die Gesamtzahl der Zonen der Zustände z_I und z_M und m die Gesamtzahl der Transitionen von z_I und z_M ergibt sich für die Schleifen 2, 4 und 6 eine Laufzeit von $O(z^2)$ sowie für die Schleifen 7 und 10 bei Verwendung geeigneter Datenstrukturen wie Hashtabellen eine Laufzeit von $O(m)$. Die Gesamtlaufzeit kann somit mit $O(z^2 m)$ abgeschätzt werden.

Mit dem Entscheidungsalgorithmus für die Ähnlichkeit von Zuständen kann mit Algorithmus 4 auch eine Zustandssimulation bestimmt werden:

Satz 4.47. *Für zwei Zonengraphen zg_I und zg_M bestimmt Algorithmus 4 die Zustandssimulation.*

Beweis. Der Algorithmus bestimmt $\sim_Z$ indirekt, indem alle Paare von Zuständen $\bar{\sim}_Z$ aufgezählt werden, die nicht in $\sim_Z$ enthalten sind.

Gemäß Zeile 1 sind alle Zustandspaare, die nicht in $\bar{\sim}_Z$ enthalten sind, zustandsähnlich. Da im Rest des Algorithmus keine Paare aus $\bar{\sim}_Z$ entfernt werden, enthält $\sim_Z$ nur zustandsähnliche Paare.

Algorithmus 4 Entscheidungsprozedur für Zustandssimulation

```
1   ~̄_Z := {⟨z_I, z_M⟩ ∈ Z_I × Z_M : ¬(z_I ≃_Z z_M)}
2   forall ⟨z'_I, z'_M⟩ ∈ ~̄_Z do
3           Z'_M := {z_M : z_M -σ-> z'_M ∧ ∀z''_M. z_M -σ-> z''_M ∧ z'_I ~̄_Z z''_M}
4           forall z_I : z_I -σ-> z'_I do
5                   forall z_M ∈ Z'_M do
6                           ~̄_Z := ~̄_Z ∪ {⟨z_I, z_M⟩}
7                   end forall
8           end forall
9   end forall
10  ~_Z := (Z_I × Z_M) \ ~̄_Z
```

Laut ihrer Definition setzt eine Zustandssimulation Zustandsähnlichkeit voraus, wobei ein Zustand z_I einen Zustand z_M nur dann simuliert, wenn die Nachfolger von z_M durch die Nachfolger von z_I simuliert werden. Daraus folgt, dass nur dann $z_I \simeq_Z z_M$ aber nicht $z_I \sim_Z z_M$ gelten kann, wenn Nachfolgezustände z'_I und z'_M von z_I und z_M in $\bar{\sim}_Z$ enthalten sind.

Für alle Zustände $z'_I \bar{\sim}_Z z'_M$ (Zeile 2) werden in Zeile 3 die Vorgängerzustände von z'_M ermittelt, die keinen Nachfolger z''_M besitzen, der durch z'_I simuliert wird. Damit besteht laut Definition 4.44 keine Zustandssimulation zwischen Zuständen z_M in Z'_M und den Vorgängern z_I von z'_I. Alle entsprechenden Paare werden in Zeile 6 in $\bar{\sim}_Z$ aufgenommen. □

Die Laufzeit von Zeile 1 ergibt sich aus der Laufzeit der Bestimmung von $\simeq_Z$ für alle Paare von Zuständen sowie den nachfolgenden Schleifen. Sei n die Anzahl der Zustände, m die Gesamtzahl der Transitionen und z die Anzahl der Zonen. Dann benötigt Zeile 1 $O(z^2m)$, da höchstens jede Zone jedes Zustands z_I mit jeder Zone jedes Zustands z_M verglichen wird. Die Komplexität der Schleife 2 hängt von der Anzahl der diskreten Zustände ab und kann mit $O(n^2m)$ abgeschätzt werden. Nimmt man an, dass jeder Zustand des SPS-Zeitautomaten erreichbar ist, gilt $n < z$ und es ergibt sich eine Gesamtkomplexität von $O(z^2m)$, womit das Entscheidungsverfahren polynomiell in der Größe der Zonengraphen ist.

4.5 Zusammenfassung

In diesem Kapitel wurde gezeigt, wie eine Translation Validation auf Zeitautomaten durchgeführt werden kann. Dabei werden im Wesentlichen drei Schritte durchlaufen:

1. Im ersten Schritt wurden mit SPS-Zeitautomaten Modelle definiert, die Steuermodell und -programm abbilden. SPS-Zeitautomaten erfüllen die Anforderungen aus Kapitel 3 und erlauben die *unmittelbare Darstellung* der diskreten Zustände von TNCES-Modellen und ST-Programmen. Durch diesen Schritt hat die komplexe Semantik der

Steuermodelle und -programme keinen Einfluss auf die weitere Translation Validation. Zudem ist es möglich, andere Steuermodell- und -programmarten zu ergänzen.

2. Im zweiten Schritt wird das Verhalten der SPS-Zeitautomaten analysiert. Dazu werden die Zeitzuständer der SPS-Zeitautomaten symbolisch durch Zonen repräsentiert. Zonen basieren auf Hyperpolyedern und erlauben die direkte Abbildung aller zur Analyse von SPS-Zeitautomaten benötigten Operationen. Eine *Umwandlung der Modelle* zur Abbildung der speziellen Eigenschaften von SPS oder andere *Zwischenschritte sind nicht notwendig*.

   Zonengraphen beschreiben das Verhalten der SPS-Zeitautomaten für Modell und Implementierung. Eine angepasste Version der Vorwärtsanalyse erlaubt deren direkten Aufbau. Satz 4.39 stellt die Korrektheit der Zonengraphen sicher.

   Der Aufbau und die Speicherung der Zonengraphen ist der zeit- und speicherintensivste Teil der Translation Validation. Durch die direkte Abbildung der Eigenschaften von SPS wird erreicht, dass dieser Teil nicht oder nur unwesentlich aufwändiger als die Modellprüfung ist.

3. Im dritten Schritt wird die eigentliche Translation Validation durchgeführt. Dabei wird ein *angepasster Korrektheitsbegriff* verwendet. Dieser beachtet den *möglichen Nichtdeterminismus* des Steuermodells und stellt sicher, dass das Steuerprogramm *mindestens ein gültiges Verhalten* nachbildet. Um die Effizienz der Translation Validation zu erhöhen wird der *Korrektheitsnachweis auf den Zuständen der SPS-Zeitautomaten* geführt. Um den Nachweis praktisch zu erbringen, wird ein einfacher und effizienter Algorithmus vorgeschlagen. Die Sätze 4.47 und 4.15 stellen schließlich sicher, dass das Steuerprogramm auch unter realen Bedingungen kein ungültiges Verhalten besitzt.

Offen bleibt die praktische Umsetzung der für die Translation Validation notwendigen Operationen und Datenstrukturen. Im folgenden Kapitel wird gezeigt, wie diese mit Hilfe von Hyperpolyedern einheitlich implementiert werden können.

5 Symbolische Darstellung von Zeitzuständen

SPS-Zeitautomaten besitzen im Gegensatz zu klassischen endlichen Automaten eine unendliche Menge von Zeitzuständen, da SPS-Zeitautomaten neben einer endlichen Anzahl diskreter Zustände eine unendliche Menge möglicher Uhrenstände besitzen. Es ist daher nicht möglich, alle Zeitzustände eines SPS-Zeitautomaten direkt aufzuzählen.

Um einen SPS-Zeitautomaten zu analysieren, werden dessen Zeitzustände symbolisch repräsentiert. Beginnend mit der symbolischen Repräsentation des Startzustands wird eine symbolische Repräsentiation aller Nachfolgezeitzustände bestimmt. Ausgehend von dieser Menge werden erneut die Nachfolger bestimmt, bis alle erreichbaren Zeitzustände symbolisch repräsentiert wurden. Als Datenstruktur für die symbolische Repräsentation von Zeitzuständen werden in Modellprüfern vorwiegend Differenzgrenz-Matrizen eingesetzt, da diese die einfache und effiziente Umsetzung aller notwendiger Operationen erlauben [BY04].

Um sicherzustellen, dass Zeitzustände nicht immerfort aufgezählt werden und die Analyse terminiert, werden bereits besuchte Zeitzustände ebenfalls symbolisch indiziert. Die Anforderungen an die dazu verwendete Indexstruktur unterscheiden sich wesentlich von den Anforderungen an Differenzgrenz-Matrizen, da das Einfügen und Finden von Zeitzuständen die wichtigsten Operationen auf diesen Indexstrukturen darstellen [Ben01].

Die in anderen Arbeiten vorgestellten Indexstrukturen wie Entscheidungsdiagramme, minimale Beschränkungssysteme oder gepackte Differenzgrenz-Matrizen besitzen zwar eine sehr gute Speichereffizienz, machen aber meist eine aufwändige Konvertierung zwischen Differenzgrenz-Matrizen und Indexstruktur notwendig [Ben01, LLPY97, Wan04]. Zudem sind insbesondere Entscheidungsdiagramme nicht dazu geeignet, zu jedem indizierten Zeitzustand dessen Nachfolger zu speichern, wie es für den in dieser Arbeit vorgestellten Vergleich von Zeitsystemen notwendig ist.

Ziel dieses Kapitels ist die Entwicklung einer Datenstruktur, die zur Speicherung von Zonengraphen geeignet ist. Dazu wird im ersten Abschnitt dieses Kapitels gezeigt, wie die zur Analyse von SPS-Zeitautomaten benötigten Operationen mit Hyperpolyedern umgesetzt werden können. Im zweiten Abschnitt wird eine Datenstruktur zur Speicherung von Zonengraphen vorgestellt, die ebenfalls auf Hyperpolyedern basiert.

5.1 Implementierung von Hyperpolyedern

Sowohl eine einzelne Beschränkung als auch eine Differenzgrenz-Matrix stellt einen Spezialfall eines Hyperpolyeders dar. Operationen, die auf Differenzgrenz-Matrizen anwendbar sind, können auf Hyperpolyeder übertragen werden. Die Operationen sind auf Grund der geringeren Anzahl von Beschränkungen mit Hyperpolyedern häufig effizienter ausführbar und entsprechen im schlechtesten Fall den Operationen auf Differenzgrenz-Matrizen. Zudem erlauben Hyperpolyeder die speichersparende Darstellung weniger Beschränkungen.

Das ermöglicht den Einsatz von Hyperpolyedern sowohl bei der Aufzählung der Zeitzustände eines SPS-Zeitautomaten als auch ihre Anwendung in Indexstrukturen wie Entscheidungsdiagrammen oder den im Abschnitt 5.2 vorgestellten Suchbäumen.

5.1.1 Spezielle Beschränkungen von Hyperpolyedern

Für die Implementierung der Operationen auf Hyperpolyedern werden Beschränkungen mit bestimmten Eigenschaften benötigt. Je nach ihren Eigenschaften werden solche Beschränkungen allgemeine, Grenz- oder Minimalbeschränkungen genannt.

Grenzbeschränkungen

Genau wie bei Differenzgrenz-Matrizen kann dieselbe Menge von Uhrenständen durch unterschiedliche Hyperpolyeder dargestellt werden. So könnte der Hyperpolyeder $\hbar$ aus Beispiel 3.19 durch eine beliebige Beschränkung $\beta^{2-1} = \mathsf{u}_2 - \mathsf{u}_1 \diamond \mathsf{c}$, wobei $\mathsf{c} \geq 3, \diamond \in \{<, \leq\}$, ergänzt werden, ohne dadurch die Menge der durch $\hbar$ dargestellten Uhrenstände zu verändern. Für manche Operationen auf Hyperpolyedern ist die schärfste Beschränkungen über zwei Uhren, also $\mathsf{u}_2 - \mathsf{u}_1 < 3$, wichtig und wird als Grenzbeschränkung bezeichnet.

Grenzbeschränkungen sind allgemein wie folgt definiert:

Definition 5.1 (Grenzbeschränkung)**.** *Sei $U = \{u_1, \ldots, u_n\}$ eine Menge von Uhren und $\hbar$ ein beliebiger, nicht leerer Hyperpolyeder über U. Ferner seien $i, j \in \{0, 1, \ldots, n\}$ zwei Uhrenindizes, mit $i \neq j$.*

Eine Beschränkung heißt Grenzbeschränkung von $\hbar$ über den Uhren u_i und u_j, geschrieben $\beta^{i-j}_{gbs}(\hbar) = \mathsf{u}_\mathsf{i} - \mathsf{u}_\mathsf{j} \diamond \mathsf{c}$, gdw. (i) $[\![\hbar]\!] \subseteq [\![\beta^{i-j}_{gbs}(\hbar)]\!]$ und (ii) keine Beschränkung β^{i-j} existiert, so dass $[\![\hbar]\!] \subseteq [\![\beta^{i-j}]\!]$ und $\beta^{i-j} \lhd \beta^{i-j}_{gbs}(\hbar)$.

Beispiel 5.2. *In Abbildung 3.5 sind alle Beschränkungen des Hyperpolyeder $\hbar$ Grenzbeschränkungen. Zusätzlich ist die Beschränkung $\beta^{2-1}_{gbs}(\hbar) = \mathsf{u}_2 - \mathsf{u}_1 < 3$ eine Grenzbeschränkung von $\hbar$.*

Anmerkung 5.3. *Grenzbeschränkungen entsprechen den Beschränkungen einer normalisierten Differenzgrenz-Matrix, da eine Differenzgrenz-Matrix gerade dann normalisiert ist, wenn alle Beschränkungen der Differenzgrenz-Matrix die schärfsten Beschränkungen sind [CGP99, Dil89].*

Zur Bestimmung einer Grenzbeschränkungen eines Hyperpolyeders kann ein Hyperpolyeder wie eine Differenzgrenz-Matrix als ein gewichteter Graph repräsentiert werden. Eine Grenzbeschränkung kann mit Hilfe des Bellman-Ford-Algorithmus in $O(nm)$, wobei n die Anzahl der Uhren und m die Anzahl der Beschränkungen ist, berechnet werden. Die Bestimmung kürzester Pfade nach Dijkstra ist nicht anwendbar, da der entstehende Graph negative Kantengewichte enthalten kann. Der Algorithmus von Bellman und Ford lohnt sich inbesondere dann, wenn die Anzahl der Beschränkungen eines Hyperpolyeders gering ist und nur eine Grenzbeschränkung berechnet werden muss.

In den meisten Fällen kann die Menge aller Grenzbeschränkungen eines Hyperpolyeders wie bei Differenzgrenz-Matrizen einfacher mit dem Floyd-Warshall-Algorithmus in $O(n^3)$ berechnet werden. Beide Algorithmen können auch zur Detektion negativer Zyklen im Graphen verwendet werden, was anzeigt, dass der entsprechende Hyperpolyeder leer ist.

Satz 5.4 (Grenzbeschränkung)**.** *Sei U eine Menge von Uhren, $\hbar$ ein nicht leerer Hyperpolyeder über U und $u_i, u_j \in U, i \neq j$ zwei beliebige aber verschiedene Uhren. B sei die Menge aller Beschränkungen $\beta^{i,\ldots,j}$, die von in $\hbar$ enhaltenen Beschränkungen abgeleitet wurden.*

Dann ergibt sich die Grenzbeschränkung $\beta^{i-j}_{gbs}(\hbar)$ der Uhren u_i und u_j als

$$\beta^{i-j}_{gbs}(\hbar) = \min_{\lhd} B$$

Beweis. Die Bestimmung einer Grenzbeschränkung kann aus der Normalisierungsoperation für DBMs [CGP99, Dil89] abgeleitet werden. □

Korollar 5.5. *Jede Beschränkung β^{i-j} eines Hyperpolyeders $\hbar$, die keine Grenzbeschränkung ist, kann aus $\hbar$ entfernt werden, ohne $[\![\hbar]\!]$ zu verändern.*

Beweis. Jede schärfere Beschränkung $\beta^{i-j}_s \lhd \beta^{i-j}$ würde andernfalls die Menge $[\![\hbar]\!]$ verändern. Damit wäre β^{i-j} eine Grenzbeschränkung. □

Minimalbeschränkungen

Ein Hyperpolyeder über n Uhren kann bis zu $2n$ orthogonale, $n(n-1)$ diagonale und insgesamt max. $n(n+1)$ Beschränkungen besitzen. Nicht alle diese Beschränkungen sind zur Darstellung der Uhrenstände des Hyperpolyeders notwendig. Eine Beschränkung ist notwendig, wenn das Entfernen der Beschränkung aus dem Hyperpolyeder eine Veränderung der Menge der dargestellten Uhrenstände zur Folge hat. In [LLPY97] wird gezeigt, wie zu einem gegebenen Hyperpolyeder die minimale Anzahl notwendiger Beschränkungen ermittelt werden kann. In der

folgenden Definition werden derartige Beschränkungen unter dem Begriff Minimalbeschränkung eingeführt.

Definition 5.6 (Minimaler Hyperpolyeder, Minimale Beschränkung)**.** *Sei $\hbar = \beta_1 \wedge \ldots \wedge \beta_n$ ein beliebiger Hyperpolyeder. Der Hyperpolyeder $\hbar$ wird als* minimaler Hyperpolyeder *bezeichen, gdw. kein Hyperpolyeder $\hbar' = \beta_1' \wedge \ldots \wedge \beta_m'$ existiert, so dass $[\![\hbar]\!] = [\![\hbar']\!]$ und $m < n$.*

Zu einem beliebigen Hyperpolyeder $\hbar$ bezeichnet **min**($\hbar$) *einen minimalen Hyperpolyeder. Jede Beschränkung von* **min**($\hbar$) *wird als* minimale Beschränkung *bezeichnet.*

Beispiel 5.7. *Der in Abbildung 3.5 dargestellte Hyperpolyeder ist minimal und eindeutig.*

Korollar 5.8. *Alle Minimalbeschränkungen eines Hyperpolyeders sind Grenzbeschränkungen.*

Beweis. Laut Definition 5.6 gilt, dass aus einem minimalen Hyperpolyeder $\hbar$ keine Beschränkung entfernt werden kann ohne $[\![\hbar]\!]$ zu verändern. Ferner gilt laut Korollar 5.5, dass alle Beschränkungen eines Hyperpolyeders, die keine Grenzbeschränkungen sind, aus dem Hyperpolyeder entfernt werden können. □

Anmerkung 5.9. *In [LLPY97] wird ein Algorithmus vorgestellt, der in $O(n^3)$, wobei n die Anzahl der Uhren ist, zu einer gegebenen Differenzgrenz-Matrix den minimalen Hyperpolyeder bestimmt und der leicht für allgemeine Hyperpolyeder adaptiert werden kann. In [LLPY97] wird ferner gezeigt, dass der minimale Hyperpolyeder* **min**($\hbar$) *eindeutig ist, sofern $\hbar$ nicht leer ist und für keine zwei seiner Uhren u_i und u_j mit $u_i \neq u_j$ gilt, dass $\beta_{gbs}^{i-j}(\hbar) = u_i - u_j \leq c$ und $\beta_{gbs}^{j-i}(\hbar) = u_j - u_i \leq -c$ entspricht. Der minimale Hyperpolyeder bildet unter dieser Bedingung eine Normalform, d.h. jeder minimale Hyperpolyeder beschränkt genau eine Menge von Uhrenständen und zu jeder durch einen Hyperpolyeder beschränkten Menge von Uhrenständen existiert genau ein minimaler Hyperpolyeder.*

Grenzfälle ergeben sich für Hyperpolyeder, die leer sind, und Hyperpolyeder, die die oben genannte Bedingung nicht erfüllen. Hyperpolyeder die leer sind, können beispielsweise durch je zwei beliebige Beschränkungen $u_i - u_j < c$ und $u_j - u_i < -c$ minimal dargestellt werden. Für Hyperpolyeder, die die erstgenannte Bedingung nicht erfüllen, existieren mehrere minimale Hyperpolyeder. Beispielsweise kann ein Hyperpolyeder $u_1 - u_0 \leq 1 \wedge u_0 - u_1 \leq -1 \wedge u_2 - u_0 \leq 2 \wedge u_0 - u_2 \leq -2$ minimal durch $u_0 - u_1 \leq -1 \wedge u_2 - u_0 \leq 2 \wedge u_1 - u_2 \leq -1$ oder durch $u_1 - u_0 \leq 1 \wedge u_0 - u_2 \leq -2 \wedge u_2 - u_1 \leq 1$ dargestellt werden.

Der in [LLPY97] vorgestellte Algorithmus erlaubt es, auch für Differenzgrenz-Matrizen, die derartige Grenzfälle darstellen, einen minimale Hyperpolyeder in $O(n^3)$ zu bestimmen. Der Algorithmus baut auf der Bestimmung der Grenzbeschränkungen auf, so dass es möglich ist, Grenz- und Minimalbeschränkungen in einem Durchgang zu berechnen.

Der wesentliche Vorteil einer Darstellung von Hyperpolyedern als minimale Hyperpolyeder ist der geringe Platzverbrauch und die reduzierte Komplexität einiger Operationen. Minimale Hyperpolyeder bieten sich deshalb vor allem für Indexstrukturen an.

5.1.2 Darstellung von Hyperpolyedern

Um alle $n(n+1)$ möglichen Beschränkungen für n Uhren zu speichern, bietet sich die Verwendung einer Matrix an. Soll nur eine Beschränkung dargestellt werden, wie es oft bei Indexstrukturen notwendig ist, kann diese Beschränkung effizient als Tupel dargestellt werden.

Für die Darstellung von Hyperpolyeder existieren verschiedene Möglichkeiten, z.B. das Speichern von Beschränkungen in Matrizen, Listen oder Indexstrukturen, mit spezifischen Vor- und Nachteilen.

Die wesentlichen Operationen, die auf einer Datenstruktur zur Speicherung von Beschränkungen ausgeführt werden, sind das Aufzählen aller Beschränkungen sowie das Suchen und das Einfügen einer Beschränkung. Da Hyperpolyeder nicht unbedingt normalisiert sein müssen, ist es zudem sinnvoll, zu merken, welche Beschränkungen Grenz- oder Minimalbeschränkungen sind, um die Berechnung von Grenz- und Minimalbeschränkungen erst auszuführen, wenn dies notwendig ist (lazy evaluation).

Einige Datenstrukturen, die für die Implementierung von Hyperpolyedern in Frage kommen, sowie deren spezifische Vor- und Nachteile, werden im Folgenden kurz vorgestellt:

Vollständige Matrizen

Ein Hyperpolyeder kann durch ein zweidimensionales Feld repräsentiert werden. Jedes Element (i, j) des Feldes ist entweder *null* oder ein Tupel der Form $\langle c, \diamond, typ \rangle$. Ist ein Element *null*, enthält der Hyperpolyeder keine Beschränkung über den Uhren u_i und u_j, andernfalls enthält er die Beschränkung $\beta^{i-j} = \mathsf{u_i} - \mathsf{u_j} \diamond \mathsf{c}$. $typ \in \{gbs, min, null\}$ zeigt an, ob es sich bei β^{i-j} um eine Grenzbeschränkung oder eine Minimalbeschränkung handelt, oder ob dies nicht bekannt ist. Diese Darstellung entspricht bis auf die Markierung von Grenz- oder Minimalbeschränkungen einer Differenzgrenz-Matrix.

Das Suchen, Einfügen und Ersetzen einzelner Beschränkungen kann bei dieser Darstellung sehr schnell in $O(1)$ erfolgen. Das Aufzählen der Beschränkungen eines Hyperpolyeders dauert allerdings unabhängig von der tatsächlichen Anzahl an Beschränkungen stets $O(n^2)$. Auch der Speicherbedarf ist mit $O(n^2)$ für beliebige Hyperpolyeder konstant hoch. Damit bietet sich diese Datenstruktur insbesondere für Hyperpolyeder an, die sehr viele Beschränkungen enthalten.

Dünnbesetzte Matrizen

Bei dünnbesetzten Matrizen werden die Reihen oder Spalten der Matrix einzeln abgespeichert. Dabei wird nur dann Speicher für eine Spalte oder Reihe reserviert, wenn mindestens eine Beschränkung in dieser Spalte oder Reihe nicht *null* ist. Ein Beispiel für eine reihenweise abgelegte, dünnbesetzte Matrix ist in Abbildung 5.1 dargestellt.

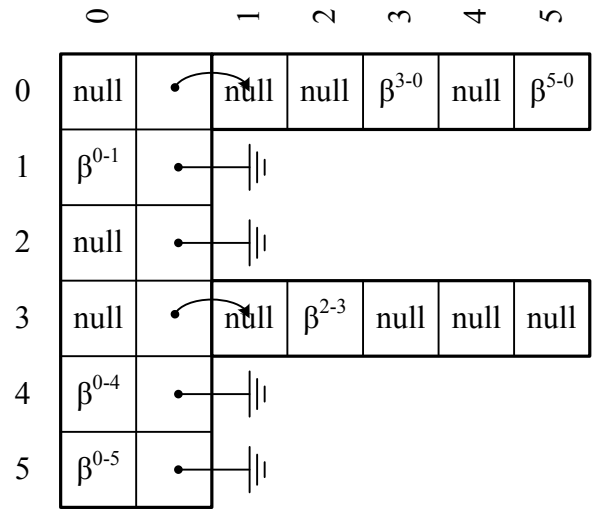


Abbildung 5.1: Repräsentation eines Hyperpolyeders $\beta^{0-1} \wedge \beta^{0-4} \wedge \beta^{0-5} \wedge \beta^{3-0} \wedge \beta^{5-0} \wedge \beta^{2-3}$ durch eine dünnbesetzte Matrix.

Der Vorteil dieser Datenstruktur liegt in ihrem geringeren Speicherverbrauch als vollständige Matrizen, wenn ein Hyperpolyeder nicht für alle Uhrenpaare Beschränkungen enthält. Insbesondere Hyperpolyeder, die nur orthogonale Beschränkungen enthalten, können so kompakt dargestellt werden. Die asymptotische Laufzeit aller Operationen entspricht der vollständiger Matrizen, wobei ein zusätzlicher Overhead für die Verwaltung der Reihen oder Spalten entsteht.

Sortierte Listen

Werden Beschränkungen $u_i - u_j \diamond c$ als Tupel $\langle i, j, c, \diamond, typ \rangle$ dargestellt, können mehrere Beschränkungen als sortierte Liste dargestellt werden. Wird die Liste als Feld dargestellt und erfolgt die Sortierung über den Indizes der Uhren, kann eine Beschränkung mit Hilfe der Binärsuche schnell gefunden werden.

Sortierte Listen benötigen wenig Speicherplatz bei wenigen Beschränkungen. Da für jede Beschränkung im Gegensatz zu Matrizen die Indizes der Uhren gespeichert werden müssen, gilt die Ersparnis von Speicherplatz nur für Hyperpolyeder mit wenigen Beschränkungen. Insbesondere das Aufzählen von m Beschränkungen kann im Gegensatz zu Matrizen in $O(m)$ erfolgen. Die Suche einer bestimmten Beschränkung kann mit Hilfe von Binärsuche in $O(\log m)$ erfolgen, das Einfügen einer Beschränkung dauert $O(m)$. Aufgrund der schlechten Leistung beim Einfügen von Beschränkungen, der guten Speicherplatzeffizienz und der einfachen Aufzählung der Beschränkungen empfehlen sich sortierte Listen insbesondere bei Indexstrukturen, deren Hyperpolyeder selten geändert werden oder bei Hyperpolyedern mit einer geringen Anzahl von Beschränkungen.

5.1.3 Operationen auf Hyperpolyedern

In diesem Abschnitt wird gezeigt, wie die Operationen „Schärfer“, „Schnitt“ und „Abschluss“ aus Abschnitt 3.2.3 sowie die Operationen „Zurücksetzen“ und „Zeitfortschritt“ aus Abschnitt 4.2 mit Hyperpolyedern praktisch umgesetzt werden können. Dabei ist es nicht notwendig, Hyperpolyeder in Differenzgrenz-Matrizen umzuwandeln oder eine Normalform zu bilden.

Schärfere Hyperpolyeder

Ob ein Hyperpolyeder $\hbar_1$ schärfer als ein Hyperpolyeder $\hbar_2$ ist, kann folgendermaßen ermittelt werden:

Satz 5.10. *Seien $\hbar_1$ und $\hbar_2$ zwei Hyperpolyeder über einer Menge U von Uhren. Es gilt $\hbar_1 \trianglelefteq \hbar_2$ gdw. für alle Beschränkungen $\beta_2^{i-j} \in \hbar_2$ gilt: $\beta_{gbs}^{i-j}(\hbar_1) \trianglelefteq \beta_2^{i-j}$.*

Es gilt weiter $\hbar_1 \triangleleft \hbar_2$ gdw. $\hbar_1 \trianglelefteq \hbar_2$ und nicht $\hbar_2 \trianglelefteq \hbar_1$ gilt.

Beweis. Für jeden Uhrenstand $\upsilon \in [\![\hbar_1]\!]$ gilt laut Definition 5.1 für jede Grenzbeschränkung $\beta_{gbs}(\hbar_1)$ von $\hbar_1$: $\upsilon \in [\![\beta_{gbs}(\hbar_1)]\!]$. Aus Definition 3.12 folgt insbesondere $\upsilon \in [\![\beta_{gbs}^{i-j}(\hbar_1)]\!] \subseteq [\![\beta_2^{i-j}]\!]$ für alle $\beta_2^{i-j} \in \hbar_2$. Damit gilt $\hbar_1 \trianglelefteq \hbar_2$. Würde auch $[\![\hbar_2]\!] \subseteq [\![\hbar_1]\!]$ gelten, dann wäre $[\![\hbar_2]\!] = [\![\hbar_1]\!]$ und somit $\hbar_1 \not\triangleleft \hbar_2$.

□

Anmerkung 5.11. *Um praktisch festzustellen, ob für zwei Hyperpolyeder $\hbar_1$ und $\hbar_2$ die Aussage $\hbar_1 \trianglelefteq \hbar_2$ gilt, kann ausgenutzt werden, dass aus $\beta_1^{i-j} \trianglelefteq \beta_2^{i-j}$ mit Definition 5.1 sofort $\beta_{gbs}^{i-j}(\hbar_1) \trianglelefteq \beta_2^{i-j}$ folgt. Damit ist die Ermittlung der Grenzbeschränkungen von $\hbar_1$ nur dann notwendig, wenn eine der Beschränkungen von $\hbar_2$ schärfer als die entsprechende Beschränkung von $\hbar_1$ ist. Gilt das nicht oder sind die Grenzbeschränkungen von $\hbar_1$ bekannt, benötigt die Operation die Zeit $O(n) \subseteq O(|U^2|)$, wobei n die Anzahl der Beschränkungen von $\hbar_2$ ist, andernfalls beträgt die Laufzeit aufgrund der Bestimmung der Grenzbeschränkungen $O(|U|^3)$.*

Schnitt

Die Darstellung des Schnitts zweier Hyperpolyeder kann folgendermaßen bestimmt werden:

Satz 5.12 (Schnitt)**.** *Seien $\hbar := \beta_1 \wedge \ldots \wedge \beta_n$ und $\hbar' := \beta'_1 \wedge \ldots \wedge \beta'_m$ zwei Hyperpolyeder. Sei $B := \{\beta_1, \ldots, \beta_n, \beta'_1, \ldots, \beta'_m\}$ die Menge der Beschränkungen von $\hbar$ und $\hbar'$. Weiter enthalte die Menge $B_\triangleleft := \{\beta \in B : \nexists \beta' \in B.\ \beta' \triangleleft \beta\}$ nur diejenigen Beschränkungen, die selbst nicht beschränkt werden. Dann kann der Schnitt der Hyperpolyeder $\hbar$ und $\hbar'$ als $\hbar \cap \hbar' = \bigwedge_{\beta \in B_\triangleleft} \beta$ bestimmt werden.*

Beweis. Für zwei beliebige Hyperpolyeder $\hbar$ und $\hbar'$ und für alle Uhrenstände υ ist zu zeigen: $\upsilon \in [\![\hbar \cap \hbar']\!] \iff \upsilon \in [\![\hbar]\!] \cap [\![\hbar']\!]$.
Nach Definition 3.18 gilt: $\upsilon \in [\![\hbar]\!] \iff \forall \beta \in \hbar.\ \upsilon \in [\![\beta]\!]$. Somit folgt $\upsilon \in [\![\hbar]\!] \cap [\![\hbar']\!] \iff \upsilon \in [\![\hbar]\!] \wedge \upsilon \in [\![\hbar']\!] \overset{3.18}{\iff} \forall \beta_i \in \{\beta_1, \ldots, \beta_n\}.\ \upsilon \in [\![\beta_i]\!] \wedge \forall \beta'_i \in \{\beta'_1, \ldots, \beta'_m\}.\ \upsilon \in [\![\beta'_i]\!] \overset{3.18}{\iff} \upsilon \in [\![\beta_1 \wedge \ldots \wedge \beta_n \wedge \beta'_1 \wedge \ldots \wedge \beta'_m]\!]$.

Für einen beliebigen Hyperpolyeder $\hbar$ und zwei Beschränkungen $\beta \in \hbar$ und $\beta' \in \hbar$, wobei $\beta \lhd \beta'$, gilt $[\![\hbar]\!] = [\![\bigwedge_{\beta \in \hbar \wedge \beta \neq \beta'} \beta]\!]$, da $[\![\hbar]\!] \overset{3.18}{\subseteq} [\![\beta]\!] \overset{3.12}{\subset} [\![\beta']\!]$. Aus $\beta_1 \wedge \ldots \wedge \beta_n \wedge \beta'_1 \wedge \ldots \wedge \beta'_m$ können demnach alle Beschränkungen gestrichen werden, die ihrerseits beschränkt werden. □

Anmerkung 5.13. *Es existieren verschiedene Möglichkeiten, den Schnitt zweier Hyperpolyeder zu berechnen. Die in Satz 5.12 vorgestellte Variante ist in der Laufzeit bei geeigneten Datenstrukturen durch $O(m+n) \subseteq O(|U|^2)$ beschränkt und der entstehende Hyperpolyeder enthält max.* $\mathbf{min}\{m+n, |U|^2\}$ *Beschränkungen. Der Schnitt wird aber unter Umständen nicht mit der minimalen Anzahl von Beschränkungen dargestellt.*

Abschluss

Auch der Abschluss kann durch einen Hyperpolyeder dargestellt werden.

Satz 5.14 (Abschluss)**.** *Seien $\hbar_1$ und $\hbar_2$ zwei beliebige, nichtleere Hyperpolyeder über den Uhren U. Sei*

$$B^{max} := \left\{\beta_{gbs}(\hbar_1) : \beta_{gbs}(\hbar_2) \trianglelefteq \beta_{gbs}(\hbar_1)\right\} \cup \left\{\beta_{gbs}(\hbar_2) : \beta_{gbs}(\hbar_1) \trianglelefteq \beta_{gbs}(\hbar_2)\right\} \tag{5.1}$$

die Menge derjenigen Grenzbeschränkungen von $\hbar_1$ bzw. $\hbar_2$, die selbst nicht beschränkt werden.

Dann gilt

$$\hbar_1 \oplus \hbar_2 = \bigwedge_{\beta \in B^{max}} \beta$$

Beweis.
Eigenschaft $[\![\hbar_1]\!] \cup [\![\hbar_2]\!] \subseteq [\![\hbar_1 \oplus \hbar_2]\!]$:
Es ist zu beweisen, dass $[\![\hbar_1]\!] \subseteq [\![\hbar_1 \oplus \hbar_2]\!]$ und $[\![\hbar_2]\!] \subseteq [\![\hbar_1 \oplus \hbar_2]\!]$. Es wird gezeigt, dass $[\![\hbar_1]\!] \subseteq [\![\hbar_1 \oplus \hbar_2]\!]$, $[\![\hbar_2]\!] \subseteq [\![\hbar_1 \oplus \hbar_2]\!]$ folgt analog.

Für jeden Uhrenstand υ gilt laut Definition 5.1: $\upsilon \in [\![\hbar_1]\!]$ gdw. für alle $\beta_{gbs}(\hbar_1)$ die Aussage $\upsilon \in [\![\beta_{gbs}(\hbar_1)]\!]$ gilt. Ferner gilt laut der Definition von B^{max} für alle $\beta_{gbs}(\hbar_1)$: Es gibt ein $\beta \in B^{max}$ und $\beta_{gbs}(\hbar_1) \trianglelefteq \beta$. Es folgt $\upsilon \in [\![\beta_{gbs}(\hbar_1)]\!] \subseteq [\![\beta]\!]$ und damit $\upsilon \in [\![\hbar_1 \oplus \hbar_2]\!]$.

Eigenschaft $\nexists \hbar_3.\ [\![\hbar_3]\!] \subset [\![\hbar_1 \oplus \hbar_2]\!] \wedge [\![\hbar_1]\!] \cup [\![\hbar_2]\!] \subseteq [\![\hbar_3]\!]$:
Beweis durch Widerspruch. Angenommen, es existiert ein Hyperpolyeder $\hbar_3$, so dass $[\![\hbar_3]\!] \subset$

$[\![\hbar_1 \oplus \hbar_2]\!]$ und $[\![\hbar_1]\!] \cup [\![\hbar_2]\!] \subseteq [\![\hbar_3]\!]$ gilt. Aus Satz 5.10 und $[\![\hbar_3]\!] \subset [\![\hbar_1 \oplus \hbar_2]\!]$ folgt, dass eine Grenzbeschränkung $\beta_{gbs}(\hbar_3)$ und eine Beschränkung $\beta \in \hbar_1 \oplus \hbar_2$ existiert, so dass $(i)\, \beta_{gbs}(\hbar_3) \triangleleft \beta$. Gemäß Gleichung 5.1 ist β eine Grenzbeschränkung von $\hbar_1$ oder $\hbar_2$. Angenommen, β ist Grenzbeschränkung von $\hbar_1$, dann folgt aus (i), dass ein Uhrenstand $\upsilon \in \hbar_1$ existiert, so dass $\upsilon \notin \hbar_3$. $\Rightarrow\Leftarrow$ Widerspruch zu $[\![\hbar_1]\!] \cup [\![\hbar_2]\!] \subseteq [\![\hbar_3]\!]$. Analog führt die Annahme, β sei Grenzbeschränkung von $\hbar_2$, zum Widerspruch. □

Zurücksetzen

Das Zurücksetzen von Uhren kann von Differenzgrenz-Matrizen abgeleitet und einfach mit Hyperpolyedern dargestellt werden.

Satz 5.15 (Zurücksetzen)**.** *Sei $\hbar$ ein Hyperpolyeder über den Uhren $U = \{u_1, \ldots, u_n\}$ und $u_R \in U$ eine beliebige Uhr in U.*

Die Menge

$$B := \{\beta^{i-j} : \beta^{i-j} \in \hbar \wedge i \neq R \wedge j \neq R\}$$

enthalte alle Beschränkungen von $\hbar$, die ihrerseits u_R nicht enthalten. Außerdem seien folgende Beschränkungen definiert

$$\begin{aligned}
\beta^{0-R} &:= \mathsf{u}_0 - \mathsf{u}_R \leq 0 \\
\beta^{R-0} &:= \mathsf{u}_R - \mathsf{u}_0 \leq 0 \\
\beta^{R-i} &:= \mathsf{u}_R - \mathsf{u}_i \diamond \mathsf{c} && \textit{wobei } \beta^{0-i}_{gbs}(\hbar) = \mathsf{u}_0 - \mathsf{u}_i \diamond \mathsf{c} \textit{ und } i \neq 0 \\
\beta^{i-R} &:= \mathsf{u}_i - \mathsf{u}_R \diamond \mathsf{c} && \textit{wobei } \beta^{i-0}_{gbs}(\hbar) = \mathsf{u}_i - \mathsf{u}_0 \diamond \mathsf{c} \textit{ und } i \neq 0
\end{aligned}$$

Dann kann das Zurücksetzen der Uhr u_R in $\hbar$ als

$$\hbar[u_R := 0] := \bigwedge_{i \in \{0,\ldots,n\}} \beta^{R-i} \wedge \bigwedge_{i \in \{0,\ldots,n\}} \beta^{i-R} \wedge \bigwedge_{\beta \in B} \beta$$

dargestellt werden.

Beweis. Die Aussage von Satz 5.15 kann direkt aus der Reset-Operation für Differenzgrenz-Matrizen hergeleitet werden [Dil89, CGP99]. □

Zeitschritt

Schließlich wird noch der Zeitschritt mit Hilfe von Hyperpolyedern dargestellt.

Satz 5.16 (Zeitschritt)**.** *Sei $U = \{u_1, \ldots, u_n\}$ eine Menge von Uhren und $\hbar$ ein nicht leerer Hyperpolyeder, dessen Beschränkungen Uhren aus U enthalten. Sei*

$$B_U := \{u_0 - u_i < c - \delta : u_0 - u_i \diamond c \in \hbar\}, u_i \in U, \diamond \in \{<, \leq\}$$

die Menge der um δ erweiterten unteren orthogonalen Beschränkungen von $\hbar$ und

$$B_O := \{u_i - u_0 < c + \Delta : u_i - u_0 \diamond c \in \hbar\}, u_i \in U, \diamond \in \{<, \leq\}$$

die Menge der um Δ erweiterten oberen orthogonalen Beschränkungen von $\hbar$. Ferner sei

$$B_D := \left\{\beta_{gbs}^{i-j}(\hbar) : i \neq j \wedge i, j \in \{1, \ldots, n\}\right\}$$

die Menge aller diagonalen Grenzbeschränkungen von $\hbar$.

Dann kann der Zeitschritt von $\hbar$ als

$$\hbar\nearrow_\delta^\Delta := \bigwedge_{\beta \in B_U} \beta \wedge \bigwedge_{\beta \in B_O} \beta \wedge \bigwedge_{\beta \in B_D} \beta$$

dargestellt werden.

Beweis. Für einen beliebigen Uhrenstand υ ist zu beweisen: $\upsilon \in [\![\hbar\nearrow_\delta^\Delta]\!] \iff \exists \tau \in \mathbb{Q}.\ \delta < \tau < \Delta \wedge \upsilon - \tau \in [\![\hbar]\!]$. Diese Aussage ist äquivalent zu

$$\upsilon \notin [\![\hbar\nearrow_\delta^\Delta]\!] \iff \nexists \tau \in \mathbb{Q}.\ \delta < \tau < \Delta \wedge \upsilon - \tau \in [\![\hbar]\!]$$

was im Folgenden bewiesen wird.

Richtung $\upsilon \notin [\![\hbar\nearrow_\delta^\Delta]\!] \Longrightarrow \nexists \tau \in \mathbb{Q}.\ \delta < \tau < \Delta \wedge \upsilon - \tau \in [\![\hbar]\!]$:
Nach der Definition von $\hbar\nearrow_\delta^\Delta$ gilt für einen Uhrenstand υ die Aussage $\upsilon \notin [\![\hbar\nearrow_\delta^\Delta]\!]$, gdw. gilt:

1. $\exists \beta_O^{i-0} \in B_O.\ \upsilon \notin [\![\beta_O^{i-0}]\!]$ oder
2. $\exists \beta_U^{0-i} \in B_U.\ \upsilon \notin [\![\beta_U^{0-i}]\!]$ oder
3. $\exists \beta_D^{i-j} \in B_D.\ \upsilon \notin [\![\beta_D^{i-j}]\!]$

1. Fall:
Für jede obere Beschränkung $\beta_O^{i-0} = u_i - u_0 < c + \Delta \in B_O$ von $\hbar\nearrow_\delta^\Delta$ gilt:

$$\begin{aligned} \upsilon \notin [\![\beta_O^{i-0}]\!] &\overset{\text{Def. 3.7}}{\iff} \upsilon(u_i) \geq c + \Delta && \mid -\tau \qquad (5.2)\\ &= \upsilon(u_i) - \tau \geq c + \Delta - \tau && \mid \tau < \Delta \\ &= \upsilon(u_i) - \tau > c && \qquad (5.3)\end{aligned}$$

Die entsprechende Beschränkung $\beta_\hbar^{i-0} \in \hbar$ hat entweder die Form:

1.1 $\beta_\hbar^{i-0} = u_i - u_0 < c$ oder

1.2 $\beta_{\hbar}^{i-0} = \mathsf{u_i} - \mathsf{u_0} \leq \mathsf{c}$.

Fall 1.1:
Es gilt $(\upsilon - \tau) \notin [\![\beta_{\hbar}^{i-0}]\!] \overset{\text{Def. 3.7}}{\iff} \upsilon(u_i) - \tau \geq \mathsf{c}$ und mit Gleichung (5.3) folgt

$$\upsilon \notin [\![\beta_{O}^{i-0}]\!] \implies (\upsilon - \tau) \notin [\![\beta_{\hbar}^{i-0}]\!]$$

Mit $(\upsilon - \tau) \notin [\![\beta_{\hbar}^{i-0}]\!] \implies (\upsilon - \tau) \notin [\![\hbar]\!]$ folgt schließlich die Behauptung.

Fall 1.2:
Aus $(\upsilon - \tau) \notin [\![\beta_{\hbar}^{i-0}]\!] \overset{\text{Def. 3.7}}{\iff} \upsilon(u_i) - \tau > \mathsf{c}$ und Gleichung (5.3) folgt

$$\upsilon \notin [\![\beta_{O}^{i-0}]\!] \iff (\upsilon - \tau) \notin [\![\beta_{\hbar}^{i-0}]\!] \tag{5.4}$$

Mit $(\upsilon - \tau) \notin [\![\beta_{\hbar}^{i-0}]\!] \implies (\upsilon - \tau) \notin [\![\hbar]\!]$ folgt ebenfalls die Behauptung.

Zusätzlich folgt für die Rückrichtung mit $\upsilon \notin [\![\beta_{O}^{i-0}]\!] \implies \upsilon \notin [\![\hbar \nearrow_{\delta}^{\Delta}]\!]$ aus Gleichung 5.4:

$$(\upsilon - \tau) \notin [\![\beta_{\hbar}^{i-0}]\!] \implies \upsilon \notin [\![\hbar \nearrow_{\delta}^{\Delta}]\!] \tag{5.5}$$

für $\upsilon(u_i) - \tau > \mathsf{c}$. Für $\upsilon(u_i) - \tau = \mathsf{c}$ erhält man durch beidseitiges Addieren von Δ und mit $\tau < \Delta$: $\upsilon(u_i) > \mathsf{c} + \Delta$. Mit Gleichung (5.2) folgt ebenfalls $\upsilon \notin [\![\beta_{O}^{i-0}]\!]$ und somit $\upsilon \notin [\![\hbar \nearrow_{\delta}^{\Delta}]\!]$.

2. Fall:
Der zweite Fall kann analog zum 1. Fall mit $\delta < \tau$ bewiesen werden.

3. Fall:
Laut der Definition von B_D gilt für alle $\beta_{D}^{i-j} = \mathsf{u_i} - \mathsf{u_j} \diamond \mathsf{c} \in B_D$, dass β_{D}^{i-j} Grenzbeschränkung von $\hbar$ ist. Damit folgt $[\![\hbar]\!] \subseteq [\![\beta_{D}^{i-j}]\!]$ und daraus: (*i*) $\forall \upsilon \notin [\![\beta_{D}^{i-j}]\!].\ \upsilon \notin [\![\hbar]\!]$ sowie (*ii*) $\forall \upsilon \notin [\![\beta_{D}^{i-j}]\!].\ \upsilon \notin [\![\hbar \nearrow_{\delta}^{\Delta}]\!]$.

Ferner gilt, dass jeder Vorgänger eines beliebigen Uhrenstandes υ von einer diagonalen Beschränkung β^{i-j}, $i \neq 0, j \neq 0$ beschränkt wird, wenn υ durch β^{i-j} beschränkt wird: (*iii*) $\forall \tau \in \mathbb{Q}.\ \upsilon - \tau \in [\![\beta^{i-j}]\!] \iff \upsilon \in [\![\beta^{i-j}]\!]$, da $(\upsilon(u_1) - \tau) - (\upsilon(u_2) - \tau) = \upsilon(u_1) - \upsilon(u_2)$.

Somit folgt für beliebige $\tau \in \mathbb{Q}$: $\upsilon \notin [\![\beta_{D}^{i-j}]\!] \overset{(iii)}{\iff} \upsilon - \tau \notin [\![\beta_{D}^{i-j}]\!] \overset{(i)}{\implies} \upsilon - \tau \notin [\![\hbar]\!]$ und damit die Behauptung.

Zusätzlich folgt

$$\upsilon - \tau \notin [\![\beta_{D}^{i-j}]\!] \overset{(iii)}{\iff} \upsilon \notin [\![\beta_{D}^{i-j}]\!] \overset{(ii)}{\implies} \upsilon \notin [\![\hbar \nearrow_{\delta}^{\Delta}]\!] \tag{5.6}$$

Richtung $\nexists \tau \in \mathbb{Q}.\ \delta < \tau < \Delta \wedge \upsilon - \tau \in [\![\hbar]\!] \implies \upsilon \notin [\![\hbar \nearrow_{\delta}^{\Delta}]\!]$:
Analog der ersten Richtung können ebenfalls drei Fälle für $\upsilon - \tau \notin [\![\hbar]\!]$ unterschieden werden. Gilt $\upsilon - \tau \notin [\![\beta_{\hbar}^{i-0}]\!]$ für obere Beschränkungen von $\hbar$ folgt die Behauptung aus Schlussfolgerung (5.5). Analog kann für untere Beschränkungen verfahren werden. Für diagonale Beschränkungen folgt die Behauptung aus Schlussfolgerung (5.6). □

5.2 Suchbäume

Zur Darstellung der Menge $\bar{\mathcal{Z}}_B$ und des Zonengraphen in der Vorwärtsanalyse werden spezielle Datenstrukturen benötigt, die das schnelle Einfügen und Auffinden von Zonen erleichtern. Die einfachste und von Modellprüfern häufig eingesetzte Datenstruktur zur Indizierung von Zeitzuständen ist eine Liste von Differenzgrenz-Matrizen. Da jede Differenzgrenz-Matrix $n(n+1)$ Beschränkungen enthält und die Bestimmung der Überdeckung einer Differenzgrenz-Matrix durch andere Differenzgrenz-Matrizen komplex ist, besitzen Listen von Differenzgrenz-Matrizen eine schlechte Speicher- und Laufzeiteffizienz.

Da die Effizienz der Speicherung besuchter Zustände die Laufzeit- und Speichereffizienz der Analyse maßgeblich bestimmt, wurden verschiedene Datenstrukturen zur effizienten Indizierung von Zeitzuständen vorgeschlagen. Besonders vielversprechend sind Datenstrukturen auf der Basis von Entscheidungsdiagrammen [BLP+99, MLAH99, Str99, Wan01].

Entscheidungsdiagramme können die Verifikation von Zeitsystemen beschleunigen, sind aber schlecht geeignet, um Zonengraphen zu speichern. Für diesen Zweck bieten sich Indexstrukturen wie Suchbäume an, die zu jeder Zone deren Nachfolger speichern können. Ferner basieren viele Datenstrukturen nicht auf den bei der Aufzählung verwendeten DBMs und erfordern deshalb neben einem Mehraufwand bei der Entwicklung oft auch eine aufwändige Umwandlung der DBMs.

In diesem Abschnitt wird ein Suchbaum zur Speicherung von Uhrenständen auf der Basis von Hyperpolyedern vorgeschlagen. Dazu werden im ersten Abschnitt Suchbäume definiert und gezeigt, wie Informationen aus Suchbäumen gelesen werden können. Der Aufbau von Suchbäumen ist Thema des anschließenden Abschnittes.

5.2.1 Definition

Für die Definition von Suchbäumen wird das Komplement eines Hyperpolyeders benötigt.

Definition 5.17. *Das Komplement $\complement\hbar$ eines Hyperpolyeders $\hbar = \beta_1 \wedge \ldots \wedge \beta_n$ ist definiert als:*

$$\complement\hbar := \complement\beta_1 \wedge \ldots \wedge \complement\beta_n$$

Das Komplement eines Hyperpolyeders $\hbar$ entspricht nicht der Invertierung von $\hbar$, d.h. $[\![\complement\hbar]\!] \neq \mathbb{Q}^U_{\geq 0} \setminus [\![\hbar]\!]$. Die Invertierung ist nicht allgemein durch einen einzigen Hyperpolyeder darstellbar. Aus Definition 5.17 ergibt sich folgendes Korollar.

Korollar 5.18. *Für alle $\hbar$ gilt: $[\![\hbar]\!] \cup [\![\complement\hbar]\!] = \mathbb{Q}^U_{\geq 0}$ gdw. $\hbar$ höchstens eine Beschränkung besitzt.*

Definition 5.19 (Suchbaum, Hülle)**.** *Ein Suchbaum $\mathcal{S}$ ist ein Tupel $\langle E, K, U, M, m, bound, split\rangle$ wobei U eine Menge von Uhren und $\mathcal{H}$ die Menge aller Hyperpolyeder über U ist und*

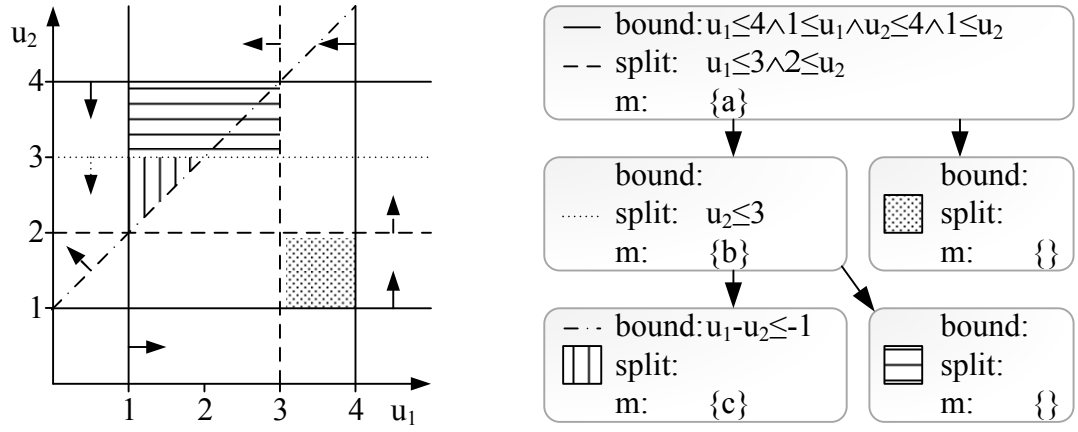


Abbildung 5.2: Beispiel für einen Suchbaum.

- $\langle E, K\rangle$ *ein geordneter Binärbaum ist,*
- M *eine beliebige Menge von Markierungen ist,*
- $m : E \to \mathcal{P}(M)$ *jeder Ecke eine Menge von Markierungen und*
- $bound : E \to \mathcal{H}$ *sowie*
- $split : E \to \mathcal{H}$ *jeder Ecke einen Hyperpolyeder zuordnet.*

Die Wurzel des Binärbaumes wird mit root bezeichnet. Die Menge aller Blätter ist $leaves \subseteq E$. *Die Kinder einer inneren Ecke* $e \in E \setminus leaves$ *werden in ein linkes Kind* $l(e)$ *bzw. rechtes Kind* $r(e)$ *unterschieden.*

Die Hülle $hull(e)$ *einer Ecke ist wie folgt definiert:*

- $hull(root) = bound(root)$
- $hull(l(e)) = hull(e) \cap bound(l(e)) \cap split(e)$
- $hull(r(e)) = hull(e) \cap bound(r(e)) \cap \complement split(e)$

Für jede Ecke $e \in E$ *muss gelten:* $hull(e) \neq \emptyset$.

Beispiel 5.20. *In Abbildung 5.2 ist rechts ein einfacher Suchbaum dargestellt. Der Suchbaum enthält drei Blätter und zwei innere Ecken. Jede Ecke des Suchbaumes entspricht einem Rechteck in der rechten Abbildung, das linke und rechte Kind ist bei inneren Ecken jeweils durch einen Pfeil markiert. Die Attribute bound, split und m sind für jede Ecke angegeben.*

Links sind die im Suchbaum vorkommenden Beschränkungen in einem Uhrenstandsdiagramm eingezeichnet. Entgegen der üblichen Konvention markieren unterschiedliche Linienmuster die verschiedenen bound- und split-Attribute der Ecken. Die entsprechenden Linienmuster sind auf der rechten Seite an den dazugehörigen Attributen zu finden.

Die Hülle von Ecken wird während der Traversierung eines Suchbaumes bestimmt und muss nicht explizit gespeichert werden. Die Hülle des Blattes $l(l(root))$ ist in Abbildung 5.2 links senkrecht schraffiert dargestellt und kann folgendermaßen abgeleitet werden:

$$\begin{aligned}
hull(root) &= u_1 - u_0 \leq 4 \wedge u_0 - u_1 \leq -1 \wedge u_2 - u_0 \leq 4 \wedge u_0 - u_2 \leq -1 \\
hull(l(root)) &= u_1 - u_0 \leq 3 \wedge u_0 - u_1 \leq -1 \wedge u_2 - u_0 \leq 4 \wedge u_0 - u_2 \leq -2 \\
hull(l(l(root))) &= u_1 - u_0 \leq 3 \wedge u_0 - u_1 \leq -1 \wedge u_2 - u_0 \leq 3 \wedge u_0 - u_2 \leq -2 \wedge \\
&\quad u_1 - u_2 \leq -1 \\
&= u_0 - u_1 \leq -1 \wedge u_2 - u_0 \leq 3 \wedge u_1 - u_2 \leq -1
\end{aligned}$$

Die Hülle des Blattes $r(l(root))$ ist waagerecht schraffiert und die Hülle des Blattes $r(root)$ gepunktet markiert. Die Schraffuren sind jeweils als Quadrate in den Blättern des Suchbaumes eingezeichnet.

In der Analyse von Zeitsystemen verknüpft ein Suchbaum die erreichbaren Uhrenstände eines Zustandes mit deren Nachfolgern. Entsprechend ordnet ein Suchbaum jedem Blatt einen Hyperpolyeder, die Hülle des Blattes, sowie eine Menge von Markierungen zu. Zur Indizierung der Blätter werden die Hüllen der Kinder einer inneren Ecke e durch $split(e)$ getrennt und sind überschneidungsfrei. Es folgt, dass die Hüllen aller Kinder ebenfalls überschneidungsfrei sind.

Statt indizierte Hyperpolyeder direkt in den Blättern des Suchbaumes zu speichern, sind Beschränkungen, die für alle Kinder einer Ecke gelten, nur einmalig in dieser Ecke enthalten. Das ermöglicht eine effektivere Traversierung des Suchbaumes, da die Menge möglicher Uhrenstände so früh wie möglich durch $bound(e)$ eingeschränkt wird, zudem werden Beschränkungen, die von mehreren Kindern verwendet werden, nur einmalig betrachtet und gespeichert.

Entsprechend wird mit Markierungen verfahren. Markierungen, die alle Blätter unterhalb einer inneren Ecke enthalten, werden in dieser Ecke gespeichert. Dieses Vorgehen reduziert den Speicherbedarf, wenn Markierungen mehrfach vorkommen.

Dass die Hüllen der Blätter unter einer Ecke von der Hülle dieser Ecke überdeckt werden, stellt folgendes Korollar sicher:

Korollar 5.21. *Aus der Definition der Hülle folgt sofort, dass $hull(l(e)) \trianglelefteq hull(e)$ und ferner $hull(r(e)) \trianglelefteq hull(e)$ und damit allgemein die Hülle einer Ecke die Hüllen aller Nachfahren einschließt.*

Unter Ausnutzung dieser Eigenschaft kann im Folgenden ein Algorithmus entwickelt werden, der für einen gegebenen Hyperpolyeder prüft, ob dieser vollständig überdeckt wird.

5.2.2 Überdeckung

Für die Anwendung von Suchbäumen zur Analyse von SPS-Zeitautomaten sind vor allem zwei Operationen wichtig: Festzustellen, ob ein Hyperpolyeder vollständig von den Hyper-

Algorithmus 5 Überdeckung $incl(e, \hbar) \rightarrow \{\textbf{true}, \textbf{false}\}$

```
11 if ħ ⊴ bound(e) then
12         // invariant: ħ ⊴ hull(e)
13         if e ∈ leaves ∨ [[ħ]] = ∅ then
14                 return true
15         end if
16         if [[ħ ∩ split(e)]] ∪ [[ħ ∩ ⋐split(e)]] = [[ħ]] then
17                 return incl(l(e), ħ ∩ split(e)) ∧ incl(r(e), ħ ∩ ⋐split(e))
18         end if
19 end if
20 return false
```

polyedern eines Suchbaumes überdeckt wird und die Markierungen zu ermitteln, die einem Hyperpolyeder zugeordnet wurden. Beide Operationen sind vergleichsweise einfach implementierbar.

Definition 5.22 (Überdeckung)**.** *Ein Suchbaum $\mathcal{S}$ überdeckt einen Hyperpolyeder $\hbar$ gdw. gilt:*

$$[\![\hbar]\!] \subseteq \bigcup_{e \in leaves} [\![hull(e)]\!]$$

Algorithmus 5 stellt eine Funktion $incl(e, \hbar)$ dar, welche feststellt, ob ein Hyperpolyeder $\hbar$ von einer Ecke e eines Suchbaums überdeckt wird. In Zeile 16 wird geprüft, ob der Hyperpolyeder $\hbar$ durch $split(e)$ und $\complement split(e)$ vollständig überdeckt wird.

Lemma 5.23. *Sei U eine Menge von Uhren, $\mathcal{S}$ ein Suchbaum über U und $\hbar$ ein Hyperpolyeder über U. Algorithmus 5 berechnet dann, ob Überdeckung $\hbar$ von $\mathcal{S}$ überdeckt wird.*

Beweis. Da $hull(root) = bound(root)$ (Definition 5.19) und $\hbar \trianglelefteq bound(root)$ (Zeile 11) gilt die Invariante in Zeile 12 für die Wurzel. Für alle weiteren Ecken folgt die Invariante aus Zeile 11 und Zeile 17 sowie der Definition der Hülle. Gilt die Invariante nicht, folgt aus Korollar 5.21, dass mindestens ein Uhrenstand existieren muss, der von keinem Blatt überdeckt wird.

Handelt es sich bei der betrachteten Ecke e um ein Blatt, folgt aus der Invariante die Überdeckung von $\hbar$. Da der leere Hyperpolyeder in jedem anderen Hyperpolyeder enthalten ist, existiert eine Überdeckung für alle Ecken, wenn $[\![\hbar]\!] = \varnothing$ (Zeile 13).

Existiert ein Uhrenstand $\upsilon \in \hbar$, so dass $\upsilon \notin split(e) \cup \complement split(e)$, enthält weder die Hülle des linken, noch die Hülle des rechten Kindes und somit auch kein Blatt diesen Uhrenstand (Zeile 16). In Zeile 17 gilt andernfalls, dass $\hbar \trianglelefteq hull(e)$ (Invariante), e eine innere Ecke ist (Zeile 13) und $[\![\hbar]\!] \trianglelefteq [\![split(e)]\!] \cup [\![\complement split(e)]\!]$ (Zeile 16). Demnach muss jeder Uhrenstand von $\hbar$ entweder vom linken oder rechten Kind überdeckt werden. Dies wird in Zeile 17 schließlich durch rekursiven Aufruf geprüft. □

Algorithmus 6 Überschneidung $get(e, \hbar) \rightarrow M$

```
1  if ħ ∩ bound(e) ≠ ∅ then
2          if e ∈ leaves then
3                  return m(e)
4          end if
5          return m(e) ∪ get(l(e), ħ ∩ split(e)) ∪ get(r(e), ħ ∩ ⊂split(e))
6  end if
7  return ∅
```

Anmerkung 5.24. *Das algorithmische Prinzip zur Ermittlung der Überdeckung eines Hyperpolyeders kann auch für andere Operationen auf Suchbäumen verwendet werden. In Algorithmus 6 ist ohne Beweis als Beispiel dargestellt, wie zu einem Hyperpolyeder $\hbar$ alle Markierungen eines Suchbaumes ermittelt werden können, die von $\hbar$ überschnitten werden.*

Unter der Annahme, dass $split(e)$ nur eine Beschränkung enthält, ist die Bestimmung, ob $\hbar$ schärfer als $bound(e)$ ist, mit $O(|U^3|)$ die komplexeste Operation in $incl(e, \hbar)$. Da im schlechtesten Fall alle Ecken eines Suchbaumes $\mathcal{S}$ besucht werden, kann die Laufzeit von $incl(root, \hbar)$ für $\mathcal{S}$ mit $O(|E||U|^3)$ abgeschätzt werden.

In der Praxis werden selten alle Blätter eines Suchbaumes von einem Hyperpolyeder überdeckt. Für die Aufzählung der Zustände eines Zeitsystems ist die Feststellung, ob ein Hyperpolyeder nicht überdeckt wird und besucht werden muss, relevanter. In diesem Fall kann die Suche bei einem Suchbaum oft sehr früh abgebrochen werden während beispielsweise eine Liste von Differenzgrenz-Matrizen komplett durchsucht werden muss.

Im folgenden Abschnitt wird gezeigt, wie ein Suchbaum aufgebaut wird.

5.2.3 Einfügen

Die Darstellung einer Menge von Zonen mit Hilfe von Suchbäumen ist nicht eindeutig, d.h. es können mehrere Suchbäume existieren, die ein und dieselbe Menge von Zonen repräsentieren. Das konkrete Aussehen des Suchbaums hängt von der zum Einfügen verwendeten Funktion ab. Diese kann zum Teil Freiheitsgrade wie verschiedene mögliche Splitdimensionen beim Aufbau des Suchbaums nutzen, um dessen Aussehen zu beeinflussen.

Im Folgenden wird eine einfache Variante zum Einfügen von Paaren der Form $\langle \hbar, N \rangle$ in einen Suchbaum vorgestellt und bewiesen. Das Einfügen erfolgt rekursiv, beginnend mit der Wurzel. Dabei wird unterschieden, ob es sich bei der Ecke, in die $\langle \hbar, N \rangle$ eingefügt werden soll, um ein Blatt oder eine innere Ecke handelt. Dieser Umstand ist in Algorithmus 7 zu sehen.

Für die Implementierung der eigentlichen Einfügeoperationen wird eine Operation benötigt,

Algorithmus 7 *insert*$(e, \hbar, N)$

```
1  if e ∈ leaves then
2          insertLeaf(e, ħ, N)
3  else
4          insertInner(e, ħ, N)
5  end if
```

die aus einem Hyperpolyeder alle Beschränkungen entfernt, die in einem zweiter Hyperpolyeder enthalten sind. Diese Operation wird als Differenz zweier Hyperpolyeder bezeichnet.

Definition 5.25 (Differenz)**.** *Seien $\hbar_1$ und $\hbar_2$ zwei Hyperpolyeder über einer Menge U von Uhren. Der Hyperpolyeder*

$$\hbar_1 \setminus \hbar_2 := \bigwedge_{\beta^{i-j} \in \hbar_1 : \beta^{i-j} \notin \hbar_2} \beta^{i-j}$$

wird als Differenz von $\hbar_1$ und $\hbar_2$ bezeichnet.

Algorithmus 8 fügt einen Hyperpolyeder $\hbar$ mit der Markierung N in einen Suchbaum ein, dessen Wurzel ein Blatt ist. Es müssen das Blatt e, dessen Hülle $hull(e)$ und Markierung $m(e)$ bekannt sein.

Beim Einfügen eines Hyperpolyeders in ein Blatt e werden drei Fälle unterschieden:

1. Der erste Fall ist in Abbildung 5.3 dargestellt: Die Hülle von e und der einzufügende Hyperpolyeder überschneiden sich nicht. In diesem Fall wird die Menge $\mathcal{H}_1$ der Beschränkungen von $\hbar$ ermittelt, die als Splitbeschränkungen in Frage kommen. Anschließend werden zwei Blätter erzeugt: $l(e)$ enthält alle Zeitzustände von $\hbar$, $r(e)$ enthält die Zeitzustände von e. Eine neue Wurzel enthält als Splitbeschränkungen die Beschränkungen aus $\mathcal{H}_1$. Abschließend werden noch alle Beschränkungen, die sowohl das linke als auch das rechte Blatt beschränken in $bound(root)$ übernommen und aus den Kindecken entfernt. Der entstandene Suchbaum ist in Abbildung 5.4 abgebildet.
2. Der zweite Fall ist in Abbildung 5.5 dargestellt: Die Hülle von e und $\hbar$ überschneiden sich, sind aber nicht gleich. In diesem Fall enthält die Menge $\mathcal{H}_{2_a}$ alle Beschränkungen, die als Splitbeschränkung in Frage kommen. Es werden wie im ersten Fall zwei Blätter konstruiert, wobei eine beliebige Beschränkung aus $\mathcal{H}_{2_a}$ als Splitbeschränkung verwendet wird. Die Wahl der Splitbeschränkung hat dabei Einfluss auf die Anzahl neuer Blätter, die erzeugt werden müssen.

   Das rechte Blatt enthält denjenigen Teil der Hülle von e, der $\hbar$ nicht schneidet, das linke Blatt entsprechend den Teil der Hülle, der $\hbar$ schneidet. Da das linke Blatt weiterhin $\hbar$ schneidet, wird Algorithmus 8 für das linke Blatt rekursiv aufgerufen. Danach wird

Algorithmus 8 *insertLeaf*$(e, \hbar, N)$

```
1  H1  := {β ∈ ħ : hull(e) ⊴ ⟃β}
2  H2a := {β ∈ ħ : ¬(hull(e) ⊴ β)}
3  H2b := {β ∈ hull(e) : ¬(ħ ⊴ β)}
4  if H1 ≠ ∅ then
5          // Erster Fall
6          split(e)      := ⋀_{β∈H1} β
7          invariant 1:  ħ ⊴ split(e) ∧ hull(e) ⊴ ⟃split(e)
8          hull(l(e))    := ħ
9          hull(r(e))    := hull(e)
10         m(l(e))       := N
11         m(r(e))       := m(e)
12 else if H2a ≠ ∅ then
13         // Zweiter Fall
14         choose        β ∈ H2a
15         split(e)      := β
16         hull(l(e))    := hull(e) ∩ split(e)
17         hull(r(e))    := hull(e) ∩ ⟃split(e)
18         m(l(e))       := m(e)
19         m(r(e))       := m(e)
20         insertLeaf(l(e), ħ, N)
21 else if H2b ≠ ∅ then
22         // Zweiter Fall, wobei hull(e) und ħ sowie m(e) und N vertauscht werden
23 else
24         // Dritter Fall
25         m(e) := m(e) ∪ N
26         return
27 end if
28 invariant 3:  hull(l(e)) ⊴ split(e) ∧ hull(r(e)) ⊴ ⟃split(e)
29 hull(e)      := hull(l(e)) ⊕ hull(r(e))
30 // Entferne mehrfache Beschränkungen und Markierungen.
31 bound(e)     := hull(e)
32 bound(l(e))  := hull(l(e)) \ (bound(e) ∩ split(e))
33 bound(r(e))  := hull(r(e)) \ (bound(e) ∩ ⟃split(e))
34 m(e)         := m(l(e)) ∩ m(r(e))
35 m(l(e))      := m(l(e)) \ m(e)
36 m(r(e))      := m(r(e)) \ m(e)
```

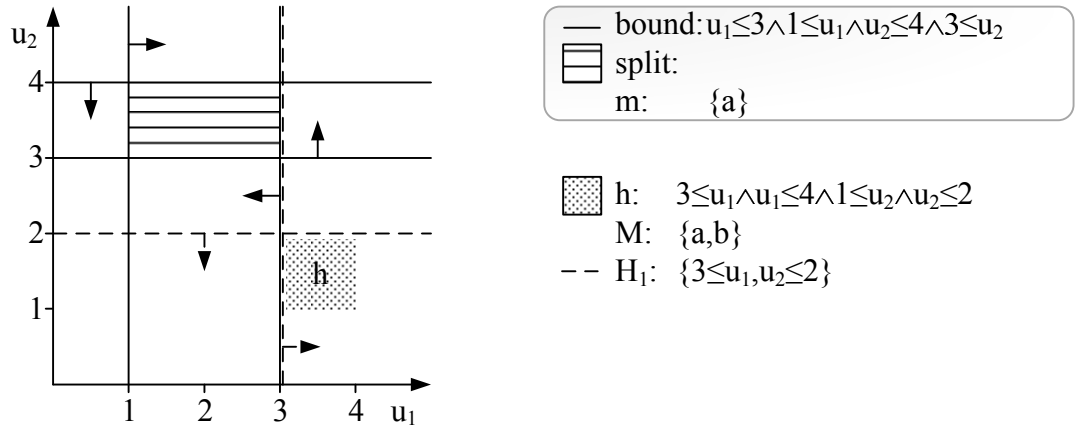


Abbildung 5.3: Einfügen eines neuen Hyperpolyeders in einen Suchbaum, erster Fall.

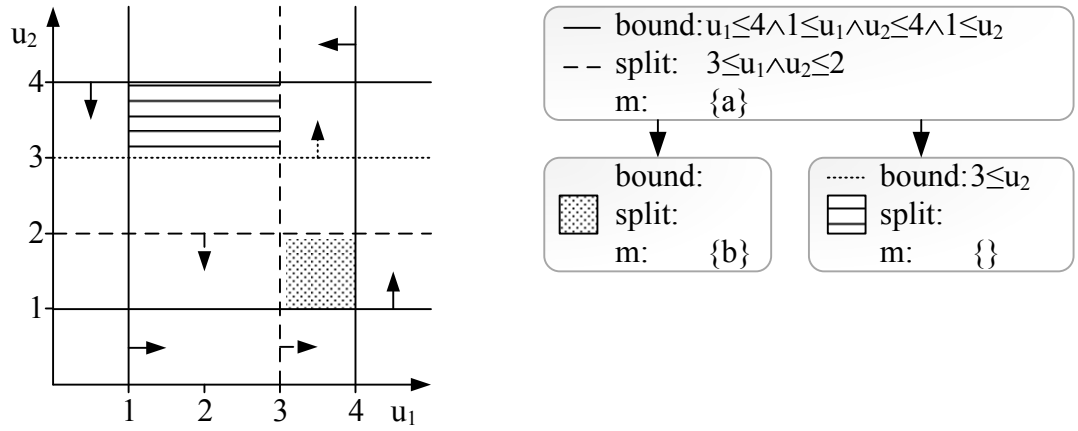


Abbildung 5.4: Einfügen eines neuen Hyperpolyeders in einen Suchbaum, Ergebnis erster Fall.

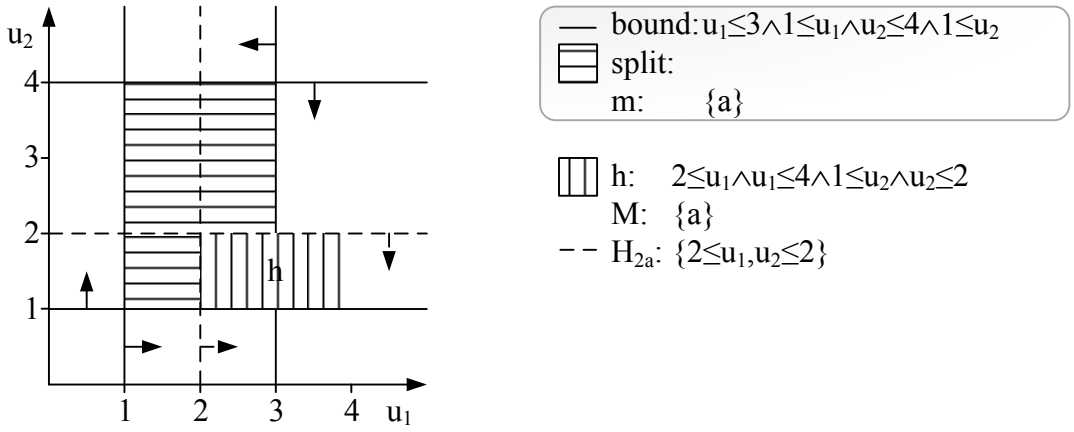


Abbildung 5.5: Einfügen eines neuen Hyperpolyeders in einen Suchbaum, zweiter Fall.

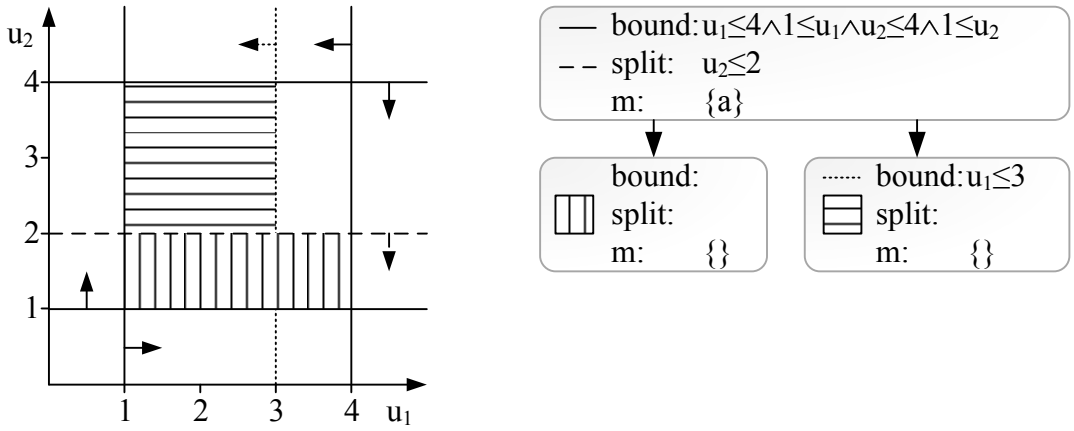


Abbildung 5.6: Einfügen eines neuen Hyperpolyeders in einen Suchbaum, Ergebnis zweiter Fall.

bound(*root*) wieder wie im ersten Fall aufgebaut. Das Ergebnis des Einfügens ohne den rekursiven Aufruf ist in Abbildung 5.5 dargestellt.

Ein Sonderfall tritt auf, wenn $\hbar$ die Hülle von e komplett enthält. In diesem Fall existiert keine geeignete Splitbeschränkung in $\hbar$, sondern nur in der Hülle von e, weshalb dieser Fall mit $\mathcal{H}_{2_b}$ gesondert betrachtet wird.

3. Im dritten Fall sind $\hbar$ und die Hülle von e gleich. Dann muss lediglich die Markierung von e angepasst werden.

Es gilt folgendes Lemma:

Lemma 5.26. *Gegeben sei ein Suchbaum $\mathcal{S}$, dessen Wurzel ein Blatt ist. Dann berechnet Algorithmus 8 eine Suchbaum $\mathcal{S}'$, so dass für alle Blätter e' von $\mathcal{S}'$ gilt:*

$$\bigcup_{e' \in leaves} [\![hull(e')]\!] = [\![hull(e)]\!] \cup [\![\hbar]\!] \tag{5.7}$$

$$hull(e') \trianglelefteq \hbar \Longrightarrow N \subseteq m(e') \tag{5.8}$$

$$hull(e') \trianglelefteq hull(e) \Longrightarrow m(e) \subseteq m(e') \tag{5.9}$$

Beweis. Es werden drei Fälle unterschieden: 1. $\hbar$ und $hull(e)$ überschneiden sich nicht, 2. $\hbar$ und $hull(e)$ überschneiden sich, sind aber nicht gleich und 3. $\hbar$ und $hull(e)$ sind gleich. Entsprechend ist $\mathcal{H}_1$ die Menge derjenigen Beschränkungen von $\hbar$, die $\hbar$ und $hull(e)$ exakt trennen und $\mathcal{H}_{2_{a/b}}$ die Mengen der Beschränkungen von $\hbar$ bzw. $hull(e)$, die den jeweils anderen Hyperpolyeder schneiden.

1. Da $\hbar \trianglelefteq \beta$ und $hull(e) \trianglelefteq \complement\beta$ für alle Beschränkungen $\beta \in \mathcal{H}_1$ gilt, folgt Invariante 1. Im Weiteren werden zwei Blätter $l(e)$ und $r(e)$ erzeugt und deren Hüllen und Markierungen bestimmt. Aus den Zeilen 8 und 9 folgt $[\![hull(l(e))]\!] \cup [\![hull(r(e))]\!] = [\![hull(e)]\!] \cup [\![\hbar]\!]$ und damit Behauptung 5.7. Ferner kann $hull(l(e)) \trianglelefteq \hbar$ aus Zeile 8 sowie $N \subseteq m(l(e))$ aus Zeile 10 gefolgert werden, wodurch mit Invariante 1 Behauptung 5.8 bewiesen wird. Analog kann mit Behauptung 5.9 und den Zeilen 9 und 11 verfahren werden.

2. Es wird angenommen, dass eine Beschränkung $\beta \in \hbar$ existiert, so dass $\neg(hull(e) \trianglelefteq \beta)$. Da Fall 1 nicht zutrifft, gilt $\mathcal{H}_1 = \varnothing$ und es folgt $\neg(hull(e) \trianglelefteq \complement\beta)$. Mit $split(e) = \beta$ (Zeile 15) und $hull(e) \neq \varnothing$ (Definition 5.19) ergibt sich für die zwei erzeugten Blätter $hull(l(e)) = hull(e) \cap split(e) \neq \varnothing$ und $hull(r(e)) = hull(e) \cap \complement split(e) \neq \varnothing$ sowie $[\![hull(l(e))]\!] \cup [\![hull(r(e))]\!] = [\![hull(e)]\!]$ (Zeilen 16 und 17), ferner $m(l(e)) = m(e)$ (Zeile 18) und $m(r(e)) = m(e)$ (Zeile 19), weshalb $l(e)$ und $r(e)$ die Hülle $hull(e)$ auf $m(e)$ abbilden.

   Mit $\hbar \trianglelefteq split(e)$ und dem Bekanntsein der Hülle $hull(l(e))$ und Markierung $m(l(e))$ des linken Blattes sind alle Voraussetzungen erfüllt, um $\hbar$ in $l(e)$ einzufügen (Zeile 20). Aus $hull(l(e)) = hull(e) \cap split(e)$ und Zeile 15 folgt $hull(l(e)) \trianglelefteq \beta$, weshalb β weder in

$\mathcal{H}_{2_a}$ noch in $\mathcal{H}_{2_b}$ von $l(e)$ enthalten sein kann. Da zudem keine Beschränkungen in $\hbar$ eingefügt werden, kann Fall 2 nur endlich oft auftreten und schließlich muss Fall 1 oder Fall 3 zutreffen.

Gilt $\mathcal{H}_{2_a} = \emptyset$, dann gilt für alle Beschränkungen $\beta \in \hbar$: $hull(e) \trianglelefteq \beta$ und es folgt $hull(e) \trianglelefteq \hbar$. Ist $hull(e)$ echt schärfer als $\hbar$, enthält $\mathcal{H}_{2_b}$ mindestens eine Beschränkung und $\hbar$ kann unter Nutzung dieser Beschränkung geteilt werden. Andernfalls gilt $hull(e) = \hbar$.

3. Gilt $hull(e) = \hbar$ folgt trivialerweise Behauptung 5.7 für e. Mit $m(e) := m(e) \cup N$ (Zeile 25) folgt ferner Behauptung 5.8 und 5.9. Da $hull(e)$ nicht verändert wird, bleibt *bounde* unverändert.

Schließlich muss garantiert werden, dass der neue Baum ein Suchbaum ist. Für Fall 3 folgt dies trivialerweise, da der Suchbaum im dritten Fall nicht verändert wird. Für Fall 1 folgt Invariante 3 aus Invariante 1 sowie den Zeilen 8 und 9, für den zweiten Fall aus $hull(e) \cap split(e) \trianglelefteq split(e)$ und $hull(e) \cap \complement split(e) \trianglelefteq \complement split(e)$ (Zeilen 16 und 17). Aus Zeile 29 folgt ferner $hull(l(e)) \trianglelefteq hull(e)$ und $hull(r(e)) \trianglelefteq hull(e)$. Damit ist die Suchbaumeigenschaft für die Hüllen von e, $l(e)$ und $r(e)$ erfüllt.

Aus den Zeilen 31, 32 und 33 folgt schließlich $hull(e) = bound(e)$ sowie $hull(l(e)) = hull(e) \cap split(e) \cap bound(l(e))$ und $hull(r(e)) = hull(e) \cap \complement split(e) \cap bound(r(e))$, womit e die Suchbaumeigenschaften gemäß Definition 5.19 erfüllt. Entsprechend der Beschränkungen werden in den Zeilen 34, 35 und 36 Markierungen aus $m(l(e))$ und $m(r(e))$ in $m(e)$ zusammengefasst. □

Algorithmus 9 fügt einen Hyperpolyeder in einen Suchbaum ein, dessen Wurzel kein Blatt ist. Dazu werden erneut drei Fälle unterschieden:

1. Im ersten Fall wird $\hbar$ von einigen oder allen Beschränkungen aus $split(e)$ beschränkt. Die entsprechenden Beschränkungen sind in der Menge $\mathcal{H}_1$ enhalten.

   Gibt es eine oder mehrere Beschränkungen in $split(e)$, die nicht in $\mathcal{H}_1$ enthalten sind, so werden diese Beschränkungen aus $split(e)$ entfernt und in $bound(l(e))$ bzw. $bound(r(e))$ eingefügt, um die Hüllen der Kindecken nicht zu verändern. Durch das Entfernen von Beschränkungen aus $split(e)$ muss $\hbar$ nicht zerteilt werden, sondern kann direkt in das linke Kind eingefügt werden. Nach dem Einfügen hat sich die Hülle des linken Kindes eventuell vergrößert, weshalb im letzten Schritt auch die Hülle von e angepasst wird.

2. Der zweite Fall verhält sich analog zum ersten, wobei $\hbar$ in die rechte Kindecke eingefügt wird.

3. Im dritten Fall existiert keine Beschränkung in $split(e)$ oder $\complement split(e)$, die $\hbar$ komplett beschränkt. In diesem Fall wird $\hbar$ mit einer beliebigen Beschränkung aus $split(e)$ zerteilt. Anschließend können die zwei Teile von $\hbar$ entsprechend in die linke bzw. rechte Kindecke eingefügt werden. Zuletzt werden die Hüllen wieder entsprechend angepasst.

Folgendes Lemma stellt die Korrektheit von Algorithmus 9 sicher:

Algorithmus 9 *insertInner*$(e, \hbar, N)$

```
1  // Wähle korrekte Splitdimension aus!
2  H_1  := {β ∈ split(e) : ħ ⊴ β}
3  H_2  := {β ∈ split(e) : ħ ⊴ ∁β}
4  if H_1 ≠ ∅ then
5          // Erster Fall
6          split(e)        := ⋀_{β∈H_1} β
7          invariant 1:    ħ ⊴ split(e)
8          bound(l(e))     := bound(l(e)) ∩ ⋀_{β∈split(e)\H_1} β
9          bound(r(e))     := bound(r(e)) ∩ ⋀_{β∈split(e)\H_1} ∁β
10         invariant 2:    hull(l(e)) = hull_pre(l(e)) ∧ hull(r(e)) = hull_pre(r(e))
11         insert(l(e), ħ, N)
12 else if H_2 ≠ ∅ then
13         // Zweiter Fall
14         split(e)        := ⋀_{β∈H_2} β
15         bound(l(e))     := bound(l(e)) ∩ ⋀_{β∈split(e)\H_2} β
16         bound(r(e))     := bound(r(e)) ∩ ⋀_{β∈split(e)\H_2} ∁β
17         insert(r(e), ħ, N)
18 else
19         // Dritter Fall
20         choose          β ∈ split(e)
21         split(e)        := β
22         bound(l(e))     := bound(l(e)) ∩ ⋀_{β'∈split(e)\β} β'
23         bound(r(e))     := bound(r(e)) ∩ ⋀_{β'∈split(e)\β} ∁β'
24         invariant 2:    hull(l(e)) = hull_pre(l(e)) ∧ hull(r(e)) = hull_pre(r(e))
25         insert(l(e), ħ ∩ split(e), N)
26         insert(r(e), ħ ∩ ∁split(e), N)
27 end if
28 invariant 3:   hull(l(e)) ⊴ split(e) ∧ hull(r(e)) ⊴ ∁split(e)
29 hull(e)       := hull(l(e)) ⊕ hull(r(e))
30 // Entferne mehrfache Beschränkungen und Markierungen.
31 bound(e)      := hull(e)
32 bound(l(e))   := hull(l(e)) \ (bound(e) ∩ split(e))
33 bound(r(e))   := hull(r(e)) \ (bound(e) ∩ ∁split(e))
34 m(e)          := m(l(e)) ∩ m(r(e))
35 m(l(e))       := m(l(e)) \ m(e)
36 m(r(e))       := m(r(e)) \ m(e)
```

Lemma 5.27. *Gegeben sei ein Suchbaum $\mathcal{S}$, dessen Wurzel eine innere Ecke e ist, sowie ein Hyperpolyeder $\hbar$ und eine Markierung N. Der Algorithmus 9 fügt $\hbar$ dann in das linke und/oder rechte Kind ein, wobei $[\![\hbar \cap split(e)]\!] \cup [\![\hbar \cap \complement split(e)]\!] = [\![\hbar]\!]$ gilt.*

Beweis. Es werden drei Fälle unterschieden: Der Hyperpolyeder $\hbar$ kann 1. entweder in das linke oder 2. in das rechte Kind eingefügt werden oder $\hbar$ muss 3. auf beide Kinder aufgeteilt werden.

1. Es gibt mindestens eine Beschränkung $\beta \in split(e)$, so dass $\hbar \trianglelefteq \beta$. In diesem Fall folgt Invariante 1 aus der Definition von $\mathcal{H}_1$ und Zeile 6. Die Invariante 2 folgt aus der Definition der Hülle (Definition 5.19) und den Zeilen 8 und 9, wobei $hull_{pre}(e)$ die Hülle einer Ecke vor Ausführung von *insert* bezeichnet.

   Da gemäß Invariante 1 $\hbar$ die Hülle des rechten Kindes nicht schneidet, folgt die Behauptung für $r(e)$. Durch Einfügen von $\hbar$ in $l(e)$ (Zeile 11) wird die Behauptung für das linke Kind sichergestellt. Da ein Suchbaum nur endlich viele innere Ecken besitzt, kann dieser Fall nur endlich oft auftreten.

2. Es gibt mindestens eine Beschränkung $\beta \in split(e)$, so dass $\hbar \trianglelefteq \complement\beta$. Der Beweis dieses Falles gestaltet sich analog dem ersten Fall, wobei $\hbar \trianglelefteq \complement split(e)$ gilt und $\hbar$ in $r(e)$ eingefügt wird.

3. Gilt $\mathcal{H}_1 = \varnothing$ und $\mathcal{H}_2 = \varnothing$, schneidet jede Beschränkung $\beta \in split(e)$ den Hyperpolyeder $\hbar$. Aus Zeile 21 folgt $[\![\hbar \cap split(e)]\!] \cup [\![\hbar \cap \complement split(e)]\!] = [\![\hbar]\!]$ sowie Invariante 2 aus den Zeilen 22 und 23. In den Zeilen 25 und 26 wird schließlich $\hbar \cap split(e)$ in das linke und $\hbar \cap \complement split(e)$ in das rechte Kind eingefügt.

Nach dem Einfügen von $\hbar$ in die Kindecken sind deren Hüllen möglicherweise erweitert. Invariante 3 folgt für Fall 1 und 2 aus Invariante 1 und Invariante 2, für den dritten Fall aus $\hbar \cap split(e) \trianglelefteq split(e)$ und $\hbar \cap \complement split(e) \trianglelefteq \complement split(e)$ sowie Invariante 2. Aus Zeile 29 folgt ferner $hull(l(e)) \trianglelefteq hull(e)$ und $hull(r(e)) \trianglelefteq hull(e)$. Damit ist die Suchbaumeigenschaft für die Hüllen von $e, l(e)$ und $r(e)$ erfüllt. Aus den Zeilen 31, 32 und 33 folgt schließlich $hull(e) = bound(e)$ sowie $hull(l(e)) = hull(e) \cap split(e) \cap bound(l(e))$ und $hull(r(e)) = hull(e) \cap \complement split(e) \cap bound(r(e))$, womit e die Suchbaumeigenschaften gemäß Definition 5.19 erfüllt. Entsprechend der Beschränkungen werden in den Zeilen 34, 35 und 36 Markierungen aus $m(l(e))$ und $m(r(e))$ in $m(e)$ zusammengefasst. □

Mit den Lemmata 5.26 und 5.27 kann die Korrektheit von Algorithmus 7 sichergestellt werden:

Satz 5.28. *Für einen beliebigen Suchbaum $\mathcal{S}$ sowie ein Hyperpolyeder $\hbar$ und eine Markierung*

N erzeugt Algorithmus 9 einen Suchbaum $\mathcal{S}'$, für dessen Blätter gilt:

$$\bigcup_{e' \in leaves} [\![hull(e')]\!] = [\![hull(e)]\!] \cup [\![\hbar]\!]$$
$$hull(e') \trianglelefteq \hbar \Longrightarrow N \subseteq m(e')$$
$$hull(e') \trianglelefteq hull(e) \Longrightarrow m(e) \subseteq m(e')$$

Beweis. Die Behauptung folgt aus den Lemmata 5.26 und 5.27 sowie Zeile 1 in Algorithmus 9. □

Anmerkung 5.29. *Die vorgestellten Algorithmen zum Einfügen von Hyperpolyedern in Suchbäume lassen sich auf vielfältige Weise verbessern. Es ist beispielsweise möglich, durch Wahl geeigneter Splitbeschränkungen die Gesamtzahl der entstehenden Blätter zu reduzieren. Eine weitere wichtige Verbesserung ist das Zusammenfassen von Blättern, deren Hüllen durch einen Hyperpolyeder dargestellt werden können. In praktischen Versuchen konnte durch diese Maßnahme die Eckenzahl von Suchbäumen stark reduziert werden.*

5.3 Zusammenfassung

In diesem Kapitel wurden die zur Umsetzung der Analyse und Translation Validation von SPS-Zeitautomaten notwendigen Datenstrukturen vorgestellt.

Hyperpolyeder sind eine verallgemeinerte Darstellung von *einzelnen Beschränkungen* bis hin zu kompletten *Differenzgrenz-Matrizen*. Spezialfälle, wie normalisierte Differenzgrenz-Matrizen (Differenzgrenz-Matrizen, die nur Grenzbeschränkungen enthalten) oder Minimaldarstellungen (Mengen von Minimalbeschränkungen) wurden mit Hyperpolyedern in einer Datenstruktur zusammengefasst.

Da nur die notwendigen Beschränkungen gespeichert und bei Operationen betrachtet werden, sind *viele Operationen effizienter* und *im schlechtesten Fall genauso effizient* wie mit Differenzgrenz-Matrizen umsetzbar. Die zeitaufwändige Normalisierung von Differenzgrenz-Matrizen entfällt zudem teilweise.

Der größte Vorteil von Hyperpolyedern ist jedoch, dass sie in Indexstrukturen weiterverwendet werden können. Damit sind Hyperpolyeder eine *einheitliche Datenstruktur* für die Analyse von SPS-Zeitautomaten und die Indizierung besuchter Zonen.

Als Indexstruktur wird ein binärer Suchbaum vorgeschlagen, da Entscheidungsdiagramme zur Darstellung von Zonengraphen nur bedingt geeignet sind. Jede innere Ecke des Suchbaums teilt die besuchten Zeitzustände in zwei überschneidungsfreie Kindecken. Die Blätter des Suchbaums enthalten die besuchten Zeitzustände. Folgende Maßnahmen erhöhen die Effizienz von Suchbäumen gegenüber Entscheidungsdiagrammen oder anderen Datenstrukturen:

1. Für jede Ecke grenzt eine *Hülle* die Menge erreichbarer Zeitzustände weiter ein. Die Hülle enthält Beschränkungen, die für alle Kinder gelten und *speichert diese nur einmalig*. Die Anzahl gespeicherter Beschränkungen sinkt dadurch und viele Beschränkungen werden bei Such- oder Einfügeoperationen nur *einmalig betrachtet*. Nach dem gleichen Prinzip wird mit Markierungen verfahren.
2. Hyperpolyeder werden zur Speicherung von Hüllen und zum Trennen der Zeitzustände der Kinder verwendet. Dadurch können im Gegensatz zu anderen Indexstrukturen Zeitzustände nicht nur durch eine, sondern mehrere Beschränkung getrennt werden. Durch diese Maßnahmen und die Verwendung von Hüllen können in jeder Ecke eines Suchbaums mehr Zeitzustände ausgeschlossen werden, als beispielsweise bei den bekannten Entscheidungsdiagrammen. Damit wird die Anzahl der Suchbaumecken und -beschränkungen reduziert.
3. Hyperpolyeder werden einheitlich zur Analyse und Indizierung eingesetzt. Dadurch sind keine zusätzlichen Operationen für Indexstrukturen notwendig und eine *zeitaufwändige Konvertierung* zwischen Differenzgrenz-Matrizen und Indexstruktur *entfällt*.

6 Praktische Umsetzung

Wesentliches Ziel dieser Arbeit ist der praktische Nachweis der Korrektheit einer Übersetzung. Dazu wurden die in den vorangegangenen Kapiteln beschriebenen Techniken praktisch umgesetzt. Das entstandene Programmpaket erlaubt es, ein TNCES-Modell und ST-Programm miteinander zu vergleichen.

Das Programm arbeitet in drei Schritten:

- Im ersten Schritt werden TNCES-Modell und ST-Programm in SPS-Zeitautomaten umgewandelt. In diesem Schritt verhält sich das Programm ähnlich einem Übersetzer. Die erzeugten SPS-Zeitautomaten können auch zur Implementierung des TNCES-Modells auf einer Steuerung genutzt werden.
- Im zweiten Schritt werden die Zonengraphen der SPS-Zeitautomaten aufgebaut und angezeigt. Dabei ist es möglich, zwischen *verschiedenen symbolischen Repräsentationen* zu wählen, zudem werden *Zykluszeiten beachtet*. Das Programm verhält sich in diesem Schritt ähnlich einem Modellprüfer und muss nur noch um eine Komponente zur Prüfung von Eigenschaften auf den erzeugten Zonengraphen erweitert werden.
- Im dritten Schritt werden die Zonengraphen miteinander verglichen. In diesem Schritt findet die eigentliche Translation Validation statt.

Bedingt durch die hohe Komplexität wurde bei der Entwicklung Wert auf die praktischen Durchführbarkeit und einfache Umsetzung gelegt, die Geschwindigkeit war zweitrangig.

Im Folgenden wird beispielhaft gezeigt, wie eine praktische Translation Validation aussehen kann. Dazu wird im ersten Abschnitt ein TNCES-Modell für eine einfache Ampelsteuerung sowie der daraus erzeugte SPS-Zeitautomat vorgestellt. Im zweiten Abschnitt wird das dazugehörige ST-Programm entwickelt. Im dritten Abschnitt wird gezeigt, wie die Zonengraphen der SPS-Zeitautomaten aufgebaut werden können. Im letzten Abschnitt wird schließlich die Translation Validation durchgeführt.

6.1 TNCES-Modelle

TNCES sind für die Entwicklung von Steuerungen konzipierte Petrinetze, die in [Thi02] definiert wurden. TNCES zeichnen sich durch ihren modularen Aufbau aus, der es erlaubt, eine

Steuerung aus mehreren Komponenten zusammenzusetzen. Insbesondere ermöglicht der modulare Aufbau die Modellierung von Steuerung und gesteuertem Prozess, was z.B. bei der Modellprüfung ausgenutzt werden kann, um die Anzahl zu untersuchender Zustände zu begrenzen.

Die Verbindung der einzelnen Module untereinander erfolgt durch sogenannte Ereignis- und Bedingungskanten:

Ereigniskanten verbinden eine Quell- und eine Zieltransition miteinander. Die Zieltransition darf nur dann schalten, wenn die Quelltransition schaltet.

Bedingungskanten verbinden einen Quellplatz mit einer Zieltransition. Die Zieltransition darf nur dann schalten, wenn der Quellplatz eine Marke enthält.

Sperrkanten verbinden wie Bedingungskanten einen Quellplatz mit einer Zieltransition. Im Gegensatz zu Bedingungskanten darf die Zieltransition nur dann schalten, wenn der Quellplatz keine Marken enthält.

Eine Besonderheit von TNCES sind Zeitbedingungen, die in folgendem Beispiel erläutert werden.

6.1.1 Beispiel einer Ampelsteuerung

Abbildung 6.1 stellt ein vereinfachtes TNCES-Modul zur Steuerung einer Ampel grafisch dar. Gesteuert werden eine Auto- und Fußgängerampel, wobei beide Ampeln die gleiche Fahrt- bzw. Laufrichtung besitzen, d.h. zeitgleich Grün anzeigen. Zur Steuerung einer Kreuzung werden deshalb zwei Module benötigt.

Die Fußgängerampel besitzt einen Berührungssensor, der anzeigt, ob ein Fußgänger die Straße überqueren möchte. In diesem Fall schalten sowohl die Auto- als auch die Fußgängerampel auf Grün. Wartet kein Fußgänger, schaltet nur die Autoampel auf Grün und die Grünphase verkürzt sich.

Die Plätze des Moduls repräsentieren die unterschiedlichen Ampelphasen, Rot_A, $Gelb_A$ usw. entsprechen den Farben der Auto-, Rot_F und $Grün_F$ denen der Fußgängerampel. Ausgänge des Moduls wurden aus Gründen der Übersichtlichkeit nicht eingezeichnet. Die Verwendung von Transitionen und Flusskanten entsprechend den bei Petrinetzen üblichen Konventionen [Rei82].

Jeder Platz des Moduls besitzt ein Alter. Das Alter des Platzes ist die Zeit, die seit der letzten Änderung der Markenzahl des Platzes vergangen ist. Zeitbedingungen wurden in Intervallschreibweise an den Flusskanten notiert, Zeiteinheit sind Sekunden. Wurde kein Intervall angegeben, wird implizit $[0; inf]$ angenommen. Eine Transition darf nur dann schalten, wenn das Alter der Plätze innerhalb der angegebenen Intervalle ist. Beispielsweise wird der Platz $Grün_A$ frühestens 2 Sekunden nach dem Platz $RotGelb_A$ markiert.

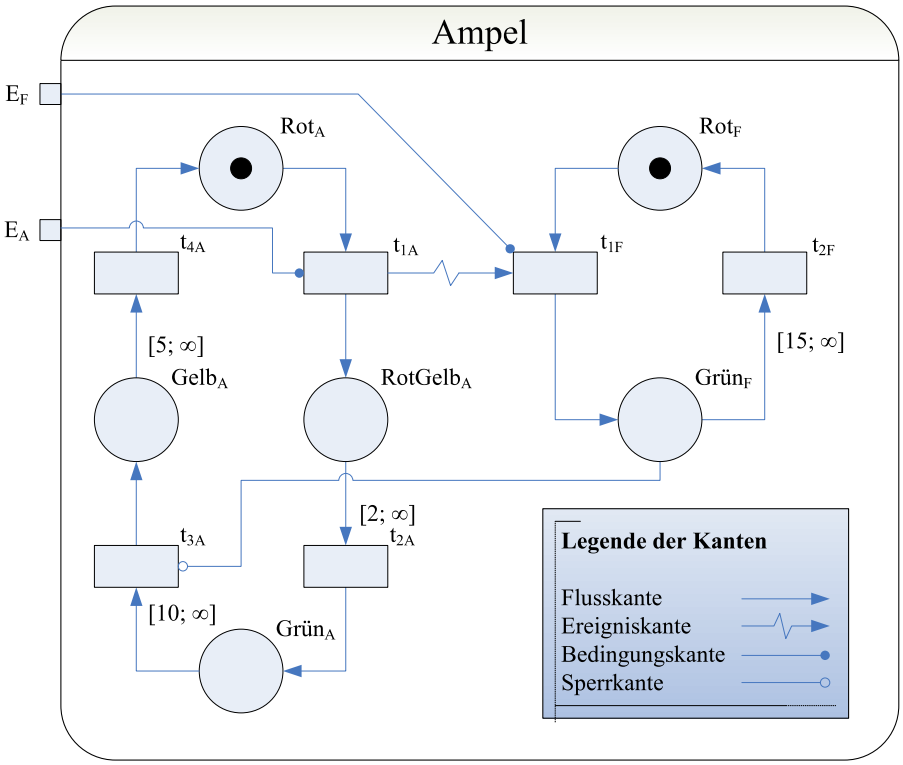


Abbildung 6.1: Modell einer Auto- und Fußgängerampel.

Das abgebildetet Modul besitzt zwei Eingänge, E_A und E_F. Zur Auswertung der Eingänge sowie zur Abstimmung der Auto- und Fußgängerampel wurden Ereignis-, Bedingungs- und Sperrkanten eingesetzt:

- Der Eingang E_A dient der Steuerung der Rotphasen zweier Ampelmodule und zeigt an, ob sich das zweite Ampelmodul im Zustand Rot_A befindet. Durch eine Bedingungskante von E_A nach t_{1A} wird sichergestellt, dass das Modul nur dann $RotGelb_A$ markiert, wenn E_A aktiv ist.
- Der Eingang E_F ist mit dem Berührungssensor der Fußgängerampel verbunden und zeigt an, dass ein Fußgänger die Strasse überqueren möchte. Die Bedingungskante von E_F nach t_{1F} stellt sicher, dass $Grün_F$ nur dann markiert wird, wenn E_F aktiv ist.

Ferner existiert eine Ereigniskante von t_{1A} nach t_{1F}. Die Ereigniskante garantiert, dass t_{1F} nur genau dann schalten darf, wenn t_{1A} schaltet. Durch diese Maßnahme kann die Fußgängerampel nicht unabhängig von der Autoampel Grün anzeigen. Das Verhalten von t_{1A} wird durch die Kante nicht beeinflusst.

Schließlich existiert eine Sperrkante zwischen $Grün_F$ und t_{3A}. Die Transition t_{3A} darf nur dann schalten, wenn $Grün_F$ nicht markiert ist. Somit verlängert sich die Grünphase der Autoampel, sobald auch die Fußgängerampel auf Grün schaltet.

```
1  FUNCTION_BLOCK Ampel
2  VAR_IN
3     EF:       BOOL;
4     EA:       BOOL;
5  END_VAR
6  VAR_OUT
7     CRotA:    BOOL;
8     CGelbA:   BOOL;
9  (...)
10 END_VAR
11 VAR
12    state:    INT    := 0;
13    CLOCKS:   ARRAY[0..3] OF UDINT;
14    SEC:      UDINT  := 1000000000;
15    ZEITEN:   ARRAY[0..3] OF UDINT := [15, 2, 10, 5];
16 END_VAR
17 (...)
```

Programm 6.1: Implementierung der Ampelsteuerung in ST, erster Teil.

6.1.2 Umwandlung in einen SPS-Zeitautomaten

Um das TNCES in einen SPS-Zeitautomaten umzuwandeln, wurde ein Interpreter für TNCES entwickelt. Dieser zählt die erreichbaren Markierungen des TNCES auf. Jede erreichbare Markierung entspricht einem Zustand des späteren SPS-Zeitautomaten.

Die Nutzung eines Interpreters ist technisch anspruchsvoller als die direkte Umwandlung des TNCES in einen SPS-Zeitautomaten, hat aber den Vorteil, nur die tatsächlich erreichbaren Markierungen zu erzeugen. Die Zahl erreichbarer Markierungen liegt meist weit unter der Anzahl möglicher Markierungen des Netzes, die exponentiell in der Anzahl der Plätze steigt.

Der auf diese Weise erzeugte SPS-Zeitautomat ist in Abbildung 6.2 dargestellt.

6.2 Steuerprogramme

Es existieren verschiedene Möglichkeiten, SPS zu programmieren. Die implementierte Translation Validation beschränkt sich auf Steuerprogramme in Structured Text, ist aber auf andere Sprachen erweiterbar. Steuerprogramme können modular aufgebaut werden, wobei jedes Modul über eine Anzahl von Eingangs- (`VAR_IN`) sowie Ausgansvariablen (`VAR_OUT`) verfügt. Module können zudem interne Variablen (`VAR`) besitzen, die den internen Zustand des Moduls darstellen.

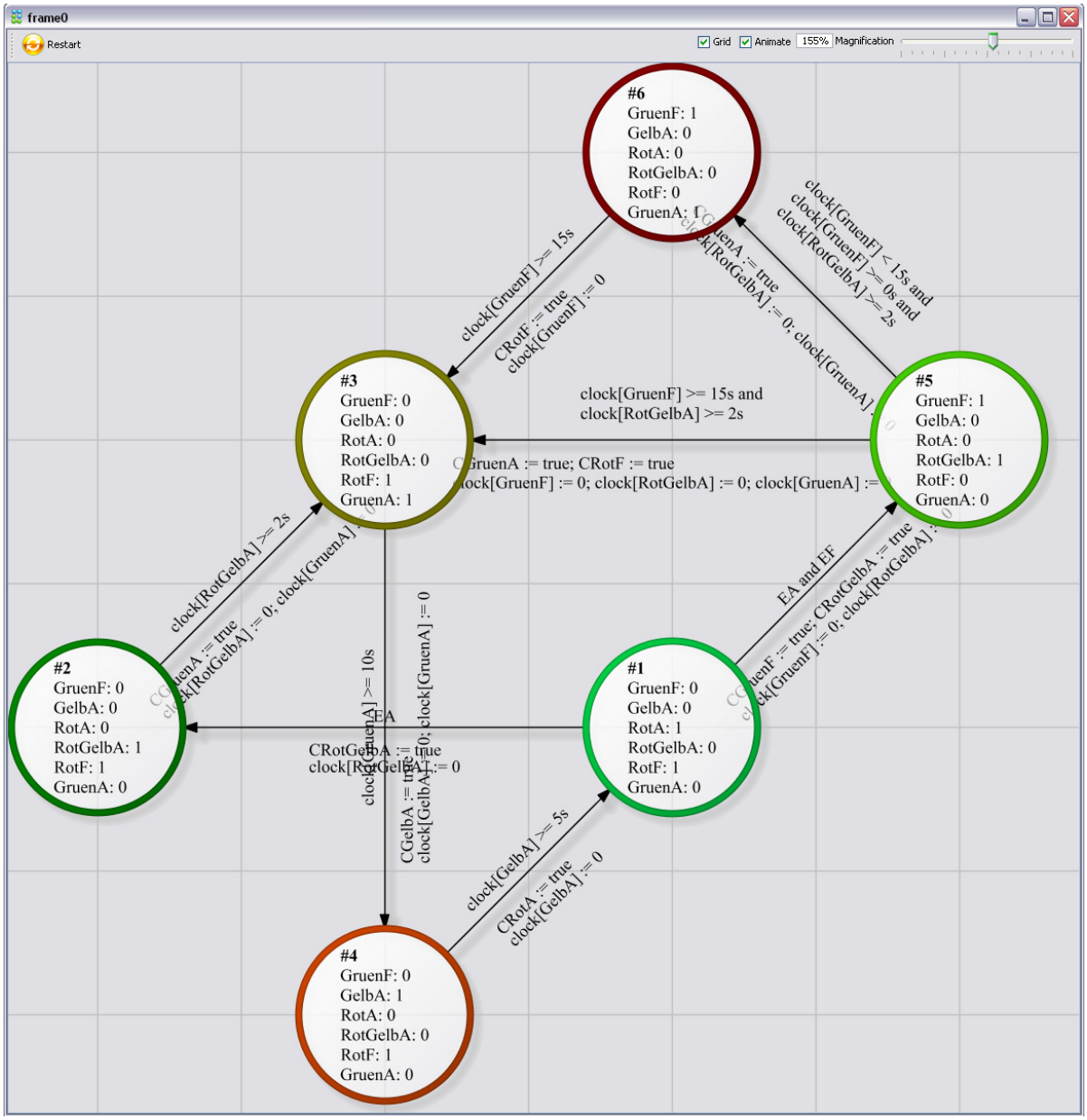


Abbildung 6.2: Aus dem Beispiel extrahierter SPS-Zeitautomat.

```
1  (...)
2  IF state = 0 AND EA AND NOT EF THEN
3     state      := 1;
4     CRotGelbA  := TRUE;
5     RESET_CLOCK(CLOCKS, state);
6  (...)
7  ELSE IF state = 3 THEN
8     IF GET_CLOCK(CLOCKS, state) >= ZEITEN[state]*SEC THEN
9        RESET_CLOCK(CLOCKS, state);
10       state    := 0;
11       CRotA    := TRUE;
12    END_IF;
13 ELSE IF state = 4 THEN
14 (...)
15 END_FUNCTION_BLOCK
```

Programm 6.2: Implementierung der Ampelsteuerung in ST, zweiter Teil.

Programm 6.1 stellt einen Ausschnitt aus der Variablendefinitionen des ST-Programms zur Ampelsteuerung dar. Neben den Eingangsvariablen `EF` und `EA` werden Ausgangsvariablen (`CRotA`, ...) sowie interne Variablen für die Uhrenstände (`CLOCKS`), Zeitenbeschränkungen (`ZEITEN`) und den Zustand der Ampel (`state`) definiert.

Für die weitere Ampelsteuerung wurden in diesem Beispiel die Zustände und Zustandsübergänge des SPS-Zeitautomaten direkt implementiert. Dies ist beispielhaft in Programmausschnitt 6.2 dargestellt.

Umwandlung

Der *Aufbau und die Arbeitsweise* des Steuerprogrammes ist für die Translation Validation *unerheblich*. Beispielsweise kann das Steuerprogramm generisch beliebige TNCES interpretieren oder das Verhalten eines TNCES direkt nachbilden. Eine weitere Möglichkeit ist die Implementierung eines SPS-Zeitautomaten.

Zur Umwandlung eines Steuerprogramms in einen SPS-Zeitautomaten wurde ein Interpreter für Structured Text geschrieben. Der Interpreter versteht die wichtigsten Konstrukte von ST. Beschränkungen bestehen bei der Behandlung von Uhren (Vgl. Abschnitt 3.1.4) sowie bei der Umsetzung der umfangreichen Laufzeitbibliotheken.

Für jede *Belegung der internen Variablen* eines ST-Moduls *am Anfang eines Zyklus* wird ein Zustand des SPS-Zeitautomaten angelegt. Durch die Arbeitsweise einer SPS wird sichergestellt, dass während der Abarbeitung des Steuerprogramms Änderungen an Ein- oder Ausgängen keinen Einfluss auf den gesteuerten Prozess haben. Es ist deshalb nicht notwendig,

Zwischenzustände des Steuerprogramms explizit zu modellieren.

Bei der Umwandlung eines Steuerprogramms in einen SPS-Zeitautomaten werden mehrere Schritte durchlaufen:

- Aus dem ST-Programm wird mit dem Übersetzerbauhilfsprogramm Coco/R ein attributierter Syntaxbaum erzeugt.
- Der Syntaxbaum dient als Eingabe für eine *Abstrakte Zustandsmaschine* (ASM). Diese wird verwendet, um das Steuerprogramm beginnend mit dem Startzustand zu interpretieren:
  - Die ASM simuliert für einen vorgegebenen Zustand einen Zyklus der SPS und ermittelt die Menge möglicher Folgezustände. Welche Folgezustände erreicht werden können, hängt von der Eingangsbelegung und den Uhrenständen ab.
  - Für jeden unbekannten Folgezustand wird ein Zustand des SPS-Zeitautomaten erzeugt.
  - Anschließend werden für jeden noch nicht besuchten Folgezustand die weiteren Folgezustände ermittelt.

Ein wesentlicher Vorteil der Verwendung einer ASM ist die leicht nachzuvollziehende Semantik und die leichte Erweiterbarkeit des Interpreters. Dieses Vorgehen ermöglicht zudem eine leichte Anpassung des Interpreters an herstellerspezifische Unterschiede in der Syntax und Semantik von ST.

6.3 Zonengraphen

In Abbildung 6.3 ist das Hauptfenster des Programmpakets zu sehen. Modell und Implementierung sind auf der linken bzw. rechten Seite als XML- bzw. ST-Programm zu erkennen. Sowohl Modell als auch Implementierung wurden automatisch in SPS-Zeitautomaten umgewandelt.

Um SPS-Zeitautomaten vergleichen zu können, müssen deren Zonengraphen erzeugt werden. Dazu wurde die Vorwärtsanalyse aus Abschnitt 4.3 implementiert. Bei der Erzeugung der Zonengraphen kann die untere und die obere Zykluszeit angegeben werden sowie die zu verwendende Datenstruktur (Differenzgrenz-Matrizen oder Suchbäume) ausgewählt werden. Der entsprechende Programmdialog ist in Abbildung 6.4 dargestellt.

Die folgenden Zeilen enthalten die Ausgabe für die Konstruktion des SPS-Zeitautomaten und Zonengraphen für das Ampelbeispiel als TNCES (`AmpelAF_3.xml`) sowie für die dazugehörige ST-Implementierung (`AmpelAF_3.st`).

```
Opening file: AmpelAF_3.xml

Trying to interpret as XML: ok.
Parsing TNCES module: ok.
```

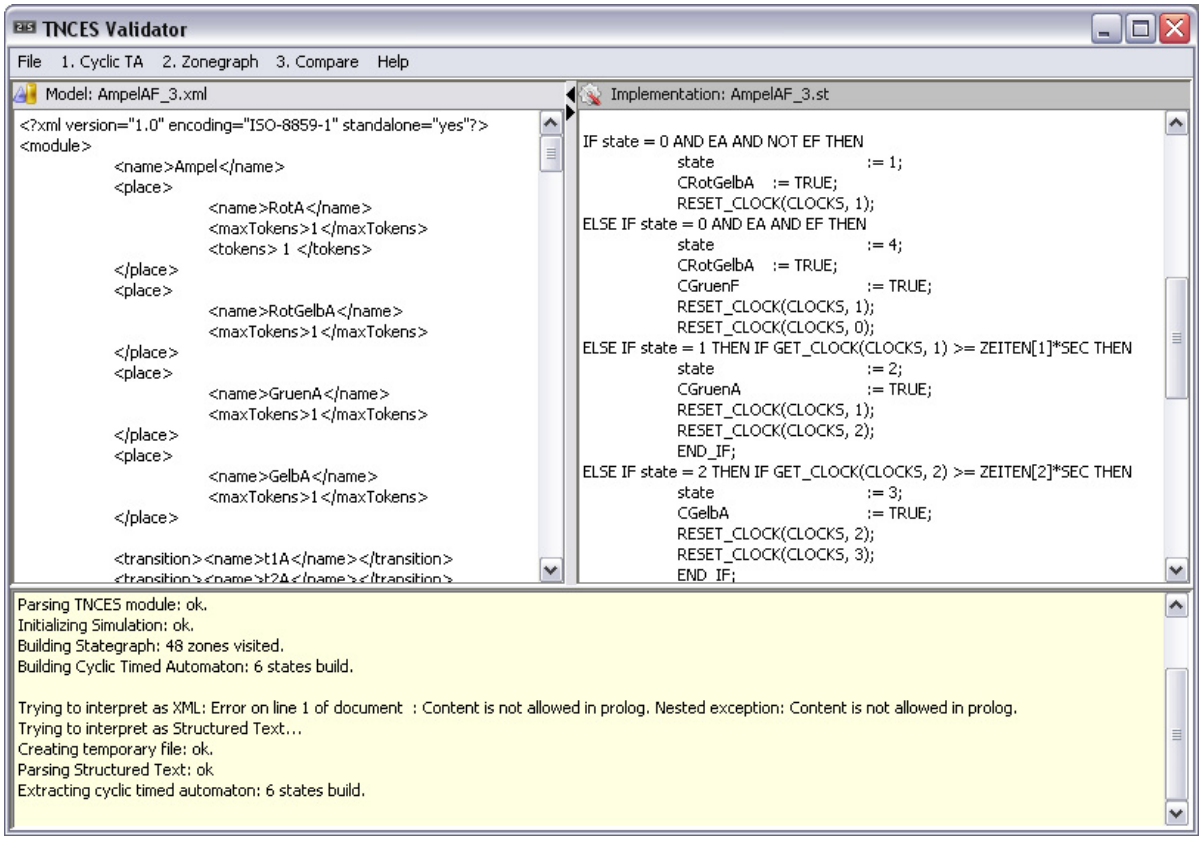


Abbildung 6.3: Programmpaket zur Translation Validation von TNCES und ST-Programmen.

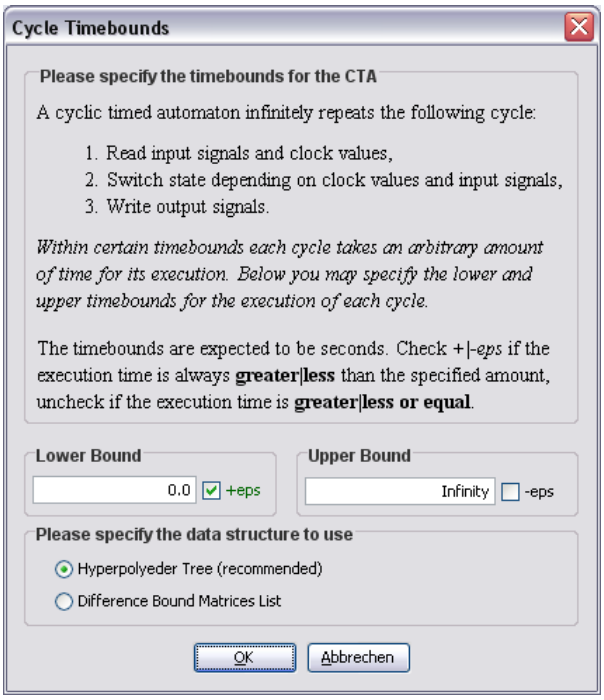


Abbildung 6.4: Dialog zum Aufbau des Zonengraphen.

```
Initializing Simulation: ok.
Building Stategraph: 48 zones visited.
Building Cyclic Timed Automaton: 6 states build.

Enumerating zones: 430 zones visited in 1,531 seconds.
The zonegraph contains 94 polyhedra with 238 restrictions.

Opening file: AmpelAF_3.st

Trying to interpret as XML: Error on line 1 of document  : Content is not ...
Trying to interpret as Structured Text...
Creating temporary file: ok.
Parsing Structured Text: ok
Extracting cyclic timed automaton: 6 states build.

Enumerating zones: 430 zones visited in 1,954 seconds.
The zonegraph contains 65 polyhedra with 188 restrictions.
```

Der Zonengraph des TNCES konnte durch die Verwendung von Suchbäumen mit 238 Beschränkungen, der Zonengraph des ST mit 188 Beschränkungen dargestellt werden. Werden statt der Suchbäume DBMs zur Darstellung des Zonengraphen verwendet, erhöht sich die Anzahl der Beschränkungen deutlich:

```
Enumerating zones: 430 zones visited in 1,282 seconds.
The zonegraph contains 430 polyhedra with 8600 restrictions.

Enumerating zones: 430 zones visited in 1,937 seconds.
The zonegraph contains 430 polyhedra with 8600 restrictions.
```

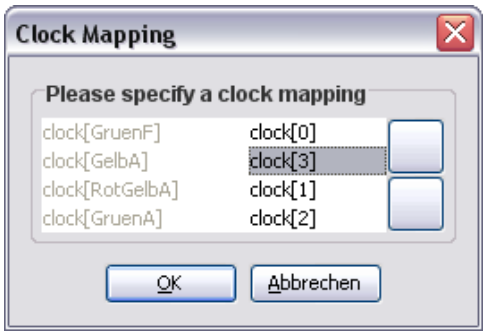


Abbildung 6.5: Mapping zwischen den Uhren des TNCES und ST-Programms.

In diesem Fall werden zur Darstellung beider Zonengraphen jeweils 8600 Beschränkungen benötigt.

Zur Veranschaulichung wurde eine Bibliothek geschrieben, die einen Zonengraphen layoutet und visualisiert. Die Abbildungen 6.6 und 6.7 stellen den Zonengraphen dar, der aus dem TNCES der Auto- und Fußgängerampel extrahiert wurde.

6.4 Translation Validation

Zur Umsetzung der Translation Validation wurde der Algorithmus aus Abschnitt 4.4.2 implementiert. Um den Vergleich auszuführen muss ein Mapping der Uhren zwischen Quelle und Ziel angegeben werden. Dieses Mapping ist in Abbildung 6.5 für die aus dem TNCES-Modul und ST-Programm extrahierten SPS-Zeitautomaten angegeben.

Danach versucht das System, eine Simulationsrelation auf den Zuständen der beiden SPS-Zeitautomaten nachzuweisen. Gelingt dies, wird eine entsprechende Meldung ausgegeben:

```
Checking simulation relation...ok.
The following simulation relation was found:
0 simulates [Location #1: Tokens{GruenF: 0, GelbA: 0, RotA: 1, RotGelbA: 0, RotF: 1, GruenA: 0}]
1 simulates [Location #2: Tokens{GruenF: 1, GelbA: 0, RotA: 0, RotGelbA: 1, RotF: 0, GruenA: 0}]
2 simulates [Location #6: Tokens{GruenF: 0, GelbA: 0, RotA: 0, RotGelbA: 1, RotF: 1, GruenA: 0}]
3 simulates [Location #3: Tokens{GruenF: 1, GelbA: 0, RotA: 0, RotGelbA: 0, RotF: 0, GruenA: 1}]
4 simulates [Location #4: Tokens{GruenF: 0, GelbA: 0, RotA: 0, RotGelbA: 0, RotF: 1, GruenA: 1}]
5 simulates [Location #5: Tokens{GruenF: 0, GelbA: 1, RotA: 0, RotGelbA: 0, RotF: 1, GruenA: 0}]
```

Misslingt der Nachweis der Simulationsrelation, wird dies ebenfalls ausgegeben. Der Grund des misslungenen Nachweises kann anschließend durch eine Analyse des Nachweises erfolgen. Das System selbst kann den Grund des Scheiterns nicht ermitteln und in Form eines Gegenbeispiels etc. ausgeben, da a priori nicht klar ist, welche Zuordnung der Zustände gefunden werden sollte.

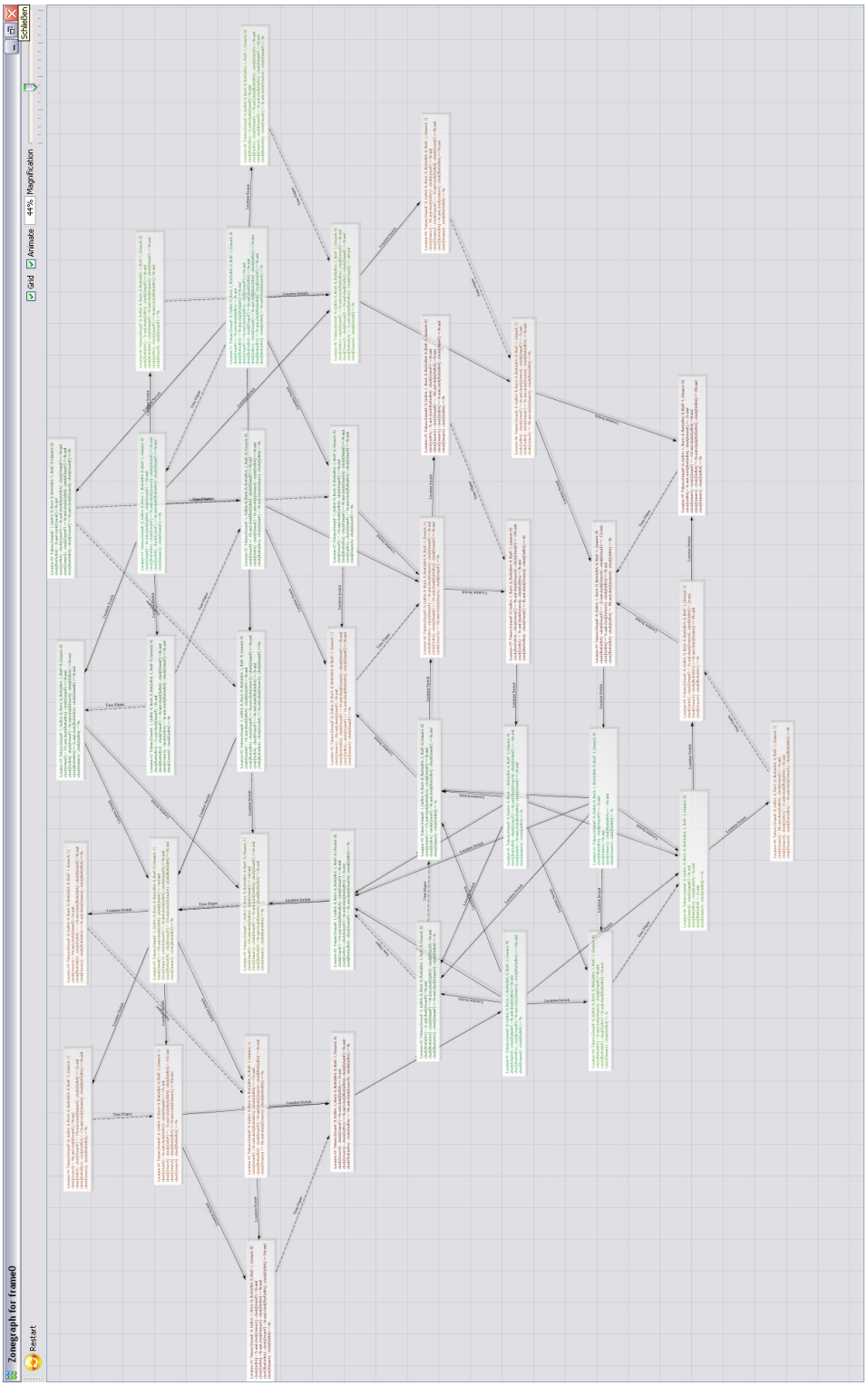


Abbildung 6.6: Gesamter Zonengraph des Ampel-TNCES.

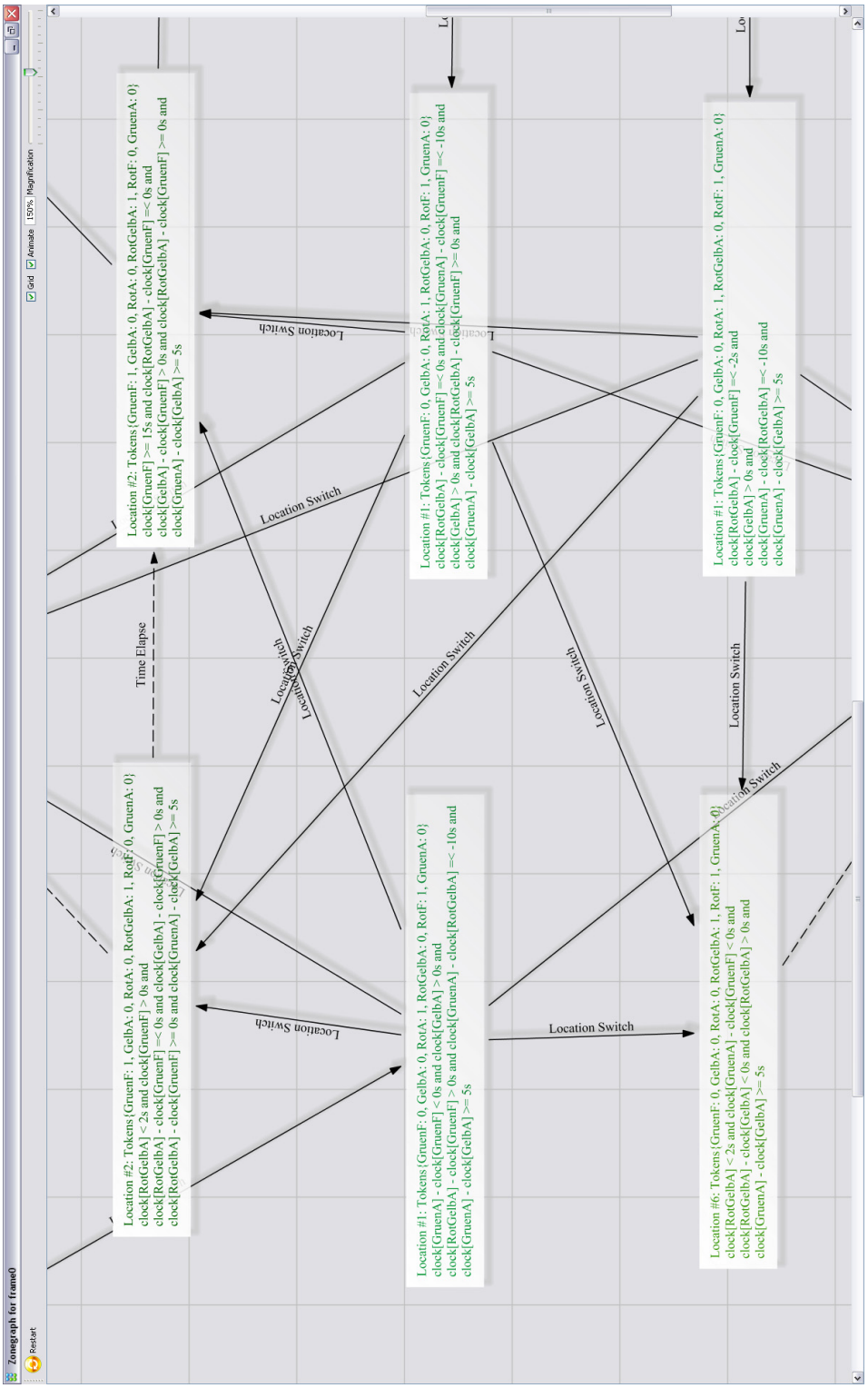


Abbildung 6.7: Ausschnitt des Zonengraphen aus Abbildung 6.6.

Neben einer fehlerhaften Implementierung können auch die angegebenen Zykluszeiten Grund für einen gescheiterten Nachweis sein. So kann es beispielsweise vorkommen, dass Zustände, die im Modell bei einer hohen Zykluszeit theoretisch erreicht werden können, in der Implementierung nicht beachtet wurden. In diesem Fall gelingt ein Nachweis nur, wenn bei der Konstruktion des Zonengraphen die Zykluszeit entsprechend eingegrenzt wird.

6.5 Zusammenfassung

In diesem Kapitel wurde ein Programmpaket zur praktischen Translation Validation eines ST-Programms und TNCES-Moduls vorgestellt. Das Programmpaket umfasst im Wesentlichen vier Komponenten:

1. Ein TNCES-Interpreter erzeugt unter Beachtung der Zeitbeschränkungen die Menge der erreichbaren Markierungen eines TNCES. Für jede erreichbare Markierung wird ein Zustand eines SPS-Zeitautomaten mit den entsprechenden Zustandsübergängen generiert.
2. Ein ST-Interpreter zählt die erreichbaren Variablenbelegungen eines ST-Moduls auf. Zeitbedingungen werden dabei ebenfalls beachtet. Für jede Variablenbelegung wird ein Zustand eines SPS-Zeitautomaten mit den dazugehörigen Zustandsübergängen generiert. Der ST-Interpreter basiert auf dem Übersetzerbauprogramm Coco/R und einer abstrakten Zustandsmaschine.
3. Eine Implementierung der Vorwärtsanalyse baut die Zonengraphen der erzeugten SPS-Zeitautomaten auf. Als Indexstruktur für den Zonengraphen können die Suchbäume aus Abschnitt 5.2 oder alternativ Listen von Differenzgrenz-Matrizen verwendet werden. Bei dem vorgestellten Beispiel reduzierte sich die Gesamtzahl der Beschränkungen des Zonengraphen von 8600 bei der Verwendung von Listen von Differenzgrenz-Matrizen auf 188 bzw. 238 Beschränkungen bei der Verwendung von Suchbäumen. Zudem können die Zykluszeiten der SPS bei der Vorwärtsanlyse eingegrenzt werden.
4. Zuletzt erlaubt eine Implementierung des Algorithmus aus Abschnitt 4.4.2 den Vergleich der Zonengraphen von Modell und Implementierung. Konnte eine vollständige Simulation für Modell und Implementierung gefunden werden, wird dies ausgegeben und die Translation Validation ist beendet. Andernfalls kann das Scheitern der Translation Validation mit Hilfe der Logdatei nachvollzogen werden. Aufgrund des Nachweises der Simulation auf den Zuständen der SPS-Zeitautomaten ist es nicht notwendig, das Produkt der Zonengraphen von Modell und Implementierung zu bilden. Die Translation Validation konnte im angegeben Beispiel deshalb unterhalb einer Sekunde durchgeführt werden.

Neben der Translation Validation erlaubt das Programmpaket zur besseren Nachvollziehbarkeit die grafische Darstellung von SPS-Zeitautomaten und Zonengraphen.

7 Zusammenfassung

Diese Arbeit beschäftigt sich mit der Translation Validation realer Steuerungen mit Zeitbedingungen. Um dieses Ziel zu erreichen, wurde auf das Wissen aus dem Bereich von Zeitsystemen zurückgegriffen, wobei bestehende Techniken und Datenstrukturen angepasst und neue entwickelt wurden. Die Translation Validation erfüllt besondere Anforderungen, die sich durch die Verwendung realer Steuerungen, realer Zeitbedingungen und realer Modelle ergeben:

Reale Steuerungen Als Prozessrechner werden Speicherprogrammierbare Steuerungen betrachtet, die in Structured Text programmiert werden. Zur Entwicklung der Steuermodelle werden TNCES, eine modulare Petrinetzvariante mit Zeitbedingungen, genutzt. Die Steuerung arbeitet mit *Signalen zur Ein- und Ausgabe*, die über einen Zeitraum aktiviert sein können. Dabei wird die Überlappung von Signalen ebenfalls beachtet.

Reale Zeitbedingungen Die meisten Modelle mit Zeitbedingungen treffen idealisierte Annahmen über die verwendeten Uhren und den Zeitbegriff. Diese Annahmen treffen in der Realität allerdings nicht zu. In dieser Arbeit wird deshalb die zyklische Arbeitsweise Speicherprogrammierbarer Steuerungen explizit beachtet. Dabei werden die minimale und maximale Zykluszeit sowie das schrittweise Voranschreiten der Steuerung in der Zeit korrekt modelliert.

Reale Modelle Die Translation Validation ist so aufgebaut, dass die Implementierung von Modellen, deren Eigenschaften mit einem Modellprüfer verifiziert wurden, validiert werden kann. Das wird dadurch erreicht, dass *Techniken aus dem Bereich der Modellprüfung* von Zeitsystemen für die Translation Validation genutzt werden. Zudem wird ein Verfahren zum Vergleich von Zeitmodellen vorgeschlagen, welches mit dem *gleichen oder einem unwesentlich höheren Aufwand* als die Modellprüfung realisiert werden kann. Diese Eigenschaften erlauben die Translation Validation für *reale Modellgrößen*.

Der komplexen Semantik der vorgegebenen Steuermodelle und -programme wird durch die Verwendung von SPS-Zeitautomaten als einfaches internes Basismodell begegnet. SPS-Zeitautomaten besitzen die Eigenschaften der Steuermodelle und -programme, was ein explizites Modellieren dieser Eigenschaften durch Zwischenzustände unnötig macht. Die Anzahl der Modellzustände und damit *die Modellkomplexität steigt während der Translation Validation deshalb nicht an*, was die Verarbeitung *realer Modellgrößen* ermöglicht.

Um die Semantik realer Steuerungen abbilden zu können, besitzen SPS-Zeitautomaten die folgenden Eigenschaften:

Eigenschaft	Zeitauto-maten [AD94]	AASAP-Zeitautom. [WDR04]	timed i/o automata [KLSV06]	SPS-Autom. [Die00]	TNCES [Thi02]	SPS-Zeit-automaten [diese Arbeit]
Mehrere Uhren	✓	✓	✓	–	✓	✓
Ausgabe	–	–	✓	✓	✓	✓
Signalüberlappung	–	–	–	–	✓	✓
ϵ-Zustandsübergänge	–	–	?	–	✓	✓
Zykl. Arbeitsweise	–	–	–	✓	–	✓
Obere Zeitschranke	–	✓	–	✓	–	✓
Untere Zeitschranke	–	–	–	–	–	✓

Tabelle 7.1: Eigenschaften verschiedener Modelle mit Zeitbeschränkungen

Mehrere Uhren Um die Zeitbedingungen von TNCES nachzubilden, können SPS-Zeitautomaten *mehrere Uhren in Zeitbedingungen* verwenden.

Ausgabe SPS-Zeitautomaten können *Signale ausgeben*, was für die Modellierung einer Steuerung unentbehrlich ist.

Signalüberlappung Die von SPS-Zeitautomaten betrachteten *Signale können sich überlappen*. Damit können SPS-Zeitautomaten wie TNCES und SPS auf Änderungen an mehreren Eingängen gleichzeitig reagieren.

ϵ-Zustandsübergänge SPS-Zeitautomaten können wie TNCES oder ST-Programme *Zustandsübergänge auch ohne Eingabe ausführen*.

Zykl. Arbeitsweise Die *zyklische Arbeitsweise* Speicherprogrammierbarer Steuerungen wird von SPS-Zeitautomaten direkt beachtet.

Obere Zeitschranke SPS-Zeitautomaten beachten eine *obere Zeitschranke für die Zykluszeiten* einer SPS.

Untere Zeitschranke Für die Erreichbarkeit von Zeitzuständen und den Zeitfortschritt ist die untere Zeitschranke ebenfalls wichtig. SPS-Zeitautomaten beachten auch eine *untere Zeitschranke für die Zykluszeit* einer SPS.

Tabelle 7.1 stellt eine Übersicht dieser Eigenschaften für vergleichbare Modelle zusammen.

7.1 Analyse und Vergleich

Um zwei SPS-Zeitautomaten miteinander zu vergleichen, werden die Transitionssysteme, die das Verhalten der SPS-Zeitautomaten beschreiben, durch *Zonengraphen symbolisch repräsentiert*. Eine *angepasste Variante der Vorwärtsanalyse* erlaubt die direkte Konstruktion des Zonengraphen eines SPS-Zeitautomaten. Die Zonengraphen können anschließend *bezüglich*

eines Korrektheitsbegriffs verglichen werden. Dabei konnten einige Verbesserungen erreicht werden:

- Der Korrektheitsbegriff ordnet jedem Zustand der Implementierung einen oder mehrere Zustände des Modells zu. Dabei bilden die Zustandsübergänge eines Zustands der Implementierung das Verhalten der entsprechenden Zustandsübergänge des Modells nach.

  Eine Simulation garantiert, dass ein Zustand der Implementierung *höchstens* die Zustandsübergänge des entsprechenden Modellzustands besitzt. Eine Bisimulation garantiert dagegen, dass ein Zustand der Implementierung *alle* Zustandsübergänge des entsprechenden Modellzustands besitzt.

  In der Praxis ist die Simulation als Korrektheitsbegriff zu schwach, da auch Implementierungen akzeptiert werden, die *überhaupt keine Zustandsübergänge* besitzen. Die Bisimulation ist zu stark, da sie bei nichtdeterministischen Modellen fordert, dass alle Zustandsübergänge auch in der Implementierung vorhanden sind. Um diese Probleme zu vermeiden wird als Korrektheitsbegriff die *vollständige Simulation* vorgeschlagen. Eine vollständige Simulation garantiert, dass jeder Zustand der Implementierung *mindestens einen* der Zustandsübergänge und *höchstens alle* Zustandsübergänge des entsprechenden Modellzustandes besitzt. Somit bildet die Implementierung mindestens ein gültiges Verhalten des Modells nach.

- Zur Prüfung einer vollständigen Simulation wird ein einfacher, aber effizienter Algorithmus präsentiert. Der Algorithmus hat nur eine unwesentlich schlechtere Laufzeit, ist aber deutlich einfacher umzusetzen und zu beweisen als die schnellsten bekannten Algorithmen zum Nachweis einer Simulation.

- Um eine Simulation auf den Zeitzuständen zweier Modell nachzuweisen, müssen die Zeitzustände beider Modelle *paarweise kombiniert* werden. Da bereits die Ermittlung der erreichbaren Zeitzustände in Platz und Laufzeit komplex ist, ist die symbolische Repräsentation der Zeitzustandspaare *praktisch meist nicht mehr möglich*.

  Der Nachweis der Simulationsrelation wird deshalb auf den Zuständen der SPS-Zeitautomaten durchgeführt, da die Anzahl der Zustände der SPS-Zeitautomaten meist deutlich unter der Anzahl der besuchten Zonen liegt. Nachteil dieser Methode ist die Notwendigkeit eines Mappings zwischen den Uhren von Modell und Implementierung.

7.2 Datenstrukturen

Zur Analyse und zum Vergleich von SPS-Zeitautomaten wurden neue Datenstrukturen vorgeschlagen. Mit Hyperpolyedern und den Operationen Schnitt und Abschluss wurde ein Verband definiert, der Beschränkungen und Differenzgrenz-Matrizen verallgemeinert. Die zur *Analyse von SPS-Zeitautomaten notwendigen Operationen* können direkt auf Hyperpolyedern abgebildet werden. Zudem wurden mit Hyperpolyedern Suchbäume definiert, die zur *Indizierung von*

Eigenschaft	DBM-Liste [Ben01]	gepackte DBMs [Ben01]	DDD [MLAH99]	CDD [BLP+99]	CRD [Wan04]	Suchbäume [diese Arbeit]
Zonengraphen	✓	✓	–	–	–	✓
Keine Konv.	✓	–	–	–	–	✓
Split	–	–	✓	✓	✓	✓
Split-Beschr.	–	–	2	1	1	∞
Zus. Beschr.	–	–	–	–	–	∞

Tabelle 7.2: Eigenschaften verschiedener Datenstruktur zur Speicherung besuchter Zonen

Zonengraphen geeignet sind. Suchbäume haben folgende Eigenschaften:

Zonengraphen Mit einem Suchbaum ist es möglich, nicht nur die besuchten Zonen sondern auch deren Nachfolger, d.h. den Zonengraphen, darzustellen.

Keine Konvertierung Durch die einheitliche Verwendung von Hyperpolyedern für die Vorwärtsanalyse und in Suchbäumen ist es nicht notwendig, Datenstrukturen zu konvertieren.

Split Jede innere Ecke eines Suchbaums *teilt die Menge der indizierten Zeitzustände* in zwei disjunkte Mengen.

Split-Beschränkungen Zur Teilung des Zeitzustände können *beliebig viele Beschränkungen* verwendet werden. Dadurch sinkt der zu durchsuchende Zustandsraum in jeder Ecke. Beschränkungen müssen dabei nur einmalig gespeichert werden.

Zusätzliche Beschränkungen Jede Ecke eines Suchbaums kann die indizierten Zeitzustände durch Beschränkungen, die nicht zur Teilung geeignet sind, *zusätzlich eingrenzen*. Derartige Beschränkungen entfallen in Kindecken und werden nur einmalig gespeichert.

In Tabelle 7.2 werden diese Eigenschaften mit anderen Datenstrukturen zur Darstellung besuchter Zonen verglichen.

7.3 Praktische Umsetzung

Für die praktische Durchführung einer Translation Validation wurden die in dieser Arbeit vorgestellten Datenstrukturen und Algorithmen in einem Programmpaket umgesetzt. Dieses Paket umfasst:

- Eine programminterne Darstellung von SPS-Zeitautomaten.
- Die Implementierung von Hyperpolyedern und Suchbäumen.
- Eine Implementierung der angegeben Vorwärtsanalyse zum Aufbau von Zonengraphen.

- Algorithmen zur Translation Validation der Zonengraphen.

Zusätzlich wurden noch folgende Komponenten bereitgestellt, die in dieser Arbeit nicht näher beschrieben werden:

- Ein Interpreter für TNCES, der unter Beachtung von Zeitbedingungen die erreichbaren Markierungen eines TNCES ermittelt und einen äquivalenten SPS-Zeitautomaten erzeugt.
- Ein Interpreter für Structured Text, der unter Beachtung von Zeitbedingungen die erreichbaren Variablenbelegungen eines ST-Moduls ermittelt und einen äquivalenten SPS-Zeitautomaten erzeugt.
- Eine graphische Oberfläche zur Benutzung des Programmpakets.
- Ein spezieller Layoutalgorithmus für Graphen und eine Bibliothek zur grafischen Darstellung der erzeugten SPS-Zeitautomaten und Zonengraphen.

7.4 Ausblick

Die Nutzung und Weiterentwicklung der Erkenntnisse dieser Arbeit lässt sich in mehrere Richtungen weiterführen. Für die Verbesserung des praktischen Einsatzes wäre insbesondere die Entwicklung weiterer Hilfsprogramme für die Entwickler von Steuerungen interessant. Da für die Translation Validation in dieser Arbeit bereits die erreichbaren Zeitzustände von Modell und Implementierung aufgezählt werden, wären diese Informationen für weitere Programme leicht nutzbar. Besonders interessant wäre die Entwicklung eines Modellprüfers, der die Verifikation von SPS-Programmen sowie TNCES- und ähnlichen Modellen unter realen Zeitbedingungen ermöglicht. Da die Aufzählung der Zeitzustände von Zeitsystemen bereits zur Translation Validierung verwendet wird, wäre dazu im Wesentlichen eine Erweiterung des Systems um Methoden zur Prüfung logischer Ausdrücke wie TCTL-Formeln notwendig. Eine weitere Anwendung wäre die (Weiter-)Entwicklung eines Übersetzers, der die Informationen bei der Modellanalyse nutzt, um optimierte Implementierungen von Steuermodellen zu erstellen. Umgekehrt können auf dem selben Wege kompakte Modelle bestehender Implementierungen erzeugt werden. Auf diese Weise könnten beispielsweise SPS-Programme mit Modellprüfern verifiziert werden.

Eine weitere Zielstellung könnte die Anpassung der Ergebnisse an andere Systeme und Anforderungen sein. Beispielsweise könnte eine effizientere Darstellung oder Abstraktion der diskreten Zustände die Analyse von Modellen oder Implementierung mit sehr großen oder unendlichen Zustandsräumen ermöglichen. Auch die Verwendung beispielsweise reellwertiger Ein- und Ausgänge würde weitere Anwendungsbereiche für die Modelle erschließen. Darüber hinaus würde insbesondere die Betrachtung von Uhren, die eine Drift besitzen, die Analyse von Systemen ermöglichen, deren Komponenten über verschiedene Uhren verfügen. Schließlich ist eine Übertragung der Erkenntnisse dieser Arbeit auf weitere Zielplattformen relativ

einfach möglich und würde die Anwendbarkeit der Hilfsprogramme weiter erhöhen. Neben den betrachteten Systemen mit zyklischer Arbeitsweise wäre ein Übertragung der Ergebnisse auf Systeme mit ereignisorientierter Arbeitsweise interessant. Die Semantik solcher Systeme ist mit den vorgestellten Mitteln relativ leicht nachbildbar.

Um den Einsatz des Programmpakets im produktiven Einsatz beispielsweise in der Industrie zu ermöglichen, ist eine weitere Steigerung der Geschwindigkeit wünschenswert. Neben der Verbesserung der bestehenden Implentierung könnte dieses Ziel beispielsweise durch eine weitere Verbesserung der verwendeten Datenstrukturen erfolgen. Dazu wäre insbesondere eine Rückkopplung der Steuerstrecke zur Optimierung der Größe der Implementierung sowie des entstehenden Zustandsraumes von Interesse.

Literaturverzeichnis

[ABK+97] E. Asarin, M. Bozga, A. Kerbrat, O. Maler, A. Pnueli, and A. Rasse.
Data Structures for the Verification of Timed Automata.
In Oded Maler, editor, *Proceedings of HART'97 (Grenoble, France)*, volume 1201 of *LNCS*, pages 346–360. Springer-Verlag, April 1997.

[ACD90] R. Alur, C. Courcoubetis, and D.L. Dill.
Model checking for real-time systems.
In *Proceedings* 5^{th} *Annual Symposium on Logic in Computer Science,* Philadelphia, USA, pages 414–425, 1990.

[ACH94] R. Alur, C. Courcoubetis, and T. A. Henzinger.
The observational power of clocks.
In *International Conference on Concurrency Theory*, pages 162–177, 1994.

[AD90] R. Alur and D. L. Dill.
Automata for modeling real-time systems.
In *Proceedings of the seventeenth international colloquium on Automata, languages and programming*, pages 322–335, New York, NY, USA, 1990. Springer-Verlag New York, Inc.

[AD94] R. Alur and D. L. Dill.
A theory of timed automata.
Theoretic Computer Science, 126(2), 1994.

[Alu99] R. Alur.
Timed automata.
In *Proceedings of CAV'99, Trento, Italy*, LNCS 1633, pages 8–22. Springer Verlag, 1999.

[AN01] Parosh Aziz Abdulla and Aletta Nylén.
Timed Petri nets and BQOs.
Lecture Notes in Computer Science, 2075:53–??, 2001.

[BBFL03] G. Behrmann, P. Bouyer, E. Fleury, and K. Larsen.
Static guard analysis in timed automata verification, 2003.

[BCH+05] B. Berard, F. Cassez, S. Haddad, D. Lime, and OH Roux.
Comparison of Different Semantics for Time Petri Nets.
Proc. ATVA05, 3707:293–307, 2005.

[BD00] B. Berard and C. Dufourd.
Timed automata and additive clock constraints.
Information Processing Letters, 75(1-2):1–7, 2000.

[BDFP04] P. Bouyer, C. Dufourd, E. Fleury, and A. Petit.
Updatable timed automata.
Theoretical Computer Science, 321(2):291–345, 2004.

[BDL04] G. Behrmann, A. David, and K. G. Larsen.
A tutorial on UPPAAL.
In Marco Bernardo and Flavio Corradini, editors, *Formal Methods for the Design of Real-Time Systems: 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM-RT 2004*, number 3185 in LNCS, pages 200–236. Springer–Verlag, September 2004.

[Bel61] R. Bellman.
Adaptive control processes: A guided tour.
Princeton University Press, New Jersey, 1961.

[Ben01] J. Bengtsson.
Efficient Symbolic State Exploration of Timed Systems: Theory and Implementation.
Licentiate thesis, 2001.

[BGP96] B. Bérard, P. Gastin, and A. Petit.
On the power of non-observable actions in timed automata.
In *Proceedings of STACS'96*, pages 257–268. Springer Verlag, 1996.

[BJK+06] S. Beyer, C. Jacobi, D. Kröning, D. Leinenbach, and W. Paul.
Putting it all together - Formal Verification of the VAMP.
In STTT Journal, Special Issue on Recent Advances in Hardware Verification, 2006.

[BK88] K.P. Brand and J. Kopainsky.
Principles and Engineering of Process Control with Petri Nets.
IEEE Transactions on Automatic Control, 33(2), 1988.

[BLP+99] G. Behrmann, K. G. Larsen, J. Pearson, C. Weise, and W. Yi.
Efficient timed reachability analysis using clock difference diagrams.
In *CAV '99: Proceedings of the 11th International Conference on Computer Aided Verification*, pages 341–353. Springer-Verlag, 1999.

[BM00] Olivier Bournez and Oded Maler.
On the representation of timed polyhedra.
In *Automata, Languages and Programming*, pages 793–807, 2000.

[BMPY97] M. Bozga, O. Maler, A. Pnueli, and S. Yovine.
Some Progress in the Symbolic Verification of Timed Automata.
In O. Grumberg, editor, *Proceedings of CAV'97 (Haifa, Israel)*, volume 1254 of *LNCS*, pages 179–190. Springer-Verlag, June 1997.

[BN03] D. Beyer and A. Noack.
Can decision diagrams overcome state space explosion in real-time verification?
In *Proc. FORTE*, LNCS 2767, pages 193–208. Springer, 2003.

[Bou03] P. Bouyer.

Untameable timed automata!
In Helmut Alt and Michel Habib, editors, *Proceedings of the 20th Annual Symposium on Theoretical Aspects of Computer Science (STACS'03)*, volume 2607 of *Lecture Notes in Computer Science*, pages 620–631, Berlin, Germany, February 2003. Springer.

[Bow96] F.D.J. Bowden.
Modelling time in Petri nets.
Proc. Second Australian-Japan Workshop on Stochastic Models, 1996.

[BP95] B. Bloom and R. Paige.
Transformational design and implementation of a new efficient solution to the ready simulation problem.
Science of Computer Programming, 24(3):189–220, 1995.

[BPDG98] B. Berard, A. Petit, V. Diekert, and P. Gastin.
Characterization of the expressive power of silent transitions in timed automata.
Fundamenta Informaticae, 36:145–182, 1998.

[bro02] *DER BROCKHAUS IN FÜNFZEHN BÄNDEN*.
F.A. Brockhaus GmbH, Leipzig, Mannheim, 2002.

[bro03] *DER BROCKHAUS Computer und Informationstechnologie*.
F.A. Brockhaus GmbH, Leipzig, Mannheim, 2003.

[Büc62] J. R. Büchi.
On a decision method in restricted second order arithmetic.
In *Proceedings of the International Congress on Logic, Methodology and Philosophy of Science*, pages 1–11, Stanford, CA, 1962. Stanford University Press.

[BY04] J. Bengtsson and W. Yi.
Timed automata: Semantics, algorithms and tools.
In W. Reisig and G. Rozenberg, editors, *In Lecture Notes on Concurrency and Petri Nets*, Lecture Notes in Computer Science vol 3098. Springer–Verlag, 2004.

[CDD98] S. Collart-Dutilleul and JP Denat.
P-Time Petri nets and the hoist scheduling problem.
In *Proc. IEEE Conf. Systems, Man, Cybernetics*, pages 558–563, 1998.

[Cer93] K. Cerans.
Decidability of bisimulation equivalences for parallel timer processes.
In *CAV '92: Proceedings of the Fourth International Workshop on Computer Aided Verification*, pages 302–315, London, UK, 1993. Springer-Verlag.

[CGP99] E.M. Clarke, O. Grumberg, and A. Peled.
Model Checking.
MIT Press, 1999.

[CO00] R. Cardell-Oliver.
Conformance Tests for Real-Time Systems with Timed Automata Specifications.
Formal Aspects of Computing, 12(5):350–371, 2000.

[DGP97] V. Diekert, P. Gastin, and A. Petit.
Removing epsilon-transitions in timed automata.
In *Proceedings of STACS'97*, pages 583–594. Springer Verlag, 1997.

[Die00] H. Dierks.
PLC-Automata: A New Class of Implementable Real-Time Automata.
Theoretical Computer Science, 253(1):61–93, 2000.

[Dil89] D. L. Dill.
Timing assumptions and verification of finite-state concurrent systems.
In *Automatic Verification Methods for Finite State Systems*, pages 197–212, 1989.

[DPP01] A. Dovier, C. Piazza, and A. Policriti.
A fast bisimulation algorithm.
Lecture Notes in Computer Science, 2102:79+, 2001.

[DT01] H. Dierks and J. Tapken.
Moby/PLC: Eine graphische Entwicklungsumgebung für SPS-Programme.
at-Automatisierungstechnik, 1:38–44, 2001.

[Fer89] J.-C. Fernandez.
An implementation of an efficient algorithm for bisimulation equivalence.
Science of Computer Programming, 13(1):219–236, 1989.

[Flo67] R. W. Floyd.
Assigning meanings to programs.
Proc. Symp. Appl. Math, 19:19–31, 1967.

[GDG+96] W. Goerigk, A. Dold, T. Gaul, G. Goos, A. Heberle, F.W. von Henke, U. Hoffmann, H. Langmaack, H. Pfeifer, H. Ruess, and W. Zimmermann.
Compiler Correctness and Implementation Verification: The Verix Approach.
In *Proceedings of the Poster Session of CC*, volume 96, pages 96–12, 1996.

[GGZ04] S. Glesner, G. Goos, and W. Zimmermann.
Verifix: Konstruktion und Architektur verifizierender Übersetzer(Verifix: Construction and Architecture of Verifying Compilers).
it- Information Technology, 46(5):265–276, 2004.

[GPC03] N. Giambiasi, J.L. Paillet, and F. Châne.
From timed automata to DEVS models.
Proceedings of the 35th conference on Winter simulation: driving innovation, pages 923–931, 2003.

[HHK95] M. R. Henzinger, T. A. Henzinger, and P. W. Kopke.
Computing simulations on finite and infinite graphs.
In *IEEE Symposium on Foundations of Computer Science*, pages 453–462, 1995.

[Hoa69] C. A. R. Hoare.
An axiomatic basis for computer programming.
Comm. Assoc. Comput. Mach., 12, 1969.

[HP92] J. Hannan and F. Pfenning.

Compiler verification in LF.
In *Logic in Computer Science, 1992. LICS'92., Proceedings of the Seventh Annual IEEE Symposium on*, pages 407–418, 1992.

[Int03] International Electrotechnical Commission.
IEC 61131-3 Ed. 2.0 English: Programmable controllers — Part 3: Programming languages.
International Electrotechnical Commission, Geneva, Switzerland, 2003.

[KBPOB99] B. Krieg-Brückner, J. Peleska, E.-R. Olderog, and A. Baer.
The UniForM Workbench, a Universal Development Environment for Formal Methods.
In J.M. Wing, J. Woodcock, and J. Davies, editors, *FM'99 – Formal Methods*, volume 1709 of *Lecture Notes in Computer Science*, pages 1186–1205. Springer, 1999.

[KLSV03] DK Kaynar, N. Lynch, R. Segala, and F. Vaandrager.
Timed I/O automata: a mathematical framework for modeling and analyzing real-time systems.
Real-Time Systems Symposium, 2003. RTSS 2003. 24th IEEE, pages 166–177, 2003.

[KLSV06] D.K. Kaynar, N. Lynch, R. Segala, and FW Vaandrager.
The Theory of Timed I/O Automata.
Morgan & Claypool Publishers, 2006.

[LLPY97] F. Larsson, K. G. Larsen, P. Pettersson, and W. Yi.
Efficient Verification of Real-Time Systems: Compact Data Structures and State-Space Reduction.
In *Proc. of the 18*th *IEEE Real-Time Systems Symposium*, pages 14–24. IEEE Computer Society Press, 1997.

[LT01] R. F. Lutje Spelberg and W. J. Toetenel.
Parametric real-time model checking using splitting trees.
Nordic Journal of Computing, 8(1):88–120, 2001.

[LY02] H. Lin and W. Yi.
Axiomatising timed automata.
Acta Informatica, 38(4):277–305, 2002.

[Mer74] P.M. Merlin.
A study of the recoverability of computing systems.
PhD thesis, University of California, Irvine, 1974.

[MLAH99] J. Möller, J. Lichtenberg, H. R. Andersen, and H. Hulgaard.
Difference decision diagrams.
In *Proceedings 13th International Workshop on Computer Science Logic*, volume 1683 of *Lecture Notes in Computer Science*, pages 111–125, Madrid, Spain, September 1999.

[MP99] S. Mukhopadhyay and A. Podelski.

Beyond region graphs: Symbolic forward analysis of timed automata.
In *Proceedings of the 19th Conference on Foundations of Software Technology and Theoretical Computer Science (FST&TCS-99)*, volume 1738 of *Lecture Notes in Computer Science*, pages 232–244, Chennai, India, December 1999. IARCS, Springer.

[MW99] A. Mader and H. Wupper.
Timed automaton models for simple programmable logic controllers.
In *Proceedings of Euromicro Conference on Real-Time Systems*, York, UK, June 1999.

[Nec00] G.C. Necula.
Translation validation for an optimizing compiler.
ACM SIGPLAN Notices, 35(5):83–94, 2000.

[Par81] D. Park.
Concurrency and automata on infinite sequences.
In *Proceedings of the 5th GI-Conference on Theoretical Computer Science*, pages 167–183, London, UK, 1981. Springer-Verlag.

[Pau05] W. Paul.
Towards a worldwide verification technology.
In *Proceedings of the Verified Software: Theories, Tools, Experiments Conference (VSTTE 2005)*, Zurich, Switzerland, October 2005.

[PSJ04] S. PalChaudhuri, A. Kumar Saha, and D. B. Johnson.
Adaptive clock synchronization in sensor networks.
In *IPSN '04: Proceedings of the third international symposium on Information processing in sensor networks*, pages 340–348, New York, NY, USA, 2004. ACM Press.

[PSS98] A. Pnueli, M. Siegel, and E. Singerman.
Translation validation.
Lecture Notes in Computer Science, 1384:151+, 1998.

[Rei82] Wolfgang Reisig.
Petrinetze, Eine Einführung.
Springer, 1982.

[S+00] R. Soley et al.
Model Driven Architecture.
OMG white paper, 2000.

[sch98] *Lexikon Informatik und Datenverarbeitung*.
Oldenbourg, 1998.

[Str99] Karsten Strehl.
Interval diagrams: Increasing efficiency of symbolic real-time verification.
In *Proceedings of the 6th International Conference on Real-Time Computing Systems and Applications (RTCSA '99)*, pages 488–491, Hong Kong, 13–15, 1999.

[Str02] M. Strecker.
Formal verification of a Java compiler in Isabelle.
volume 2392, pages 63–77. Springer, 2002.

[SWL90] B. Simons, J.L. Welch, and N. Lynch.
An overview of clock synchronization.
Fault-Tolerant Distributed Computing, pages 84–96, 1990.

[Tap98] J. Tapken.
Moby/PLC – A Design Tool for Hierarchical Real-Time Automata.
In E. Astesiano, editor, *Proceedings of FASE'98*, volume 1382 of *LNCS*, pages 326–329. Springer Verlag, 1998.

[TH02] J. Thieme and H.-M. Hanisch.
Model-Based Generation of Modular PLC Code Using IEC 61131 Function Blocks.
Proceedings of the IEEE International Symposium on Industrial Electronics, 1:199–204, 2002.

[Thi02] J. Thieme.
Symbolische Erreichbarkeitsanalyse und automatische Implementierung strukturierter, zeitbewerteter Steuerungsmodelle.
PhD thesis, Lehrstuhl fuer Automatisierungstechnik, Martin Luther Universitaet Halle-Saale, Berlin, 2002.

[Tri99] S. Tripakis.
Verifying progress in timed systems.
In *Proceedings of ARTS'99*, pages 299–314, 1999.

[VCD+85] R. Valette, M. Courvoisier, H. Demmou, JM Bigou, and C. Desclaux.
Putting Petri nets to work for controlling flexible manufacturing systems.
In *Proceedings of the IEEE International Symposium on Circuits and Systems, Kyoto, Japan*, pages 929–932, 1985.

[Wan00] Farn Wang.
Efficient data structure for fully symbolic verification of real-time software systems.
In *Tools and Algorithms for Construction and Analysis of Systems*, pages 157–171, 2000.

[Wan01] Farn Wang.
Symbolic verification of complex real-time systems with clock-restriction diagram.
In Myungchul Kim, Byoungmoon Chin, Sungwon Kang, and Danhyung Lee, editors, *FORTE*, volume 197 of *IFIP Conference Proceedings*, pages 235–250. Kluwer, 2001.

[Wan04] F. Wang.
Efficient verification of timed automata with BDD-like data structures.
International Journal on Software Tools for Technology Transfer (STTT),

6(1):77–97, 2004.

[WDMR04] M. De Wulf, L. Doyen, N. Markey, and J.-F. Raskin.
Robustness and implementability of timed automata.
In *FORMATS/FTRTFT*, volume 3253 of *Lecture Notes in Computer Science*, pages 118–133. Springer, 2004.

[WDR04] M. De Wulf, L. Doyen, and J.-F. Raskin.
Almost ASAP Semantics: From Timed Models to Timed Implementations.
In *HSCC*, volume 2993 of *Lecture Notes in Computer Science*, pages 296–310. Springer, 2004.

[Wil99] H. X. Willems.
Compact timed automata for PLC programs.
Technical Report CSI-R9925, Computing Science Institute, University of Nijmegen, 1999.

[WL97] C. Weise and D. Lenzkes.
Efficient scaling-invariant checking of timed bisimulation.
In *STACS*, Lecture Notes in Computer Science, pages 177–188, 1997.

[WR99] J.W. Webb and R.A. Reis.
Programmable Logic Controllers: Principles and Applications.
Prentice Hall, 1999.

[Yi90] W. Yi.
Real-time behaviour of asynchronous agents.
In *CONCUR '90: Proceedings on Theories of concurrency : unification and extension*, pages 502–520, New York, NY, USA, 1990. Springer-Verlag New York, Inc.

[Yov97] S. Yovine.
KRONOS: A verification tool for real-time systems.
Springer International Journal of Software Tools for Technology Transfer, 1(1-2), December 1997.

[Yov98] S. Yovine.
Model checking timed automata.
In *Lectures on Embedded Systems, European Educational Forum, School on Embedded Systems*, pages 114–152. Springer Verlag, 1998.

[ZPFG03] Lenore Zuck, Amir Pnueli, Yi Fang, and Benjamin Goldberg.
VOC: A methodology for translation validation of optimizing compilers.
Journal of Universal Computer Science, (9(3)):223–247, 2003.

Index

Erklärung

Hiermit erkläre ich, dass ich diese Arbeit selbständig und ohne fremde Hilfe verfasst habe. Ich habe keine anderen als die von mir angegebenen Quellen und Hilfsmittel benutzt. Die den benutzten Werken wörtlich oder inhaltlich entnommenen Stellen sind als solche kenntlich gemacht worden. Ich habe mich bisher nicht um den Doktorgrad beworben.

Dirk Pollmächer Bad Lauchstädt, 15. Februar 2008

Lebenslauf

Persönliche Daten

Name	Dirk Pollmächer
Adresse	E.-Thälmann-Str. 10 D-06246 Bad Lauchstädt
Telefon	0 34 61 47 90 49
E-Mail	dirk.pollmaecher@gmx.de
Geb. am	11. 08. 1977 in Markranstädt Verheiratet, ein Sohn, deutsch

Schulbildung

09.1984–09.1991	Grundschule in Halle/Saale
09.1991–06.1996	Georg-Cantor-Gymnasium Halle Gymnasium mit mathematisch-naturwissenschaftlichem Schwerpunkt Leistungskurse Physik und Biologie Abiturnote: Sehr Gut

Wehrdienst

11.1996–09.1997	4./Panzerbataillon 64 Wolfhagen

Studium an der Martin-Luther-Universität Halle-Wittenberg

10.1997–09.2003	Diplom Wirtschaftsinformatik Vertiefungsrichtungen: Allgemeine BWL, Computerintegrierte Systeme und Optimierung Gesamtnote: Gut
04.1998–02.2003	Diplom Informatik Vertiefungsrichtungen: Softwareentwicklung, Parallele Systeme und Datenbanken Gesamtnote: Sehr Gut

04.2003–08.2005	Gradiuertenstipendium zur Promotion am Lehrstuhl für Software-Engineering und Programmiersprachen Prof. Dr. Zimmermann Thema: „Konstruktion korrekter Codegeneratoren für Speicherprogrammierbare Steuerungen“
09.2005–08.2006	Drittmittelprojekt mit Codewrights GmbH Karlsruhe Reengineering und Neuentwicklung eines Compilers für Gerätebeschreibungssprachen im Bereich von Feldbussystemen unter C und Microsoft .Net

Beruflicher Werdegang

seit 09.2006	Mitarbeiter am Universitätsrechenzentrum Teamleiter für das Reengineering, die Weiterentwicklung und Pflege des universitätseigenen Web-Content-Management-Systems sowie Mitorganisation an der Umgestaltung des Universitätsauftritts

Dirk Pollmächer — Bad Lauchstädt, 15. Februar 2008