Carl v. Ossietzky
UNIVERSITÄT
OLDENBURG

Fachbereich Informatik

Diplomarbeit

Graphische Spezifikationssprachen:

Der Zusammenhang zwischen Constraint Diagrams und Real-Time Symbolic Timing Diagrams

Vorgelegt von:	Jochen Hoenicke
Erstprüfer:	Prof. Dr. E.-R. Olderog
Zweitprüfer:	Prof. Dr. W. Damm

Oldenburg, den 14. Juni 1999

Inhaltsverzeichnis

Kapitel 1

Einleitung

Für die Entwicklung fehlerfreier Systeme ist eine formale Spezifikation der gewünschten Funktionalität notwendig. Diese ist erforderlich, um die Korrektheit einer Implementierung des Systems zu beweisen. Eine formale Sprache besteht meistens aus mathematischen Formeln, was den Nachteil hat, dass sie nicht für jedermann intuitiv verständlich ist. Graphische Beschreibungen können dagegen, sofern die Semantik passend definiert wird, besser verstanden werden. Daher ist ein graphischer Formalismus wünschenswert, der eine formal definierte Semantik besitzt.

In dieser Arbeit sollen zwei graphische Spezifikationssprachen untersucht und miteinander verglichen werden. Die von Cheryl Dietz entwickelten Constraint Diagrams [Die96] [Die98] (im Folgenden CDs abgekürzt) sind relativ gut geeignet, intuitiv Anforderung aufzustellen. Bisher gibt es jedoch noch keine Möglichkeit sie automatisch zu verifizieren. Die am Institut OFFIS entwickelten Real-Time Symbolic Timing Diagrams (RTSTDs) [Fey96][FJ], eine Erweiterung der Symbolic Timing Diagrams in [SD93], können dagegen von Model-Checkern direkt verwendet werden, dafür sind einige Anforderungen nicht direkt spezifizierbar.

Beide oben erwähnten Formalismen dienen dazu, Anforderungen von *Realzeitsystemen* zu beschreiben. Ein Realzeitsystem ist ein reaktives System, das für gewisse Eingaben die zugehörigen Ausgaben innerhalb vorgegebener Zeitintervalle liefern muss. Solche Systeme sind in der Praxis weit verbreitet und finden sich zum Beispiel in der Maschinenüberwachungen oder in Flugzeug- und Bahnsteuerungen.

Wie in Kapitel 5 noch deutlich wird, ist die Semantik dieser beiden graphischen Spezifikationssprachen jedoch sehr unterschiedlich. Beide Diagrammtypen haben ihre Stärken und Schwächen. Daher kann es nützlich sein, die Fähigkeiten beider Diagramme miteinander zu verbinden.

Ziel der Arbeit ist es zu zeigen, wie eine Teilklasse von Anforderungen, die in CDs gegeben sind, in RTSTDs transformiert werden. Es wird ein formaler Beweis für die Korrektheit der Transformation gegeben und an einer Fallstudie die Praktikabilität der Transformation betrachtet.

1.1 Fallstudie Bahnübergang

Bei der Fallstudie, die in dieser Diplomarbeit behandelt wird, handelt es sich um eine Referenzfallstudie aus dem Schwerpunktprogramm „Integration von Techniken der Softwarespezifikation für ingenieurwissenschaftliche Anwendungen“ der Deutschen Forschungsgemeinschaft (DFG) [Jan97].

In dieser Fallstudie wird behandelt, wie von einem Zug aus per Funk sicher Schranken und Weichen (im Fachjargon *zustandsvariable Fahrwegelemente* genannt) fernbedient werden können. Sollte dabei eine Komponente versagen, muss der Zug rechtzeitig vor der Gefahrenstelle zum Stehen kommen.

Dazu berechnet die Zugsteuerung für jede Position die Maximalmalgeschwindigkeit, die der Zug haben darf, um noch rechtzeitig vor dem nächsten Gefahrenpunkt zum Stehen zu kommen. Diese Funktion, die zu jedem Ort die Geschwindigkeit angibt, wird *Bremskurve* genannt. Wenn der Zug die von der Bremskurve angegebene Maximalgeschwindigkeit überschreitet, wird er automatisch abgebremst. Nachdem der Zug bis zum Stillstand abgebremst wurde, bleibt immer noch eine Fahrsperre aktiv, solange der Gefahrenpunkt gesetzt ist. Sobald der Zug von der Schranke oder Weiche das Signal bekommt, dass der Fahrweg sicher ist, löscht er den Gefahrenpunkt und berechnet die Bremskurve neu. Geschieht dies rechtzeitig, so muss der Zug gar nicht erst abbremsen.

In regelmäßigen Abständen fährt der Zug über eine im Gleisbett verlegte *Balise* und liest deren Identifikation aus. Bei der Balise handelt es sich um ein passives Element, das vom Zug energetisch angeregt wird und darauf hin eine statische Kennung liefert. Anhand dieser Kennung kann der Zug über seinem elektronischen *Streckenatlas* ermitteln, an welchem Ort er sich befindet und welche Weichen und Schranken in naher Zukunft erreicht werden. Der Einfachheit halber wird in dieser Diplomarbeit nur der Fall einer einzigen Schranke betrachtet.

Nach Überfahren der Balise setzt er auf seinem Streckenatlas einen *Gefahrenpunkt* vor diese Schranke. Das Setzen des Gefahrenpunktes führt zu einer erneuten Berechnung der Bremskurve mit der Geschwindigkeit null im Gefahrenpunkt. Nun sendet der Zug rechtzeitig ein Signal an die betroffene Schranke, den sogenannten *Bahnübergangs-Stellbefehl*, damit diese sich schließt. Der genaue Zeitpunkt hängt von der Geschwindigkeit des Zuges und von der von der Schranke benötigten Zeit ab.

Sobald die Schranke geschlossen ist, sendet sie eine Statusmeldung an den Zug. Dieser löscht daraufhin den Gefahrenpunkt und kann im Idealfall mit seiner Maximalgeschwindigkeit weiterfahren. Wenn der Zug die Schranke passiert hat, wird dies durch einen Sensor erkannt, der hinter der Schranke angebracht ist. Dadurch wird die Schranke automatisch wieder geöffnet.

Unterbleibt jedoch die Statusmeldung, dass die Schranke geschlossen wurde, oder sendet die Schrankensteuerung ein Fehlersignal, so wird die Zugsteuerung bei Erreichen der Bremskurve den Zug abbremsen und rechtzeitig vor der Schranke zum Stillstand bringen. Der Zugführer bekommt dann auf seinem Display mitgeteilt, welche Art von Störung vorliegt und wie er sie eventuell selbst beheben kann. Dann kann er den ordnungsgemäßen Zustand quittieren und so den Gefahrenpunkt manuell löschen.

Wenn der Zug nicht innerhalb von fünf Minuten nach Absenden des Stellbefehls den Bahnübergang passiert, so muss er diesen als unsicher ansehen, und den Gefahrenpunkt erneut setzen.

1.2 Constraint Diagrams

Constraint Diagrams (CD) [Die96] wurden von Cheryl Dietz an der Universität Oldenburg im Rahmen ihrer Dissertation entwickelt. Die Semantik der Diagramme wird über den Duration Calculus erklärt (siehe [ZHR91]) und spezifiziert das Verhalten sogenannter Observablen. Bei den *Observablen* handelt es sich um Ein- und Ausgabegrößen, die diskrete Werte annehmen können und über die (kontinuierliche) Zeit variieren.

1.2.1 Beispiel

Das Diagramm in Abbildung 1.1 spezifiziert das Verhalten eines sogenannten *Watchdogs* (Wachhund). Ein Watchdog überprüft, ob ein gewisses Signal S in regelmäßigen Abständen kommt und schlägt im Fehlerfall Alarm (A). In Zügen muss der Fahrer beispielsweise regelmäßig einen Knopf betätigen um anzuzeigen, dass er wach ist, andernfalls wird der Zug automatisch zum Stillstand gebracht.

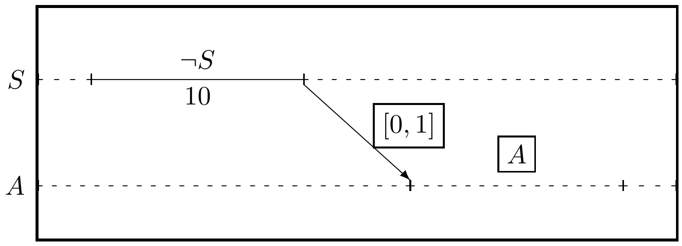


Abbildung 1.1: Spezifikation eines Watchdog

Die beiden Zeilen des Diagramms stellen den Verlauf der beiden Observablen S und A über die Zeit dar. Gestrichelte Bereiche deuten an, dass die Observable in diesem Intervall beliebige Werte annehmen kann.

Die Länge der Intervalle ist nicht maßstabsgetreu; die tatsächliche zeitliche Länge der Intervalle wird durch zusätzliche Angaben spezifiziert. Im obigen Fall wird zum Beispiel durch die 10 festgelegt, dass das Intervall $\neg S$ genau zehn Sekunden dauern soll. Auch über die zeitliche Beziehung zwischen den beiden Observablen wird zunächst keine Aussage gemacht. Lediglich durch zusätzliche Pfeile wird diese Beziehung hergestellt.

Das Diagramm besteht aus zwei Teilen, einer Voraussetzung und einer Zusicherung. Der Zusicherungsteil wird durch die Kästchen markiert. Die Voraussetzung besteht im obigen Fall im Grunde nur aus der ersten Zeile. Dort wird verlangt, dass für genau zehn Sekunden das Signal S nicht gesetzt ist. Da über die Bereiche vor und hinter diesem Intervall keine Aussage gemacht wird, kann das Signal aber durchaus länger als zehn Sekunden nicht gesetzt sein.

Ist diese Voraussetzung erfüllt, verspricht die Zusicherung daß innerhalb von einer Sekunde Alarm ausgelöst wird: Es gibt ein Intervall gibt, auf dem Alarm gesetzt ist und dieses Intervall beginnt spätestens eine Sekunde nach Beendigung des in der Voraussetzung genannten Intervalls, aber nicht früher.

1.2.2 Aktionsdiagramme

Aktionsdiagramme [Die97] sind eine Unterklasse von Constraint Diagrams, die sehr gut geeignet sind, das Verhalten von Automaten zu beschreiben.[1]

Es gibt vier Arten von Aktionsdiagrammen: Initialisierungs-, Sequenz-, Synchronisations- und Stabilitätsdiagramme. Im Folgenden seien X eine Observable, $\pi, \pi_1, \ldots, \pi_n$ Zustandsausdrücke[2] über die Observable X und φ die Konjunktion von Zustandsausdrücken $\bigwedge \varphi_i$, wobei φ_i ein Zustandsausdruck über eine Observable $\neq X$ ist.

Das einfachste Aktionsdiagramm ist das *Initialisierungsdiagramm* in Abbildung 1.2. Es besagt, dass die Observable X initial den Wert π annimmt.

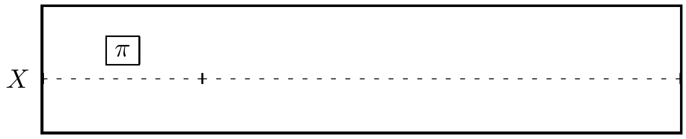


Abbildung 1.2: Aktionsdiagramme: Initialisierung

Das *Sequenzdiagramm* ist in Abbildung 1.3 zu sehen. Es legt fest, in welcher Reihenfolge die Zustände einer Observablen aufeinander folgen können. In diesem Fall darf auf dem Zustand π nur einer der Zustände $\pi_1, \ldots, \pi_n$ folgen. Sequenzdiagramme eignen sich dafür, bei der Beschreibung von endlichen Automaten die Zustandsmenge zu spezifizieren, die der Automat von einem gewissen Zustand als Nächstes erreichen kann.

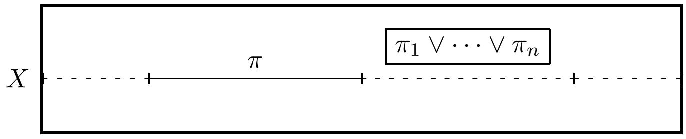


Abbildung 1.3: Aktionsdiagramme: Sequenz

Eine weitere Art von Diagrammen sind *Stabilitätsdiagramme*, siehe Abbildung 1.4. Diese Diagramme legen fest, dass ein Zustand unter gewissen Voraussetzungen stabil ist, bzw. nur eine gewisse Teilmenge von Folgezuständen zuläßt.

Die Wenn-Dann-Aussage des Stabilitätsdiagramm lautet: Wenn der Zustand X von einem anderen Zustand $(\neg\pi)$ in den Zustand π wechselt und während der ganzen Zeit, die π gilt, auch eine Nebenbedingung φ gilt und die Phase π noch weniger als t Sekunden aktiv ist, so kann X entweder den Wert π weiter beibehalten oder zu einem der angegebenen Werte $\pi_1 \vee \cdots \vee \pi_n$ wechseln. In diesem Diagramm werden mit den mit 0 beschrifteten Pfeilen die Gleichzeitigkeit der π und φ-Phasen ausgedrückt: Die Phase φ muss genau 0 Sekunden nach der π-Phase beginnen und auch genau 0 Sekunden nach der π-Phase enden.

Einige Spezialfälle des Diagramms sind noch erwähnenswert: Ist $n = 0$, so entfallen die alternativen Zustände $\pi_1 \vee \cdots \vee \pi_n$ und π muss stabil bleiben.

[1] Der mit dem Duration Calculus vertraute Leser wird eine Ähnlichkeit mit den Implementables feststellen, siehe Abschnitt 3.1.6. In der Tat sind die Aktionsdiagramme so gewählt, dass sie mit den Implementables semantisch äquivalent sind (siehe [Die97]).

[2] Die genaue Definition von Zustandsausdrücken wird in Abschnitt 3.1.2 behandelt.

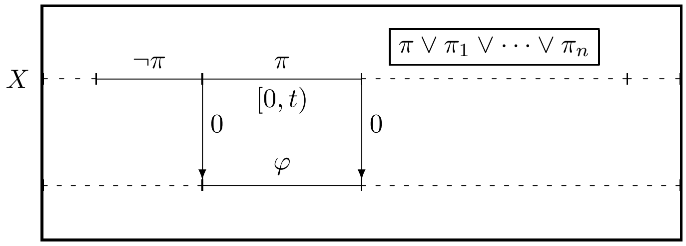


Abbildung 1.4: Aktionsdiagramme: Stabilität

Ist $t = \infty$, so entfällt die Zeitbeschränkung, und ist $\varphi = \mathbf{true}$, so entfällt die Nebenbedingung.

Weil das Diagramm fordert, dass es zunächst ein Zeitintervall gibt, auf dem $\neg\pi$ gilt, greift das Stabilitätsdiagramm nicht in der initialen Phase. Daher gibt es noch eine Abwandlung, die *initiale Stabilität*, die in Abbildung 1.5 zu sehen ist. Dieses Diagramm fordert, dass π *von Anfang* an zusammen mit der Nebenbedingung φ gilt. Dieses Diagramm wird immer benötigt, wenn π der Initialzustand von X sein kann, und tritt meist zusammen mit dem Stabilitätsdiagramm auf.

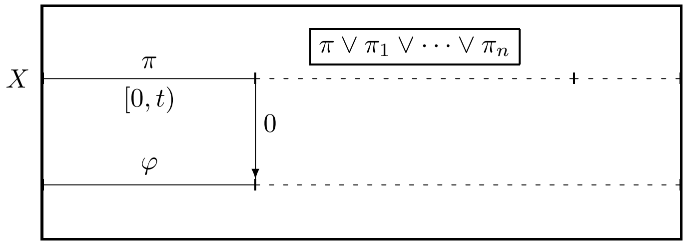


Abbildung 1.5: Aktionsdiagramme: initiale Stabilität

Das letzte Aktionsdiagramm ist das *Synchronisationsdiagramm* (siehe Abbildung 1.6). Dieses Diagramm fordert, dass wenn X den Wert π hat und eine bestimmte Bedingung φ eintritt, innerhalb einer gewissen Reaktionszeit t die Observable X ihren Wert ändert.

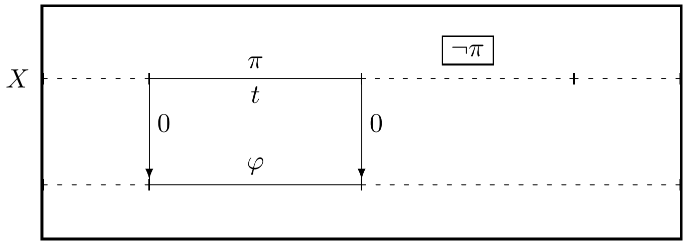


Abbildung 1.6: Aktionsdiagramme: Synchronisation

1.3 Real-Time Symbolic Timing Diagrams

Ein anderer Diagramm-Typ zur Beschreibung von Realzeitsystemen sind *Real-Time Symbolic Timing Diagrams* (RTSTD) [Fey96]. Auch hier wird das zeitliche Verhalten von Ein- und Ausgabegrößen und deren Abhängigkeiten spezifiziert.

1.3.1 Beispiel

Als Beispiel betrachten wir die Spezifikation eines einfaches „Handshaking"-Protokolls zwischen Sender und Empfänger. Dieses Protokoll dient dazu zwei unabhängige Prozesse (Sender und Empfänger) zu synchronisieren. Dazu setzt der Sender ein Signal *Req* (Request) um anzuzeigen, dass er etwas senden will. Der Empfänger antwortet daraufhin mit *Ack* (Acknowledge), um anzuzeigen, dass er bereit ist Daten entgegenzunehmen.

Eine mögliche Signalfolge ist in Abbildung 1.7 definiert. Wann immer die Signale *Req* (Request) und *Ack* (Acknowledge) nicht gesetzt sind, setzt der Sender innerhalb von zehn Sekunden das Signal *Req* und wartet auf eine Antwort *Ack* vom Empfänger. Wenn dieses Signal innerhalb der im Diagramm angegebenen Zeitspanne (zwei bis vier Sekunden) gesetzt wird, nimmt der Sender das Signal *Req* innerhalb von fünf Sekunden, aber frühestens nach einer Sekunde wieder zurück und wartet erneut auf die Bestätigung des Empfängers.

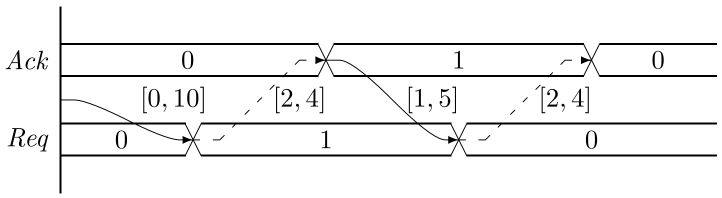


Abbildung 1.7: Spezifikation eine Senders als RTSTD

Wie bei den CDs werden die Zustandsverläufe der Ein- und Ausgabegrößen durch die Zeilen dargestellt und die Abhängigkeiten zwischen zwei verschiedenen Größen durch Pfeile spezifiziert. Diese Pfeile werden *Constraints* genannt, da sie das erlaubte Verhalten einschränken. Es gibt zwei unterschiedliche Typen von Constraints, nämlich strong und weak Constraints. *Strong Constraints* werden mit durchgezogenen Pfeilen eingezeichnet und legen fest, wie sich die Ausgabegrößen verhalten müssen. *Weak Constraints*, die mit gestrichelten Pfeilen dargestellt werden, beschreiben dagegen das Verhalten der Eingabegrößen, das von der Umwelt verlangt wird. Wird ein weak Constraint verletzt, schränkt das Diagramm das weitere Verhalten der Ein- und Ausgabegrößen nicht weiter ein.

In dem in Abbildung 1.7 gezeigten Diagramm wird das Verhalten des Empfängers (Setzen und Zurücksetzen von *Ack*) durch weak Constraints beschrieben, das Verhalten des Senders dagegen durch strong Constraints. Daher beschreibt das Diagramm nur das Verhalten des Senders unter der Annahme, dass der Empfänger wie beschrieben antwortet.

1.3.2 Informelle Semantik

Ein Real-Time Symbolic Timing Diagram besteht aus sogenannten *Waveforms*, die die erlaubten Zustandsübergänge beschreiben. Eine Waveform (siehe Abbildung 1.8) wird aktiviert, sobald die *Aktivierungsbedingung* des ersten Ereignisses ($actc(e_1)$) gültig ist. Dadurch wird auch dieses Ereignis aktiviert. Solange es aktiviert ist, muss die Aktivierungsbedingung weiterhin gelten. Wenn die Aktivierungsbedingung des nächsten Ereignisses $actc(e_2)$ gilt, kann das Ereignis e_1 abgearbeitet werden und Ereignis e_2 wird aktiviert. Die Bedingung $actc(e_2)$ wird daher auch *Fortsetzungsbedingung* von e_1 ($contc(e_1)$) genannt. Nun ist e_2 aktiviert und $actc(e_2)$ muss gelten, bis auch dieses Ereignis abgearbeitet wird. Das wiederholt sich, bis das letzte Ereignis $e_\top$ (nicht im Diagramm eingezeichnet) erreicht ist. Sobald dieses letzte Ereignis aktiviert wurde, ist der weitere Verlauf der Zustandsvariablen beliebig.

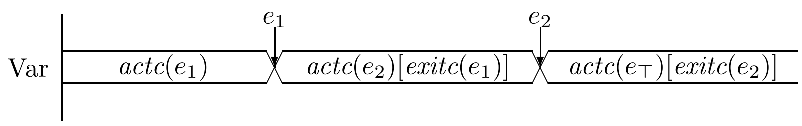


Abbildung 1.8: Allgemeiner Aufbau einer Waveform

Eine andere Möglichkeit, das Ereignis e_1 zu verlassen, ist über die *Ausstiegsbedingung* $exitc(e_1)$. In diesem Fall wird das gesamte Diagramm akzeptiert, d. h. über den weiteren Verlauf der Variablen werden keine Aussagen gemacht.

Besteht ein Diagramm aus mehreren Waveforms, können die einzelnen Ereignisse unabhängig voneinander abgearbeitet werden. Zu einem Zeitpunkt dürfen auch mehrere Ereignisse verschiedener Waveforms durchgeführt werden. Das gesamte Diagramm wird immer dann aktiviert, wenn die Aktivierungsbedingungen der ersten Ereignisse aller Variablen gleichzeitig erfüllt sind.

Durch strong Constraints (die bereits erwähnten durchgezogenen Pfeile) kann die zeitliche Abfolge von Ereignissen eingeschränkt werden. So legt in Abbildung 1.7 auf Seite 9 der mit $[1, 5]$ beschriftete strong Constraint fest, dass das zweite Ereignis von *Req* frühestens eine Sekunde und spätestens fünf Sekunden nach dem ersten Ereignis von *Ack* durchgeführt werden muss. Weak Constraints wirken dagegen ähnlich wie Ausstiegsbedingungen: Sobald ein weak Constraint verletzt wird (und kein strong Constraint gleichzeitig verletzt wird), wird das Diagramm akzeptiert und keine weiteren Aussagen über den Verlauf der Zustandsvariablen gemacht.

1.3.3 Initiale Diagramme

Die bisher betrachteten RTSTDs greifen zu jedem Zeitpunkt, zu dem die Aktivierungsbedingungen der ersten Ereignisse aller Waveforms erfüllt sind. Will man Aussagen über den initialen Zustand machen, benötigt man ein *initiales Diagramm*.

In Abbildung 1.9 ist ein initiales Diagramm dargestellt. Es ist durch einen doppelten Balken am linken Rand gekennzeichnet. Dieses Diagramm spezifiziert, dass am Anfang das Signal *Req* auf null liegt und dort auch für mindestens fünf Sekunden verweilt. Im Gegensatz zu dem vorherigen Diagrammtyp wird hier *gefordert*, dass die Aktivierungsbedingung $Req = 0$ des ersten Ereignisses

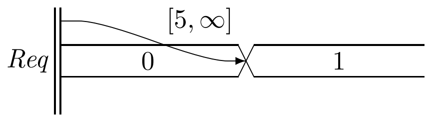


Abbildung 1.9: Beispiel eines initialen RTSTD

am Anfang gilt. Die weitere Abarbeitung der Ereignisse des Diagramms ist aber genauso, wie oben beschrieben.

1.4 Vergleich der beiden Diagrammtypen

Bei beiden Diagrammmtypen werden die Zustandsverläufe durch die Zeilen des Diagramms beschrieben, die Abhängigkeiten verschiedener Ein- und Ausgabegrößen durch Pfeile.

Damit hören die Ähnlichkeiten aber schon auf. Während die CDs sauber zwischen Voraussetzung und Zusicherung trennen ist dies bei RTSTD nicht der Fall. Es wird zwar zwischen weak und strong Constraints unterschieden, was in etwa Voraussetzung und Zusicherung entspricht, wird aber ein weak Constraint verletzt, müssen dennoch alle Zusicherungen erfüllt sein, die zeitlich vor der Verletzung liegen. Außerdem gibt es überhaupt keine Unterscheidung bei den Phasen. Die erste Phase ist dort implizit eine Voraussetzung (außer bei initialen Diagrammen), die anderen Phasen dagegen Zusicherungen.

Bei den CDs lässt man im Allgemeinen einige Phasen unspezifiziert. Bei RTSTDs ist es zwar ebenfalls möglich Phasen mit **true** zu beschriften, aber es ist davon abzuraten. Man sollte dort dafür sorgen, dass die Aktivierungsbedingungen aufeinanderfolgender Ereignisse sich gegenseitig ausschließen, denn sonst kann das Diagramm eine unerwünschte Semantik haben. Auf diesen Punkt wird in Abschnitt 5.4 näher eingegangen.

Durch die klare Unterscheidung von Voraussetzung und Zusicherung eignen sich CDs sehr gut dafür, Eigenschaften eines Systems in einer Wenn-Dann-Aussage zu spezifizieren. Die Stärken von RTSTD liegen dagegen in der Beschreibung von Kommunikationsprotokollen, wie zum Beispiel der Senderspezifikation im vorherigen Kapitel. Eine äquivalente Spezifikation durch CDs müsste aus mehreren Diagrammen bestehen und die Klarheit ginge verloren.

Kapitel 2

Constraint Diagrams für die Fallstudie

Wie im letzten Abschnitt der vorherigen Kapitels erwähnt eignen sich Constraint Diagrams besser zur Spezifikation von einfachen Wenn-Dann-Aussagen. Daher wird die Zugsteuerung der Fallstudie aus Abschnitt 1.1 in diesem Kapitel durch CDs spezifiziert.

Wir werden dabei von der Bremskurve abstrahieren, weil dafür kontinuierliche Größen wie Geschwindigkeit und Position benötigt werden. Diese können weder in Constraint Diagrams noch in Real-Time Symbolic Timing Diagrams ausgedrückt werden. Stattdessen würde man ein sogenanntes Hybrides System [MP93] benötigen.

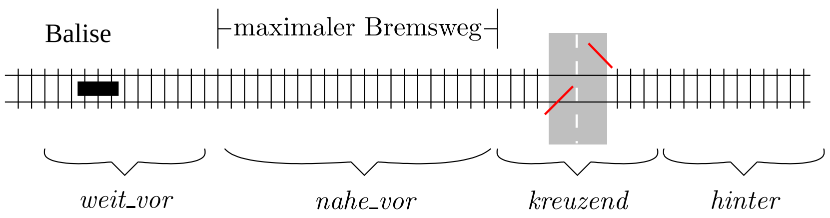


Abbildung 2.1: Schematische Darstellung des Bahnübergangs

Daher teilen wir die Position in vier Bereiche (siehe Abbildung 2.1). Wenn der Zug *weit_vor* dem Bahnübergang ist, kann er in jedem Fall noch rechtzeitig vor dem Bahnübergang zum Stehen kommen. Nur wenn der Zug beim Eintreten in den Bereich *nahe_vor* anfängt zu bremsen, bleibt er in diesem Bereich auch stehen. Der Bereich *kreuzend* bezeichnet, dass der Zug gerade den Bahnübergang überquert. Anschließend tritt er in den Bereich *hinter* ein.

Die Ein- und Ausgabegrößen der vereinfachten Fallstudie zusammen mit ihrer informellen Bedeutung sind in der Abbildung 2.2 dargestellt. Die ersten vier Observablen *Position*, *Balise*, *Statusmeldung* und *Manuell* sind dabei Eingabegrößen, die letzten drei Observablen *Timeout*, *Stellbefehl* und *GP* sind Ausgabegrößen.

$Position : \mathbf{Time} \rightarrow \{weit\_vor, nahe\_vor, kreuzend, hinter\}$	relative Position des Zuges zum Bahnübergang
$Balise : \mathbf{Time} \rightarrow \mathbb{B}$	Wird eine *Balise* erkannt
$Statusmeldung : \mathbf{Time} \rightarrow \{keine, OK, Fehler\}$	Bahnübergang gesichert
$Manuell : \mathbf{Time} \rightarrow \mathbb{B}$	Manuelle Bestätigung
$Timeout : \mathbf{Time} \rightarrow \mathbb{B}$	Bahnübergang nicht innerhalb 5 Minuten passiert
$Stellbefehl : \mathbf{Time} \rightarrow \mathbb{B}$	Bahnübergang sichern
$GP : \mathbf{Time} \rightarrow \mathbb{B}$	Gefahrenpunkt gesetzt

Abbildung 2.2: Observablen der Fallstudie

2.1 Initialisierung

Abbildung 2.3 zeigt die Initialisierung der Observablen. Es wird davon ausgegangen, dass der Zug genügend weit entfernt vom Bahnübergang startet (initial gilt $Position = weit\_vor$), die Balise initial nicht gelesen wird und der Gefahrenpunkt sowie das Timeout nicht gesetzt sind. Es ist übrigens egal, ob die Initialisierungen wie hier in einem Constraint Diagram zusammengefasst werden oder in vier einzelnen Diagrammen dargestellt werden.

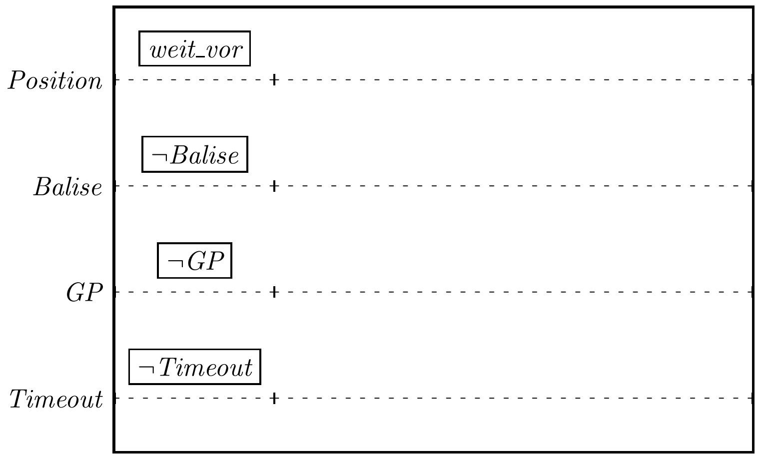


Abbildung 2.3: Initialer Zustand der relevanten Observablen

2.2 Änderung der Position

Zunächst wollen wir festlegen, wie sich die Position ändern kann. Die Intuition legt nahe, dass sich die Position nur von *weit_vor* über *nahe_vor* und *kreuzend* nach *hinter* ändern kann. Damit der Zug mehr als nur einen Bahnübergang überqueren kann, erlauben wir außerdem, dass sich die Position von *hinter* nach *weit_vor* (dies bezieht sich dann auf den nächsten Bahnübergang) ändern kann. Dabei darf keiner dieser Bereiche übersprungen werden.

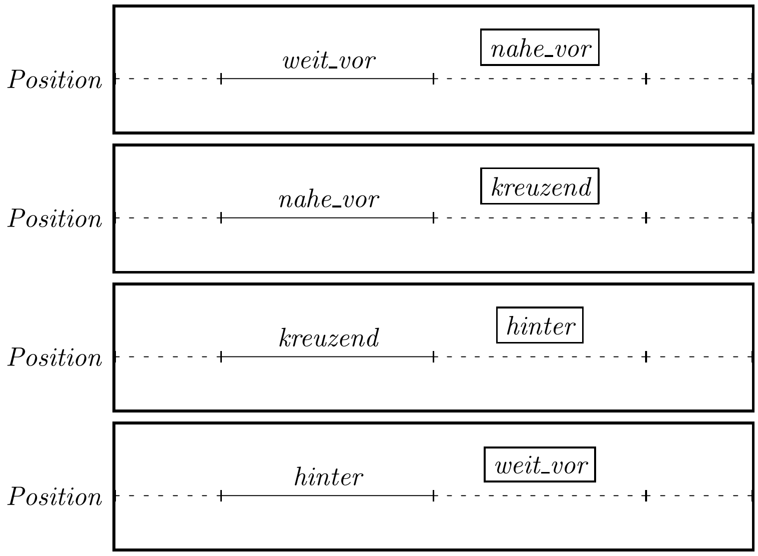


Abbildung 2.4: Mögliche Übergänge der Position

Dies ist ein Anwendungsfall für Sequenzdiagramme (siehe Abbildung 2.4). So besagt das erste Diagramm, dass wenn der Zug an der Position *weit_vor* ist, als Nächstes nur die Position *nahe_vor* folgen darf.

2.3 Gefahrenpunkt

Als Nächstes wollen wir den Gefahrenpunkt näher betrachten. Der Gefahrenpunkt soll nach Lesen einer Balise gesetzt werden und solange gesetzt bleiben, bis die Schranke meldet, dass sie geschlossen ist, oder der Zugführer manuell den ordnungsgemäßen Zustand bestätigt hat. Außerdem soll nach einem Timeout der Gefahrenpunkt erneut gesetzt werden. Ist der Gefahrenpunkt gesetzt darf der Zug nicht die Schranke überqueren.

In Abbildung 2.5 sind die beiden Fälle aufgeführt, wann der Gefahrenpunkt gesetzt werden muss. Wir erlauben der Zugsteuerung eine gewisse Reaktionszeit von ε Sekunden um auf die Eingabesignale zu reagieren. Dafür benutzen wir ein Synchronisationsdiagramm: Wenn der Gefahrenpunkt nicht gesetzt ist ($\neg GP$) und die *Balise* erkannt wird, so wird nach spätestens ε Sekunden der Gefahrenpunkt gesetzt (GP). Beim Eintreten eines Timeouts (falls der Bahnübergang nicht innerhalb von fünf Minuten passiert wurde) wird ebenfalls der Gefahrenpunkt gesetzt. Das zugehörige Diagramm ist analog.

Wenn man ein System formal spezifiziert, muss man nicht nur ausdrücken, was passieren soll, sondern auch festlegen was nicht passieren darf. In diesem Fall wollen wir, dass der Gefahrenpunkt nicht grundlos gesetzt wird. Dies ist eine Stabilitätseigenschaft: Wenn der Gefahrenpunkt nicht gesetzt ist ($\neg GP$) und keine der obigen Bedingungen (*Balise* und *Timeout*) eingetreten ist, bleibt der Gefahrenpunkt auch weiterhin nicht gesetzt. Wie bereits in Abschnitt 1.2.2

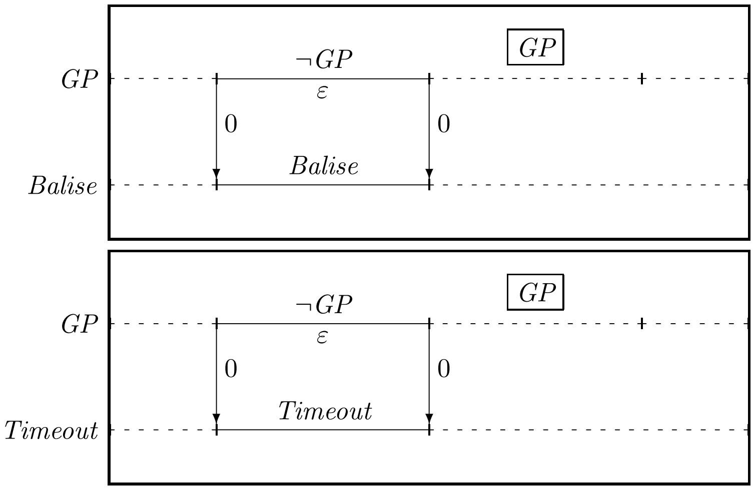


Abbildung 2.5: Setzen des Gefahrenpunktes

erklärt benötigen wir zusätzlich noch die initiale Stabilität. Diese beiden Diagramme sind in Abbildung 2.6 dargestellt.

Nachdem wir festgelegt haben, wann der Gefahrenpunkt gesetzt wird, wollen wir nun ausdrücken, dass der Zug rechtzeitig vor dem Gefahrenpunkt zum stehen kommt. Wie bereits erwähnt, betrachten wir nur den Gefahrenpunkt vor der Schranke. Wir wollen also sagen, dass der Zug nicht den Bahnübergang kreuzen kann, wenn der Gefahrenpunkt gesetzt ist.

Dies ist allerdings nicht immer möglich: Falls der Gefahrenpunkt erst gesetzt wird, wenn sich der Zug unmittelbar vor dem Bahnübergang befindet, gibt es keine Möglichkeit mehr den Zug rechtzeitig abzubremsen. Deshalb haben wir am Anfang dieses Abschnitts die Position in vier Bereiche unterteilt. Wenn also der Gefahrenpunkt bereits zu dem Zeitpunkt gesetzt ist, an dem die Position von *weit_vor* nach *nahe_vor* wechselt, so wird der Zug vor dem Gefahrenpunkt zum stehen kommen, das heißt, er wird den Bereich *nahe_vor* nicht verlassen (und damit den Bahnübergang nicht kreuzen) bevor der Gefahrenpunkt gelöscht wird (siehe Abbildung 2.7).

Diese Eigenschaft wird uns von der Bremssteuerung garantiert. Die Bremssteuerung selbst werden wir nicht weiter analysieren, da, wie bereits erwähnt, dafür Aussagen über kontinuierliche Größen wie Geschwindigkeit und genaue Position nötig sind, die hier nicht behandelt werden können. Wir nehmen sie also quasi als Axiom an.

In Abbildung 2.8 wird spezifiziert, wann der Gefahrenpunkt gelöscht werden muss, bzw. nicht gelöscht werden darf. Der Gefahrenpunkt wird gelöscht, wenn die Schranke erfolgreiches Schließen meldet, aber nur, wenn noch kein Timeout eingetreten ist. Außerdem kann der Gefahrenpunkt jederzeit durch den Zugführer manuell gelöscht werden. Wie vorhin wird dies durch Synchronisationsdiagramme ausgedrückt, und wir erlauben auch hier wieder eine gewisse Reaktionszeit ε.

Außerdem benötigen wir ein Stabilitätsdiagramm, damit der Gefahrenpunkt nicht spontan gelöscht werden kann. Die initiale Stabilität kann aber weggelassen werden, da aufgrund der Initialisierung schon klar ist, dass der Gefahren-

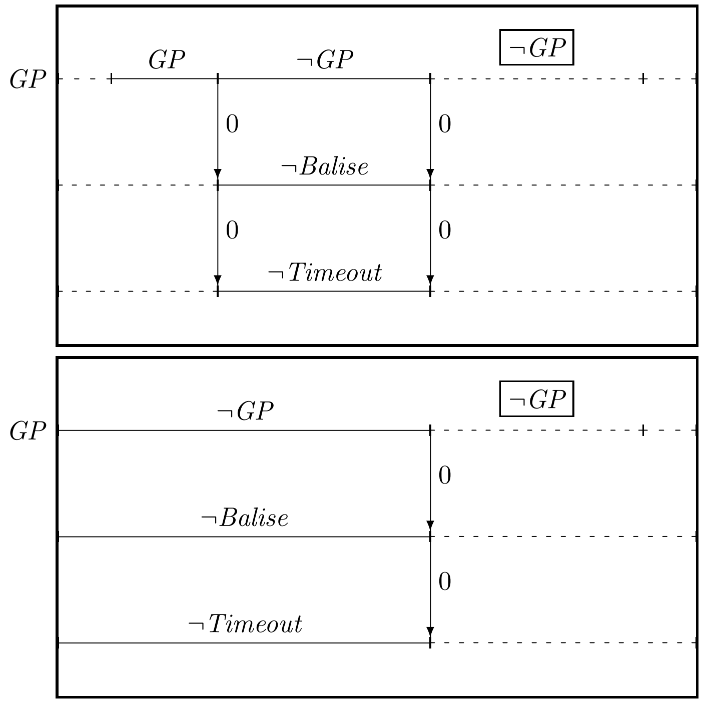


Abbildung 2.6: Stabilität von $\neg GP$

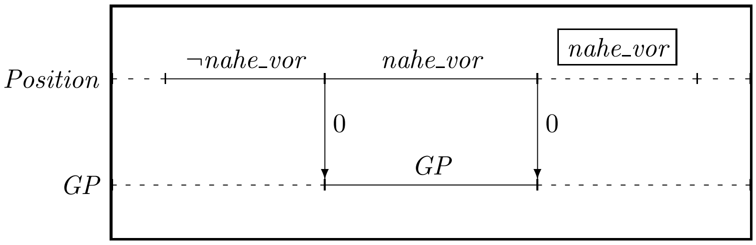


Abbildung 2.7: Funktion der Bremssteuerung

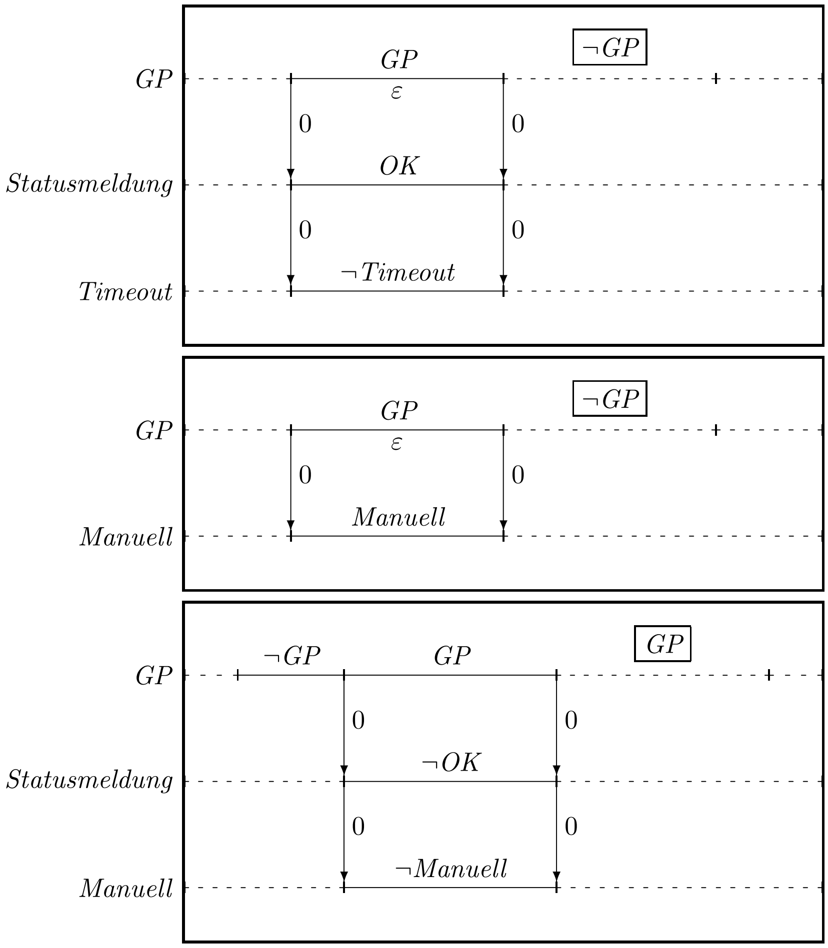


Abbildung 2.8: Löschen des Gefahrenpunktes

punkt zum Zeitpunkt null nicht gesetzt ist.

2.4 Stellbefehl

Nun wenden wir uns der nächsten Ausgabevariablen *Stellbefehl* zu. Der Stellbefehl soll nur dann gesendet werden, wenn der Zug die Balise gelesen hat. Dies kann man auch am gesetzten Gefahrenpunkt erkennen. Der genaue Zeitpunkt zu dem der Stellbefehl abgeschickt wird, wird hier unspezifiziert gelassen; der optimale Zeitpunkt hängt von vielen Faktoren, wie der Geschwindigkeit des Zuges, der Art des Bahnübergangs (Halbschranken oder Vollschranken) oder die Entfernung der Balise vom Bahnübergang, die hier überhaupt nicht betrachtet werden sollen. Nachdem der Stellbefehl gesendet ist, bleibt er so lange aktiv, bis der Bahnübergang passiert wurde.

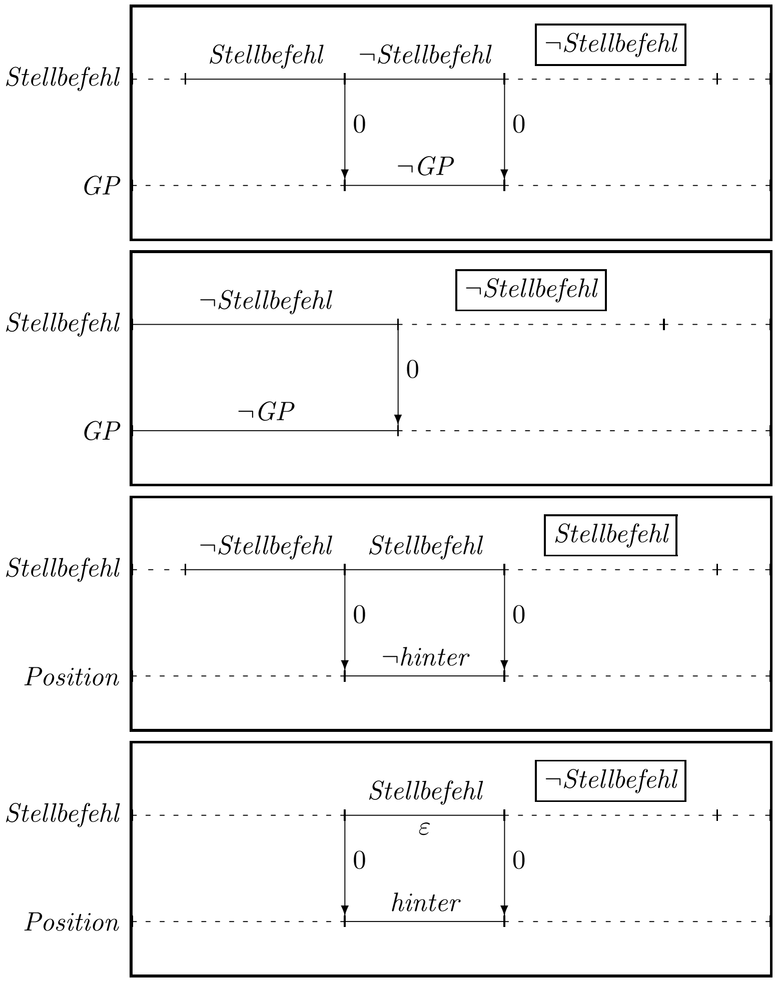


Abbildung 2.9: Der Stellbefehl

Die Diagramme in Abbildung 2.9 spezifizieren die eben genannten Bedingungen. Das erste Diagramm ist ein Stabilitätsdiagramm, das dafür sorgt, dass ein

Stellbefehl nicht gesendet wird, solange der Gefahrenpunkt noch nicht gesetzt ist ($\neg GP$). Wieder benötigen wir die initiale Stabilität, die im zweiten Diagramm dargestellt ist. Das dritte Diagramm sorgt dafür, dass ein Stellbefehl solange stabil anliegt, wie der Bahnübergang noch nicht passiert wurde, d.h. $Position \neq hinter$. Sobald der Bahnübergang passiert wurde $Position = hinter$ sorgt das letzte Diagramm dafür, dass innerhalb einer Reaktionszeit von ε Sekunden der Stellbefehl wieder zurückgenommen wird.

Die eben aufgeführte Spezifikation erlaubt auch, dass der *Stellbefehl* nie gesendet wird. Dies ist jedoch kein sicherheitskritisches Verhalten, denn der Gefahrenpunkt ist gesetzt, sodass der Zug rechtzeitig abbremsen wird, weil er keine Statusmeldung von der Schranke erhält.

2.5 Timeout

Die letzte Ausgabevariable ist *Timeout*. Dieses Signal soll immer dann gesetzt werden, wenn der Bahnübergang innerhalb von fünf Minuten nach Senden des *Stellbefehls* nicht passiert wurde. Das zugehörige Constraint Diagram ist in Abbildung 2.10 dargestellt. Es fordert, dass der *Stellbefehl* gesetzt ist und gleichzeitig der Bahnübergang noch nicht passiert wurde ($\neg hinter$). Wenn dann für 300 Sekunden (fünf Minuten) die Position $\neg hinter$ bleibt, wird innerhalb von ε Sekunden *Timeout* gesetzt.

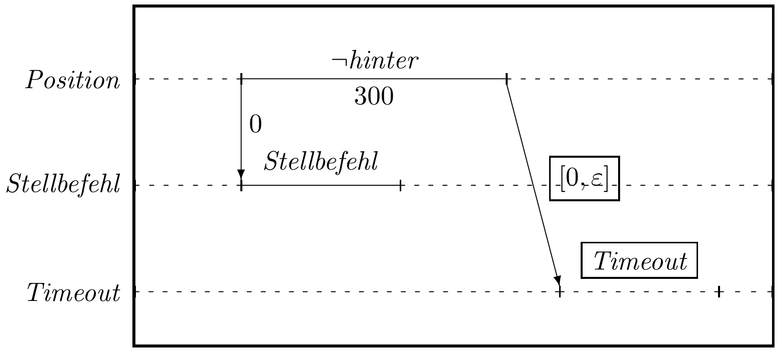


Abbildung 2.10: Setzen des Timeouts

2.6 Annahmen über die Umgebung

Im nächsten Kapitel wollen wir beweisen, dass solange der Zug keine Nachricht erhält, dass der Bahnübergang gesichert ist, er den Bahnübergang nicht überquert. Dafür benötigen wir allerdings noch einige weitere Informationen über die Eingabevariablen. Zum Beispiel reagieren wir nur auf das Signal der Balise, wenn es für mindestens ε Sekunden anliegt (siehe Abbildung 2.5). Deshalb müssen wir fordern, dass dieses Signal für diese Zeit stabil bleibt. Ähnliches gilt auch für andere Eingabegrößen, wie *Manuell*, *Statusmeldung* und *Position*. Das Stabilitätsdiagramm der Balise ist in Abbildung 2.11 zu sehen.

Trotzdem haben wir immer noch keine Gewissheit, dass die Balise überhaupt gelesen wird. Wenn aber keine Balise vorhanden ist, wird der Gefahrenpunkt

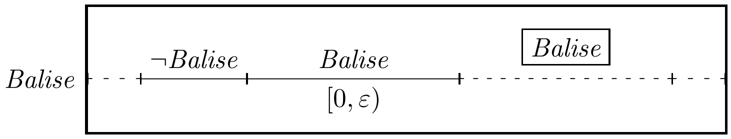


Abbildung 2.11: Das Signal *Balise* ist für ε Sekunden stabil.

nie gesetzt. Der Zug „weiß“ dann gar nicht, dass er sich einem Bahnübergang nähert. Um das auszuschließen, fordern wir, dass die Balise gelesen wird, bevor wir uns dem Bahnübergang nähern. Anders formuliert (siehe Abbildung 2.12): Die Position des Zuges bleibt solange *weit_vor*, wie die Balise noch nicht gelesen wurde. Dies ist ein Stabilitätsdiagramm der Position.

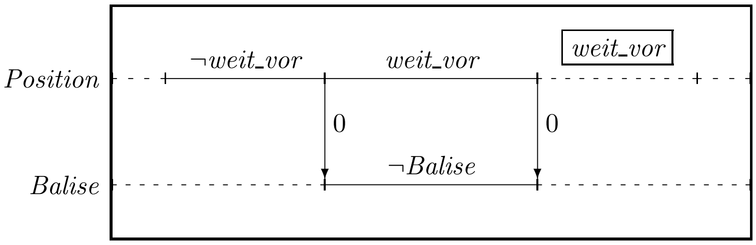


Abbildung 2.12: Balise ist korrekt verlegt (erste Version)

Nun haben wir aber immer noch keine Gewissheit, dass der Zug rechtzeitig vor der Schranke zum Stehen kommt. Wenn das Signal *Balise* anliegt, wird nämlich der Gefahrenpunkt nicht sofort gesetzt, sondern erst nach einer gewissen Reaktionszeit von ε Sekunden. Der Gefahrenpunkt muss aber gesetzt sein, bevor der Zug in den Bereich *nahe_vor* eintritt, damit er noch rechtzeitig zum Stillstand kommt. Daher ändern wir das letzte Diagramm noch ein wenig ab. Außerdem benötigen wir noch die initiale Stabilität. Das ergibt dann die beiden Diagramme in Abbildung 2.13. Die Voraussetzung wird in diesen Diagrammen dahingehend abgeschwächt, dass nach dem Signal $\neg$*Balise* noch maximal ε Sekunden verstreichen dürfen.

Im nächsten Kapitel werden wir die Semantik von Constraint Diagrams weiter beleuchten. Dies wird uns die Möglichkeit geben, zu beweisen, dass der Zug den Bahnübergang nur überqueren kann, wenn zuvor eine *Statusmeldung* mit dem Inhalt *OK* von der Schranke gesendet wurde.

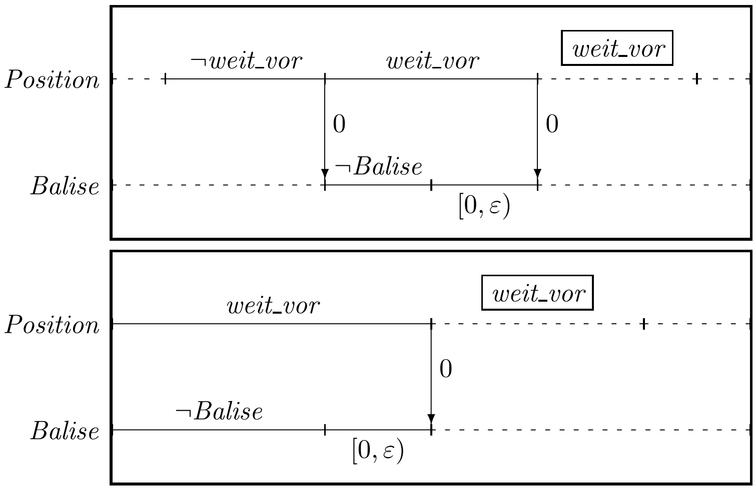


Abbildung 2.13: Balise ist korrekt verlegt (endgültige Version)

Kapitel 3

Semantik von Constraint Diagrams

Erst die formale Semantik der graphischen Formalismen macht die Bedeutung eines Diagramms in Zweifelsfällen eindeutig und ist für die Anforderungsdefinition, die einen Vertrag zwischen Auftraggeber und Software-Entwickler darstellt, unabdingbar.

Die Semantik eines Constraint Diagrams wird durch eine Duration Calculus Formel definiert. Daher werden wir zunächst einmal den Duration Calculus näher vorstellen.

3.1 Duration Calculus

Der *Duration Calculus* (DC) [ZHR91], [HZ97] ist eine auf der Prädikatenlogik basierende Zeitintervall-Logik. Die Zeit wird im DC durch die positiven reellen Zahlen dargestellt, d. h. $\mathbf{Time} = \mathbb{R}_{\geq 0}$, Zeitintervalle entsprechend durch Intervalle $[b, e]$ mit $b, e \in \mathbb{R}$.

DC-Formeln sind aus Prädikaten über Termen zusammengesetzt, Terme wiederum aus Konstanten, Variablen und Integralen über Zustandsausdrücke und Zustandsausdrücke schließlich aus Observablen. Diese Teile einer Formel werden wir auf den nächsten Seiten genauer betrachten.

Um den Wahrheitswert einer Duration Calculus-Formel zu bestimmen, benötigt man eine *Interpretation* $\mathcal{I}$, für die Konstanten und Observablen, eine *Belegung* $\mathcal{V}$, die den globalen Variablen einen Wert zuweist, und ein *Zeitintervall* $[b, e]$ mit $b, e \in \mathbf{Time}$.

3.1.1 Variablen und Konstanten

In Formeln des Duration Calculus tauchen folgende Arten von Variablen und Konstanten auf:

- *Observablen* X, auch *Zustandsvariablen* genannt, bezeichnen Größen, die sich über die Zeit ändern. Sie werden meist mit Großbuchstaben, oder mit großgeschriebenen Bezeichnern gekennzeichnet. Eine Observable hat einen endlichen Datentyp D_X und darf in einem endlichen Zeitintervall nur endlich oft den Wert wechseln. Die letzte Eigenschaft wird auch mit *endlicher Variabilität* bezeichnet. Die Semantik von Observablen $\mathcal{I}(X)$ ist eine Funktion vom Typ: $\textbf{Time} \to D_X$

- *Globale Variablen* x haben die gleiche Funktion wie in der Prädikatenlogik. Sie ändern sich nicht über die Zeit, man kann aber über sie mit den Quantoren $\forall$ und $\exists$ quantifizieren. Die Semantik von globalen Variablen wird durch die Belegung $\mathcal{V}$ durch $\mathcal{V}(x) \in \mathbb{R}$ gegeben.
- *Konstanten* c, haben in einer gegebenen Interpretation $\mathcal{I}$ nur einen konstanten Wert $\mathcal{I}(c) \in \mathbb{R}$. Beispiele sind $0, 1$, oder auch die Reaktionszeit ε aus der Fallstudie.
- *Konstante Funktionssymbole* f haben eine Interpretation $\mathcal{I}(f)$ vom Typ $\mathbb{R}^n \to \mathbb{R}$. Sie werden häufig in Infix-Notation geschrieben. Beispiele für konstante Funktionssymbole sind $+$ oder $-$.
- *Prädikatssymbole* p vom Typ $\mathbb{R}^n \to \{\textbf{true}, \textbf{false}\}$. Wie konstante Funktionen haben sie in einer gegebenen Interpretation immer den gleichen Wert und werden häufig in Infix-Notation geschrieben. Beispiele sind $=$ oder $\leq$.

In den weiteren Abschnitten wird die Interpretation $\mathcal{I}$ auf Zustandsausdrücke, Terme und Formeln erweitert.

3.1.2 Zustandsausdrücke

Zustandsausdrücke im Duration Calculus bezeichnen einen Wahrheitswert, der sich über die Zeit ändern. Ihr Typ ist $\textbf{Time} \to \{0, 1\}$ wobei 0 als falsch und 1 als wahr interpretiert werden kann. Ein Zustandsausdruck P hat die folgende Syntax.

$$P := 0 \mid 1 \mid X = d \mid P_1 \wedge P_2 \mid \neg P$$

Die Semantik $\mathcal{I}(P)(t)$ für einen gegebenen Zeitpunkt $t \in \textbf{Time}$ lässt sich induktiv über den Aufbau von P wie folgt berechnen:

$$\mathcal{I}(0)(t) = 0 \text{ und } \mathcal{I}(1)(t) = 1 \tag{3.1}$$

$$\mathcal{I}(X = d)(t) = \begin{cases} 1 & \text{falls } \mathcal{I}(X)(t) = d \\ 0 & \text{sonst} \end{cases} \tag{3.2}$$

$$\mathcal{I}(P_1 \wedge P_2)(t) = \mathcal{I}(P_1)(t) \cdot \mathcal{I}(P_2)(t) \tag{3.3}$$

$$\mathcal{I}(\neg P)(t) = 1 - \mathcal{I}(P)(t) \tag{3.4}$$

Zu (3.3) und (3.4) sollte man sich klar machen, dass die Formeln gerade der logischen UND- beziehungsweise NICHT-Verknüpfung entsprechen, wenn man 0 als falsch und 1 als wahr interpretiert.

Beispiel 3.1.1. Wir nehmen die Observable *Position* aus der Fallstudie mit dem Datentyp $\{weit\_vor, nahe\_vor, kreuzend, hinter\}$. Eine mögliche Interpretation $\mathcal{I}(Position)$ dieser Observablen ist in Abbildung 3.1 gezeigt. Sei

$$P := \neg(Postion = hinter)$$

dann ist $\mathcal{I}(P)$ eine Funktion, die auf dem Intervall $[0, 4]$ konstant 1, auf dem Intervall $(4, 7]$ den Wert 0 hat, usw.

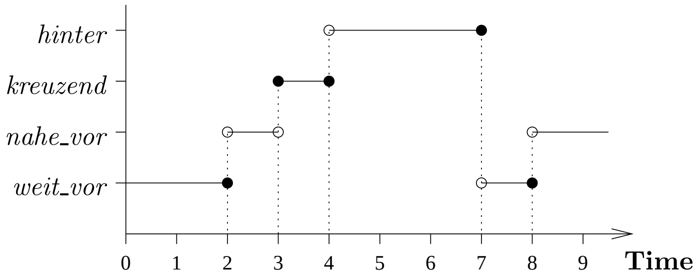


Abbildung 3.1: Möglicher Verlauf der Observablen *Position*

3.1.3 Terme

Terme stehen für reelle Zahlen und haben folgende Syntax:

$$T := c \mid f(T_1, \dots, T_n) \mid x \mid \ell \mid \textstyle\int P$$

Dabei steht ℓ für die Länge eines Zeitintervalls und $\int P$ für die Dauer, die P auf einem gegebenem Intervall den Wert 1 (wahr) hat.

Funktionssymbole f stehen häufig in Infix-Notation, z. B. 3+4, statt $+(3,4)$. Zur Eindeutigkeit müssen dann teilweise Klammern gesetzt werden. Dabei gelten die üblichen Klammerregeln, zum Beispiel Punkt- vor Strichrechnung.

Der reelle Wert eines Terms hängt nicht nur von der Interpretation $\mathcal{I}$ ab, sondern auch von der Variablenbelegung $\mathcal{V}$ und dem gerade betrachteten Intervall $[b,e] \subseteq$ **Time**. Man kann deshalb die Interpretation $\mathcal{I}(T)$ eines Terms als Funktion von $\mathit{Val} \times \mathit{Intv} \to \mathbb{R}$ ansehen, wobei *Val* die Menge aller Belegungen und *Intv* die Menge aller abgeschlossenen Intervalle $[b,e]$ mit $b, e \in \mathbb{R}$ ist.

Zu einer Belegung $\mathcal{V}$ und einem Intervall $[b,e] \subseteq$ **Time** lässt sich dann die Interpretation eines Terms wie folgt berechnen:

$$\mathcal{I}(c)(\mathcal{V},[b,e]) = \mathcal{I}(c) \tag{3.5}$$

$$\mathcal{I}(f(T_1,\dots,T_n))(\mathcal{V},[b,e]) = \mathcal{I}(f)(\mathcal{I}(T_1)(\mathcal{V},[b,e]),\dots,\mathcal{I}(T_n)(\mathcal{V},[b,e])) \tag{3.6}$$

$$\mathcal{I}(x)(\mathcal{V},[b,e]) = \mathcal{V}(x) \tag{3.7}$$

$$\mathcal{I}(\ell)(\mathcal{V},[b,e]) = e - b \tag{3.8}$$

$$\mathcal{I}(\textstyle\int P)(\mathcal{V},[b,e]) = \displaystyle\int_b^e \mathcal{I}(P)(t)\,dt \tag{3.9}$$

Das Integral in Formel (3.9) steht für ein Riemann-Integral. Aufgrund der endlichen Variabilität von Observablen folgt auch die endliche Variabilität von Zustandsausdrücken und deshalb hat das Riemann-Integral immer einen definierten Wert.

Beispiel 3.1.2. Seien *Position* und $\mathcal{I}(\mathit{Position})$ wie im letzten Beispiel. Seien $+$, $-$ und $\cdot$ konstante Funktionssymbole, deren Interpretation die Addition, Subtraktion bzw. Multiplikation in den reellen Zahlen ist und sei 5 eine Konstante mit $\mathcal{I}(5) = 5 \in \mathbb{R}$. Sei $\mathcal{V}$ eine Belegung mit $\mathcal{V}(x) = 6$, dann gilt:

$$
\begin{aligned}
&\mathcal{I}((x+\ell) - 5 \cdot \textstyle\int \mathit{Position} = \mathit{hinter})(\mathcal{V}, [0,9]) \\
&= \mathcal{I}(-)(\mathcal{I}(x+\ell)(\mathcal{V}, [0,9]), \mathcal{I}(5 \cdot \textstyle\int \mathit{Position} = \mathit{hinter})(\mathcal{V}, [0,9])) \\
&= \mathcal{I}(x)(\mathcal{V}, [0,9]) + \mathcal{I}(\ell)(\mathcal{V}, [0,9]) - \mathcal{I}(5 \cdot \textstyle\int \mathit{Position} = \mathit{hinter})(\mathcal{V}, [0,9])) \\
&= \mathcal{V}(x) + (9-0) - \mathcal{I}(5)(\mathcal{V}, [0,9]) \cdot \mathcal{I}(\textstyle\int \mathit{Position} = \mathit{hinter})(\mathcal{V}, [0,9]) \\
&= 6 + 9 - \mathcal{I}(5) \cdot \int_0^9 \mathcal{I}(\mathit{Position} = \mathit{hinter})(t)\, dt = 6 + 9 - 5 \cdot 3 = 0
\end{aligned}
$$

3.1.4 Formeln

Nun können wir uns den DC-Formeln zuwenden. Sie haben folgende Syntax.

$$F := \mathbf{true} \mid \mathbf{false} \mid p(T_1, \dots, T_n) \mid F_1 \wedge F_2 \mid \neg F \mid \forall x \cdot F \mid F_1 ; F_2$$

Bis auf den letzten Operator $F_1 ; F_2$, der *Chop* genannt wird, entspricht die obige Syntax der Syntax von Formeln in der Prädikatenlogik. Auch die Semantik ist identisch, sieht man einmal davon ab, dass die Semantik von Termen und damit auch von Formeln von dem gegebenen Zeitintervall abhängt. Der Chop-Operator erlaubt es nun das Zeitintervall in zwei Unterintervalle aufzuteilen. Die Formel $F_1 ; F_2$ ist wahr auf einem Intervall, genau dann wenn es eine Unterteilung dieses Intervalls in zwei Intervalle gibt, sodass F_1 auf dem linken Intervall und F_2 auf dem rechten Intervall gilt.

Um die Semantik einer Formel zu beschreiben, benötigen wir wie in der Prädikatenlogik die Substitution einer Belegung $\mathcal{V}$. Zu einer Variablen x und einem Datenwert $d \in \mathbb{R}$ sei $\mathcal{V}[d/x]$ die Belegung mit dem Wert $\mathcal{V}[d/x](x) = d$ und $\mathcal{V}[d/x)(y) = \mathcal{V}(y)$ für alle Variablen $y \neq x$.

Die Interpretation einer Formel lässt sich wie bei Termen als Funktion über Belegungen und Intervallen darstellen, für Formeln liefert die Funktion allerdings einen Wahrheitswert in $\{\mathbf{true}, \mathbf{false}\}$. Zu einer gegebenen Belegung $\mathcal{V}$ und einem Intervall $[b, e]$ gilt:

$$\mathcal{I}(\mathbf{true})(\mathcal{V}, [b,e]) = \mathbf{true} \text{ und } \mathcal{I}(\mathbf{false})(\mathcal{V}, [b,e]) = \mathbf{false} \tag{3.10}$$

$$\mathcal{I}(f(T_1, \dots, T_n)(\mathcal{V}, [b,e]) = f(\mathcal{I}(T_1)(\mathcal{V}, [b,e]), \dots, \mathcal{I}(T_n)(\mathcal{V}, [b,e])) \tag{3.11}$$

$$\mathcal{I}(F_1 \wedge F_2)(\mathcal{V}, [b,e]) = \begin{cases} \mathbf{true} & \text{falls } \mathcal{I}(F_1)(\mathcal{V}, [b,e]) = \mathbf{true} = \mathcal{I}(F_2)(\mathcal{V}, [b,e]) \\ \mathbf{false} & \text{sonst} \end{cases} \tag{3.12}$$

$$\mathcal{I}(\neg F)(\mathcal{V}, [b,e]) = \begin{cases} \mathbf{true} & \text{falls } \mathcal{I}(F)(\mathcal{V}, [b,e]) = \mathbf{false} \\ \mathbf{false} & \text{sonst} \end{cases} \tag{3.13}$$

$$\mathcal{I}(\forall x \cdot F)(\mathcal{V}, [b,e]) = \begin{cases} \mathbf{true} & \begin{array}{l}\text{falls für alle } d \in \mathbb{R} \\ \quad \mathcal{I}(F)(\mathcal{V}[d/x], [b,e]) = \mathbf{true}\end{array} \\ \mathbf{false} & \text{sonst} \end{cases} \tag{3.14}$$

$$\mathcal{I}(F_1 ; F_2)(\mathcal{V}, [b,e]) = \begin{cases} \mathbf{true} & \begin{array}{l}\text{falls es ein } m \in [b,e] \text{ gibt mit} \\ \quad \mathcal{I}(F_1)(\mathcal{V}, [b,m]) = \mathbf{true} = \mathcal{I}(F_2)(\mathcal{V}, [m,e])\end{array} \\ \mathbf{false} & \text{sonst} \end{cases} \tag{3.15}$$

Beispiel 3.1.3. Sei wieder $\mathcal{I}, \mathcal{V}$ wie im letzten Beispiel. Seien $=, \leq$ Prädikatssymbole mit ihrer kanonischen Interpretation. Es gilt:

$$\begin{aligned}\mathcal{I}(\ell = x)(\mathcal{V}, [0,9]) &= \mathcal{I}(=)(\mathcal{I}(\ell)(\mathcal{V}, [0,9]), \mathcal{I}(x)(\mathcal{V}, [0,9])) \\ &= \mathcal{I}(=)(9-0, \mathcal{V}(x)) = \mathcal{I}(=)(9,6) = \mathbf{false}\end{aligned}$$

$$\begin{aligned}\mathcal{I}(\ell = x)(\mathcal{V}, [0,6]) &= \mathcal{I}(=)(\mathcal{I}(\ell)(\mathcal{V}, [0,6]), \mathcal{I}(x)(\mathcal{V}, [0,9])) \\ &= \mathcal{I}(=)(6-0, \mathcal{V}(x)) = \mathcal{I}(=)(6,6) = \mathbf{true}\end{aligned}$$

$$\begin{aligned}&\mathcal{I}(\textstyle\int(\mathit{Position} = \mathit{hinter}) \leq \ell)(\mathcal{V}, [6,9]) \\ &\quad = \mathcal{I}(\leq)(\mathcal{I}(\textstyle\int(\mathit{Position} = \mathit{hinter}))(\mathcal{V}, [6,9]), \mathcal{I}(\ell)(\mathcal{V}, [6,9])) \\ &\quad = \mathcal{I}(\leq)(1,3) = \mathbf{true}\end{aligned}$$

Aus den beiden letzten Formeln folgt mit Chop-Punkt $m = 6$:

$$\mathcal{I}(\ell = x; \textstyle\int(\mathit{Position} = \mathit{hinter}) \leq \ell) = \mathbf{true}$$

□

Wenn eine Formel F für eine gegebene Interpretation $\mathcal{I}$, Variablenbelegung $\mathcal{V}$ und Zeitintervall $[b, e]$ den Wert **true** liefert, sagt man auch die Formel F *gilt* in $\mathcal{I}, \mathcal{V}, [b, e]$, in Zeichen $\mathcal{I}, \mathcal{V}, [b, e] \models F$. Im obigen Beispiel gilt also:

$$\mathcal{I}, \mathcal{V}, [0,9] \models \ell = x; \textstyle\int(\mathit{Position} = \mathit{hinter}) \leq \ell$$

Eine Interpretation $\mathcal{I}$ *erfüllt* eine Formel F, in Zeichen $\mathcal{I} \models F$, wenn für alle Belegungen $\mathcal{V}$ und alle Zeitintervalle $[b, e] \subseteq \mathbf{Time}$ gilt $\mathcal{I}, \mathcal{V}, [b, e] \models F$. Beispielsweise gilt:

$$\mathcal{I} \models \textstyle\int(\mathit{Position} = \mathit{hinter}) \leq \ell$$

Eine Interpretation $\mathcal{I}$ *erfüllt* eine Formel F *von 0 an*, in Zeichen $\mathcal{I} \models_0 F$, wenn für alle Belegungen $\mathcal{V}$ und alle Punkte $e \in \mathbf{Time}$ gilt $\mathcal{I}, \mathcal{V}, [0, e] \models F$, d. h. umgangssprachlich F gilt auf jedem Anfangsstück.

3.1.5 Abkürzungen

Nachdem nun Syntax und Semantik von DC-Formeln eingeführt wurde, sind noch einige Abkürzungen zu definieren. Im Folgenden stehen F, G für Formeln und P, Q für Zustandsausdrücke. Mit dem Symbol $\hat{=}$ soll ausgedrückt werden, dass der linke Teil durch den rechten Teil definiert wird.

- Sei D_X der (endliche) Typ einer Observablen X. Ist $D_X = \{0, 1\}$ kann man statt $X = 1$, bzw. $X = 0$ auch kurz X bzw. $\neg X$ schreiben. Ist der Datentyp von X disjunkt mit allen anderen Datentypen, kann statt $X = d$ auch kurz d und statt $X \neq d$ auch kurz $\neg d$ geschrieben werden.

- Logisches oder ($\vee$) kann durch $\neg$ und $\wedge$ ausgedrückt werden. Es kann sowohl in Formeln als auch in Zustandsausdrücken vorkommen:

  $$F \vee G \mathrel{\hat{=}} \neg(\neg F \wedge \neg G) \text{ und } P \vee Q \mathrel{\hat{=}} \neg(\neg P \wedge \neg Q)$$

  Das gleiche gilt für die Implikation $\Rightarrow$ und Äquivalenz $\Leftrightarrow$.

- Der Existenzquantor $\exists$ kann mithilfe des Allquantors $\forall$ ausgedrückt werden:
$$\exists x \cdot F \mathrel{\hat{=}} \neg \forall x \cdot \neg F$$
- Mit der Formel $\lceil\rceil$ wird ausgedrückt, dass das Intervall ein *Punktintervall* ist, d. h. ein Intervall der Länge 0:
$$\lceil\rceil \mathrel{\hat{=}} \ell = 0$$
- Mit der Formel $\lceil P \rceil$ wird ausgedrückt, dass ein Zustandsausdruck P den Wert 1 fast überall in einem gegebenen Intervall annimmt:
$$\lceil P \rceil \mathrel{\hat{=}} \textstyle\int P = \ell \wedge \ell > 0$$
- Die Formel $\Diamond F$ bedeutet, dass es ein Teilintervall gibt, auf dem die Formel F gilt. Mithilfe des Chop-Operators, kann $\Diamond F$ wie folgt definiert werden:
$$\Diamond F \mathrel{\hat{=}} \mathbf{true}; F; \mathbf{true}$$
- Die Formel $\Box F$ gilt, wenn F für alle Teilintervalle gilt. Es besteht eine ähnliche Beziehung zu $\Diamond F$, wie zwischen den Quantoren $\exists$ und $\forall$:
$$\Box F \mathrel{\hat{=}} \neg \Diamond \neg F$$

3.1.6 Standardformen und Implementables

Die sogenannten *Standardformen* dienen zur einfachen Spezifikation von Kontrollautomaten. Sei F eine Formel, P ein Zustandsausdruck und t ein Term, in dem kein $\int$ und kein ℓ vorkommt.

- followed-by: $F \longrightarrow \lceil P \rceil \mathrel{\hat{=}} \Box \neg (F; \lceil \neg P \rceil)$
- leads-to: $F \xrightarrow{t} \lceil P \rceil \mathrel{\hat{=}} (F \wedge \ell = t) \longrightarrow \lceil P \rceil$
- up-to: $F \xrightarrow{\leq t} \lceil P \rceil \mathrel{\hat{=}} (F \wedge \ell \leq t) \longrightarrow \lceil P \rceil$

Die *Implementables* von DC entsprechen den in Abschnitt 1.2.2 vorgestellten Aktionsdiagrammen.[1] Sei X_i eine Observable, seien $\pi, \pi_1, \ldots, \pi_n$ Zustandsausdrücke, die nur von der Observablen X_i abhängen und φ ein Zustandsausdruck, der nicht von X_i abhängt. Sei t wie oben. Es gibt folgende Implementables:

- Initialisierung: $\ell = 0 \vee (\lceil \pi \rceil; \mathbf{true})$
- Sequenz: $\lceil \pi \rceil \longrightarrow \lceil \pi \vee \pi_1 \vee \cdots \vee \pi_n \rceil$
- Stabilität: $\lceil \neg \pi \rceil; \lceil \pi \wedge \varphi \rceil \xrightarrow{\leq t} \lceil \pi \vee \pi_1 \vee \cdots \vee \pi_n \rceil$
- initiale Stabilität: $\neg(\lceil \pi \wedge \varphi \rceil \wedge \ell \leq t; \lceil \neg(\pi \vee \pi_1 \vee \cdots \vee \pi_n) \rceil; \mathbf{true})$
- Synchronisation: $\lceil \pi \wedge \varphi \rceil \xrightarrow{t} \lceil \neg \pi \rceil$

In [Die97] wurde gezeigt, dass die Semantik der Aktionsdiagramme mit der Semantik der DC-Implementables (die von 0 an gelten sollen) übereinstimmt.

[1] Genauer gesagt ist es umgekehrt. Die Aktionsdiagramme wurden so gewählt, dass sie den DC Implementables entsprechen.

3.2 Constraint Diagrams

Nun können wir auf der Basis des eben eingeführten Duration Calculus die Semantik von Constraint Diagrams festlegen.

3.2.1 Syntax

Die allgemeine Syntax eines Constraint Diagrams *cd* ist in Abbildung 3.2 dargestellt. Auf der linken Seite sind die Observablen aufgelistet, in diesem Fall X_i und X_j. Die Abschnitte der einzelnen Zeilen werden Phasen genannt. Ausdrücke $\pi_l^i, \tilde{\pi}_l^i$ stehen für Zustandsausdrücke über die Observable X_i. Dabei bezeichnen π_l^i, π_l^j die Annahmen über X_i bzw. X_j und die eingerahmten Ausdrücke $\tilde{\pi}_l^i, \tilde{\pi}_l^j$ die Zusicherungen. Ist einer dieser Ausdrücke $\pi_l^i \equiv 1$ (also immer wahr), so kann dieser weggelassen werden. Im Falle eines eingerahmten Ausdruckes $\tilde{\pi}_l^i$ wird dann auch der Rahmen weggelassen.

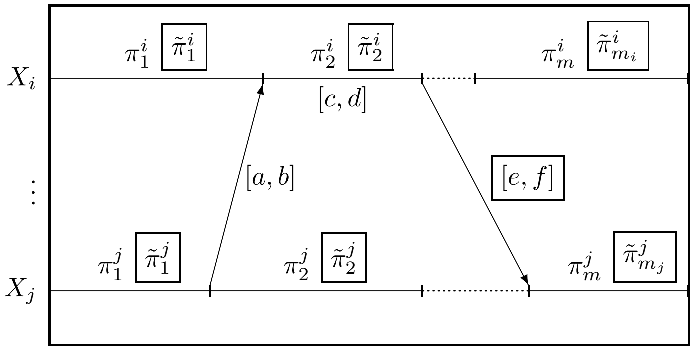


Abbildung 3.2: Syntax von Constraint Diagrams

Mit $[a, b], [c, d], [e, f]$ werden Intervalle aus $\mathbf{Time} \cup \{\infty\}$ bezeichnet. Ist $a = b$, kann statt $[a, b]$ auch kurz a geschrieben werden. Wird eine Phase, wie im Beispiel π_2^i mit einem Intervall $[c, d]$ beschriftet, ist dies äquivalent zu einem mit $[c, d]$ beschrifteten Pfeil, der am Anfang der Phase startet und zum Ende der Phase zeigt.

Wird in einer Phase keine Annahme gemacht, das heißt $\pi_l^i \equiv 1$ und die Phase ist nicht mit einer Längenanforderung wie $[c, d]$ versehen, dann wird diese Phase durch eine gestrichelte Linie dargestellt.

3.2.2 Semantik

Die Semantik eines Constraint Diagram *cd* ist immer von der Form

$$[\![cd]\!] := \forall \varepsilon_i^X \cdot (\mathit{Annahmeteil} \Rightarrow \exists \delta_i^X \cdot (\mathit{Zusicherungsteil}))$$

Dabei steht $\forall \varepsilon_i^X$ für die Quantifizierung über alle ε_i^X, wobei X alle Observablen und i alle Phasen der zugehörigen Sequenz durchläuft. Analoges gilt für $\exists \delta_i^X$.

Der Annahmeteil und Zusicherungsteil ist eine logische Konjunktion aus sogenannten Sequenz- und Differenzformeln. Dabei handelt es sich um Teilmengen des DCs. Die Sequenzformeln lassen sich durch folgende Grammatik beschreiben:

$$\mathit{Sequ} ::= \ulcorner\urcorner \mid (\ulcorner \pi \urcorner \wedge \ell = \varepsilon) \mid \mathit{Sequ}_1 ; \mathit{Sequ}_2$$

Dabei ist π ein Zustandsausdruck $\neq 0$, ε eine globale Variable. Formeln der Form $\ulcorner\pi\urcorner \wedge \ell = \varepsilon$ werden Phasen genannt.

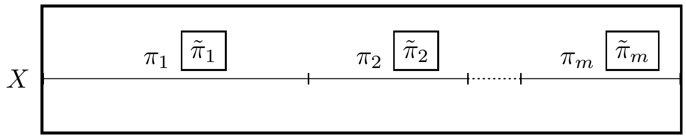
Differenzformeln haben immer die Form

$$\left(\sum_{i=1}^{m} \varepsilon_i - \sum_{j=1}^{n} \tilde{\varepsilon}_j\right) \in [a, b] \quad ,$$

wobei ε_i, $\tilde{\varepsilon}_j$ globale Variablen sind und $[a, b]$ ein Intervall aus $\mathbf{Time} \cup \{\infty\}$.

Semantik von Sequenzen

Gegeben sei eine Sequenz des Constraint Diagrams, wie zum Beispiel:



Diese Sequenz fügt zum Annahmeteil folgende Formel hinzu:

$$(\ulcorner\pi_1\urcorner \wedge \ell = \varepsilon_1^X); (\ulcorner\pi_2\urcorner \wedge \ell = \varepsilon_2^X); \cdots (\ulcorner\pi_n\urcorner \wedge \ell = \varepsilon_n^X)$$

Der Zusicherungsteil ist etwas komplizierter. Er prüft, ob die Annahmen für δ_i^X immer noch erfüllt sind, und zusätzlich die Zusicherungen:

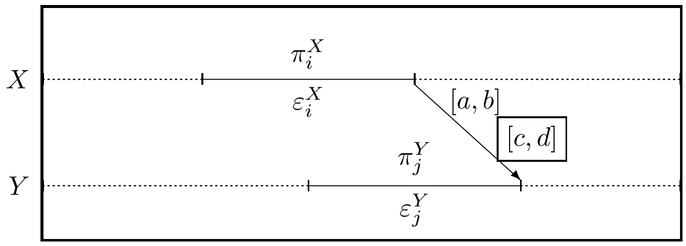
$$\mathsf{Pref}((\ulcorner\pi_1 \wedge \tilde{\pi}_1\urcorner \wedge \ell = \delta_1^X); (\ulcorner\pi_2 \wedge \tilde{\pi}_2\urcorner \wedge \ell = \delta_2^X); \cdots (\ulcorner\pi_n \wedge \tilde{\pi}_n\urcorner \wedge \ell = \delta_n^X))$$

Dabei ist Pref der sogenannte *Präfixoperator*. Er soll ausdrücken, dass das gerade betrachtete Intervall ein Anfangsstück eines Intervalls ist, auf dem die Formel gilt. Der *Präfixoperator* ist für Sequenzformeln induktiv definiert:

$$\begin{aligned}
\mathsf{Pref}(\ulcorner\urcorner) &\mathrel{\hat{=}} \ulcorner\urcorner \\
\mathsf{Pref}(\ulcorner\pi\urcorner \wedge \ell = \varepsilon) &\mathrel{\hat{=}} \ulcorner\urcorner \vee (\ulcorner\pi\urcorner \wedge \ell \leq \varepsilon) \\
\mathsf{Pref}(\mathit{Sequ}_1; \mathit{Sequ}_2) &\mathrel{\hat{=}} \mathsf{Pref}(\mathit{Sequ}_1) \vee \mathit{Sequ}_1; \mathsf{Pref}(\mathit{Sequ}_2)
\end{aligned}$$

Semantik von Pfeilen

Ein Pfeil in einem Constraint Diagram, fügt der Duration Calculus Formel einige Differenzformeln hinzu. Wir betrachten einen beliebigen Pfeil zwischen zwei Observablen:



Die Annahme eines solchen Pfeils ist, dass das Ende der Phase π_j^Y mindestens a und maximal b Sekunden nach dem Ender der Phase π_i^X ist. Der

Zeitpunkt, zu dem die Phase π_i^X endet, lässt sich durch Aufsummieren der ε_k^X für $k = 1, \ldots, i$ berechnen, analog für die Phase π_j^Y. Daher lässt sich die Annahme durch folgende Differenzformel ausdrücken:

$$\left(\sum_{k=1}^{j} \varepsilon_k^Y - \sum_{k=1}^{i} \varepsilon_k^X \right) \in [a, b]$$

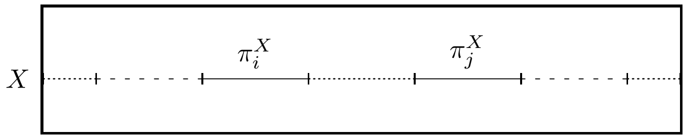
Beim Zusicherungsteil verwenden wir wieder die δ_i^X statt der ε_i^X. Wir müssen wieder prüfen, ob die Annahme auch mit den δ_i^X noch gilt:

$$\left(\sum_{k=1}^{j} \delta_k^Y - \sum_{k=1}^{i} \delta_k^X \right) \in [a, b] \wedge \left(\sum_{k=1}^{j} \delta_k^Y - \sum_{k=1}^{i} \delta_k^X \right) \in [c, d]$$

Fehlt eine der Intervall-Angaben $[a, b]$ oder $[c, d]$ so fallen die entsprechenden Zusicherungen und Annahmen weg.

Zusammenhang der ε_i^X und δ_i^X

Um den Zusammenhang zwischen Annahme- und Zusicherungsteil herzustellen, müssen wir dafür sorgen, dass der Zusicherungsteil die gleichen Phasen benutzt, wie der Annahmeteil. Wir fügen dafür weitere Formeln in den Zusicherungsteil ein. Diese Zusicherungen treten bei Pfeilen, sowie bei Übergängen zwischen spezifizierten und unspezifizierten Phasen auf. Betrachten wir zunächst den zweiten Fall:



Dabei seien π_i^X und π_j^X echte Zusicherungen $\neq 1$. In diesem Fall wird für den Übergang in die Phase π_i^X die Formel

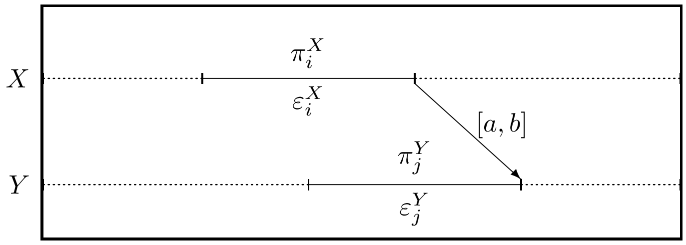
$$\sum_{k=1}^{i-1} \delta_k^X \leq \sum_{k=1}^{i-1} \varepsilon_k^X$$

zum Zusicherungsteil hinzugefügt und für den Übergang von π_i^Y zu **true** folgende Formel:

$$\sum_{k=1}^{j} \varepsilon_k^X \leq \sum_{k=1}^{j} \delta_k^X$$

Diese Formeln lassen sich leicht zu Differenzformeln umformen.

Für Pfeile, die in den Annahmeteil eingehen, werden ebenfalls solche Formeln hinzugefügt. Wir nehmen wieder einen Pfeil zwischen zwei Observablen:



Ein Pfeil darf in der Zusicherung nur früher beginnen und nur später enden als in der Annahme. Das wird durch folgende Formel im Zusicherungsteil ausgedrückt:

$$\sum_{k=1}^{i} \delta_k^X \leq \sum_{k=1}^{i} \varepsilon_k^X \quad \wedge \quad \sum_{k=1}^{j} \varepsilon_k^Y \leq \sum_{k=1}^{j} \delta_k^Y$$

Beispiel 3.2.1. Wir betrachten den Watchdog aus der Einleitung in Abbildung 3.3.

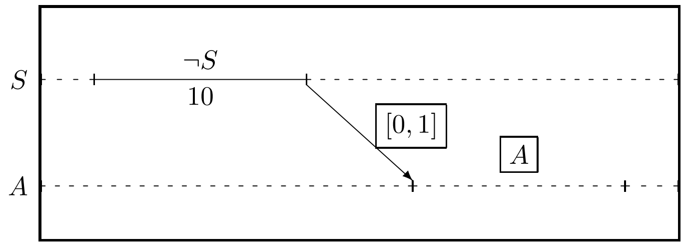


Abbildung 3.3: Spezifikation des Watchdog

Die wörtliche Übersetzung des Diagramms gemäß der vorgestellten Vorschrift ist:

$$\begin{aligned}
&\forall \varepsilon_1^S, \varepsilon_2^S, \varepsilon_3^S, \varepsilon_1^A, \varepsilon_2^A, \varepsilon_3^A \cdot \\
&\quad \big((\mathbf{true} \wedge \ell = \varepsilon_1^S); (\ulcorner \neg S \urcorner \wedge \ell = \varepsilon_2^S); (\mathbf{true} \wedge \ell = \varepsilon_3^S) \\
&\quad \wedge (\mathbf{true} \wedge \ell = \varepsilon_1^A); (\mathbf{true} \wedge \ell = \varepsilon_2^A); (\mathbf{true} \wedge \ell = \varepsilon_3^A) \\
&\quad \wedge ((\varepsilon_1^S + \varepsilon_2^S) - \varepsilon_1^S) \in [10, 10] \\
&\quad \Rightarrow \exists \delta_1^S, \delta_2^S, \delta_3^S, \delta_1^A, \delta_2^A, \delta_3^A \cdot \\
&\quad \big(\mathsf{Pref}((\mathbf{true} \wedge \ell = \delta_1^S); (\ulcorner \neg S \urcorner \wedge \ell = \delta_2^S); (\mathbf{true} \wedge \ell = \delta_3^S)) \\
&\quad \wedge \mathsf{Pref}((\mathbf{true} \wedge \ell = \delta_1^A); (\ulcorner A \urcorner \wedge \ell = \delta_2^A); (\mathbf{true} \wedge \ell = \delta_3^A)) \\
&\quad \wedge ((\delta_1^S + \delta_2^S) - \delta_1^S) \in [10, 10] \\
&\quad \wedge (\delta_1^A - (\delta_1^S + \delta_2^S)) \in [0, 1] \\
&\quad \wedge \delta_1^S \leq \varepsilon_1^S \wedge \varepsilon_1^S + \varepsilon_2^S \leq \delta_1^S + \delta_2^S\big)\big)
\end{aligned}$$

Es gilt in jedem Fall $\varepsilon_2^S = \delta_2^S = 10$. Wegen den letzten beiden Ungleichungen ist dann auch $\varepsilon_1^S = \delta_1^S$. Wenn man dann die nicht benötigten Variablen entfernt und die Formel etwas vereinfacht, ergibt sich die Formel:

$$\begin{aligned}
&\forall \varepsilon_1^S \cdot \big(\ell = \varepsilon_1^S; (\ulcorner \neg S \urcorner \wedge \ell = 10); \mathbf{true} \\
&\qquad \Rightarrow \exists \delta_1^A \cdot \big(\mathsf{Pref}(\ell = \delta_1^A; \ulcorner A \urcorner; \mathbf{true}) \wedge (\delta_1^A - (\varepsilon_1^S + 10)) \in [0, 1]\big)\big)
\end{aligned}$$

3.3 Eine Sicherheitseigenschaft der Fallstudie

Nachdem die Semantik von Constraint Diagrams geklärt ist, können wir nun eine Sicherheitseigenschaft der Fallstudie beweisen. Wir wollen jetzt zeigen, dass der

Zug die Schranke nicht kreuzen kann, bevor er eine Statusmeldung *OK* von der Schranke bekommen hat, oder der Zugführer manuell den ordnungsgemäßen Zustand der Schranke bestätigt hat. In Abbildung 3.4 ist diese Sicherheitseigenschaft als CD dargestellt. Es ist eine Stabilitätseigenschaft der Observablen *Position*: Wenn die Position *nicht kreuzend* ist, und die ganze Zeit über *nicht* $Statusmeldung = OK$ gilt, und *nicht Manuell* gesetzt ist, dann bleibt die Position *nicht kreuzend*. Wir haben auch in diesem Fall wieder zwei Diagramme: Ein initiales Stabilitätsdiagramm für den ersten Bahnübergang und ein zweites für alle weiteren Bahnübergänge, die vom Zug passiert werden.

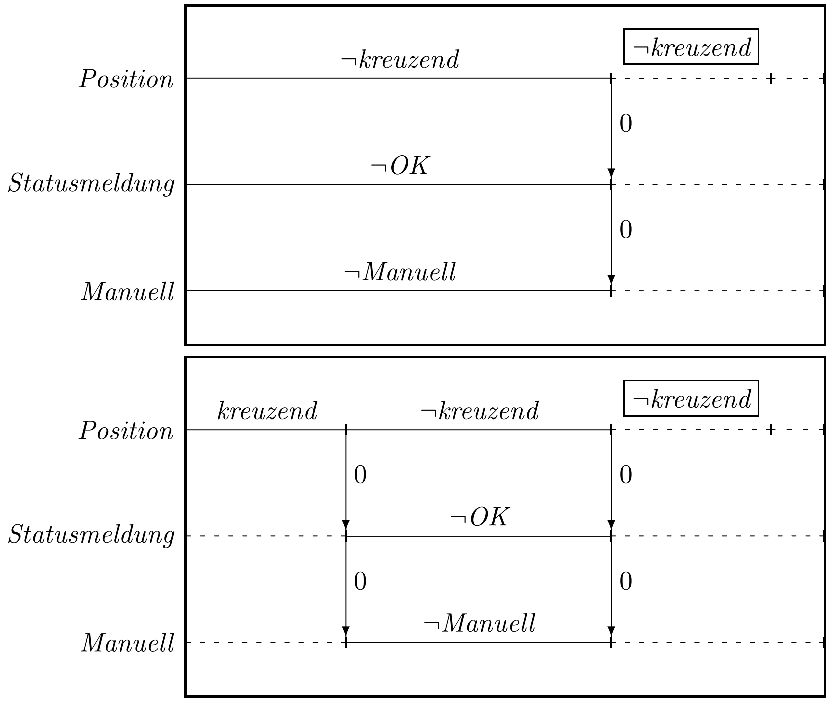
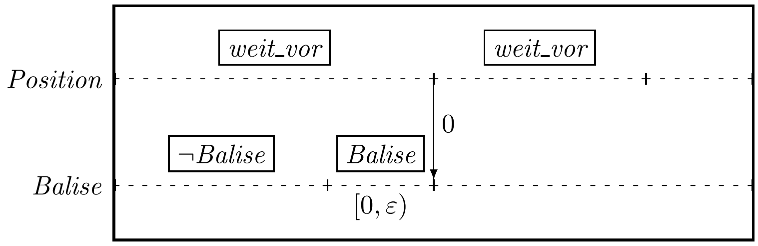


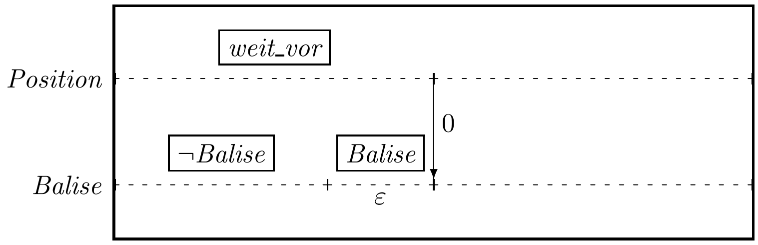
Abbildung 3.4: Die Sicherheitseigenschaft

Satz 3.3.1 (Sicherheitseigenschaft)**.** *Gegeben seien die Observablen aus Kapitel 2 und alle dort gezeigten Constraint Diagrams seien erfüllt. Dann gilt die Sicherheitseigenschaft, die durch die Constraint Diagrams in Abbildung 3.4 gegeben ist.*

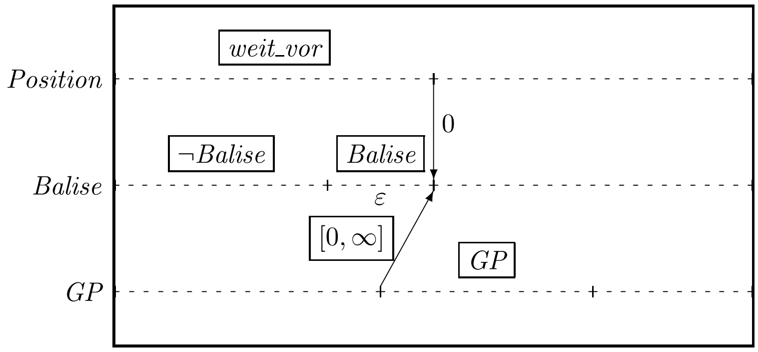
Beweis. Wir konzentrieren uns auf das erste Diagramm, nämlich die Sicherheitseigenschaft für den ersten Bahnübergang. Um sie zu beweisen, werden wir uns zusätzlich den Verlauf der Observablen *Balise* und *GP* anschauen. Wegen des Initialisierungsdiagramms in Abbildung 2.3 gilt initial $Position = weit\_vor$ und $\neg Balise$. Wegen Abbildung 2.13, die besagt, dass die Balise im Bereich *weit_vor* liegt, kann sich die *Position* nicht ändern, solange $\neg Balise$ gilt. Weil wir zeigen wollen, dass *nicht* $Position = kreuzend$ gilt, brauchen wir also nur den Fall zu betrachten, dass die Balise irgendwann gelesen wird:



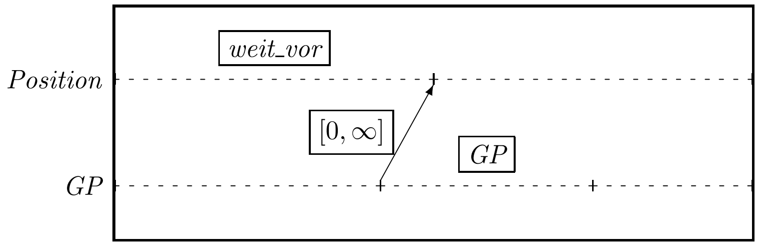
Wenn die Balise erst einmal gelesen wird, bleibt sie für ε Sekunden stabil (Abbildung 2.11). Es gilt also sogar folgendes Diagramm:



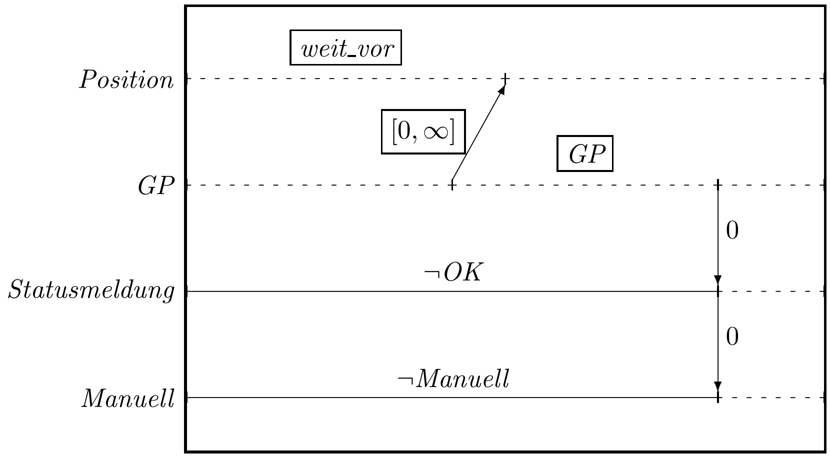
Nun können wir das Diagramm in Abbildung 2.5 (Setzen des Gefahrenpunktes) ausnutzen. Wenn der Gefahrenpunkt bis jetzt nicht gesetzt ist, so ist wird er spätestens jetzt gesetzt (wir wollen aber hier nicht ausschließen, dass er schon früher gesetzt worden ist):



Jetzt können wir wieder von der Balise abstrahieren. Für uns ist nur interessant, dass der Gefahrenpunkt in jedem Fall gesetzt wird, während die *Position* noch im Bereich *weit_vor* ist:

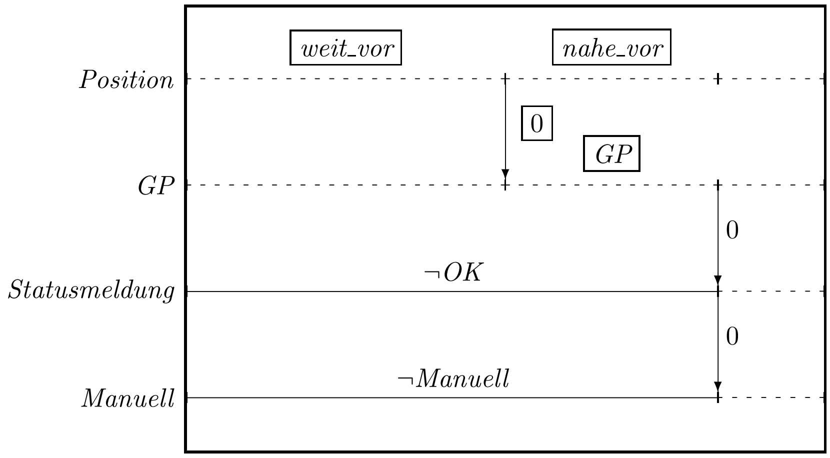


Weil der Gefahrenpunkt gesetzt bleibt, solange $Statusmeldung \neq OK$ und $\neg Manuell$ gilt (Abbildung 2.8), ergibt sich jetzt folgendes Diagramm:

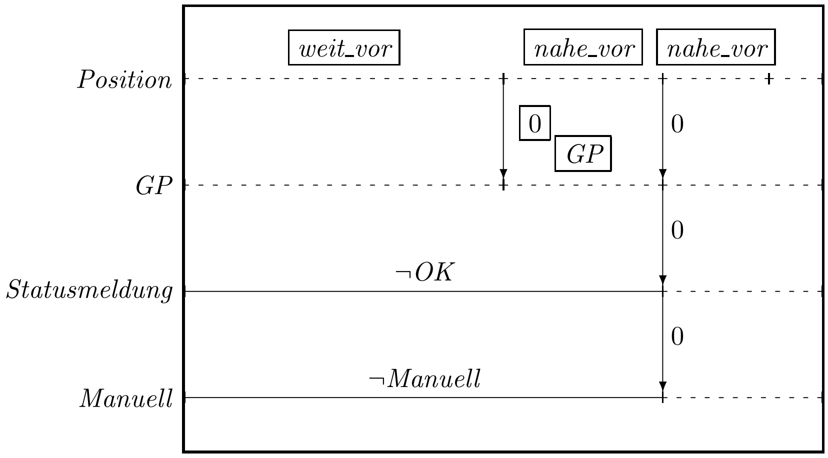


Nun betrachten wir den möglichen Verlauf der Position etwas genauer. Bleibt weiterhin bis zum Ende der Phase *GP* die *Position* = *weit_vor*, so ist unsere Sicherheitseigenschaft erfüllt. Denn die Sequenzdiagramme in Abbildung 2.4 garantieren uns, dass die Position anschliessend nicht sofort nach *kreuzend* wechseln kann.

Auf Position *weit_vor* kann aber sonst nur *nahe_vor* folgen. Wenn der Wechsel noch während der *GP*-Phase geschieht, haben wir folgende Situation:



Nun können wir die Funktion der Bremssteuerung (Abbildung 2.7) ausnutzen, die uns garantiert, dass die Position *nahe_vor* bleibt, solange der Gefahrenpunkt nicht gelöscht wird:



Das letzte Diagramm impliziert die Sicherheitseigenschaft für den ersten Bahnübergang. Um die Eigenschaft für die weiteren Bahnübergänge zu zeigen, beachte man, dass die Position von *kreuzend* nur über *hinter*, *weit_vor* und *nahe_vor* wieder nach *kreuzend* wechseln kann. Dann kann man beim Beginn der *weit_vor*-Phase ansetzen und dort den Beweis analog durchführen. □

Kapitel 4

Semantik von Real-Time Symbolic Timing Diagrams

Auch die Semantik von RTSTDs wird durch eine Temporale Logik ausgedrückt. Hierbei handelt es sich um eine Abwandlung der Timed Propositional Temporal Logic (TPTL), die wir im nächsten Abschnitt näher erläutern werden.

4.1 Timed Propositional Temporal Logic

Die der Semantik von Real-Time Symbolic Timing Diagrams zugrundeliegende Logik ist eine Abwandlung der Timed Propositional Temporal Logic [AH94] namens $\mathrm{TPTL}^{\mathrm{C}}$. Während TPTL sogenannte *freeze quantifier* benutzt, benutzt $\mathrm{TPTL}^{\mathrm{C}}$ Uhren und *reset quantifier*. Zum Beispiel lässt sich die Eigenschaft „jedesmal wenn p gilt wird q innerhalb von fünf Sekunden gelten“ in $\mathrm{TPTL}^{\mathrm{C}}$ durch folgende Formel ausdrücken:

$$\Box(z.p \Rightarrow \Diamond(q \wedge z \leq 5))$$

Dabei ist z eine Uhrenvariable, mit der die fünf Sekunden gemessen werden. Mit dem $\Box$-Operator wird ausgedrückt, dass diese Formel für alle Zeitpunkte gilt. Dann wird durch $z.$ die Uhr z auf null gesetzt und anschließend getestet, ob p gilt. Gilt p, so fordert der rechte Teil der Implikation, dass es später einen Zeitpunkt gibt, an dem q gilt und dass die Uhrenvariable z noch kleiner oder gleich fünf ist, d. h. es dürfen bis dahin maximal fünf Sekunden verstreichen.

4.1.1 Timed State Sequences

Wir betrachten eine Entity-Deklaration $ed \mathrel{\hat{=}} (\mathit{Vars}, \mathit{typedecl})$, wie sie in VHDL benutzt wird um Schnittstellen zu spezifizieren. Dabei sind $v \in \mathit{Vars}$ die Variablen, die sich über die Zeit ändern, und $\mathit{typedecl}(v)$ ist der endliche Typ von v. Damit ergeben sich die möglichen Zustände der Variablen durch folgende Menge:

$$\mathit{Val}_{ed} \mathrel{\hat{=}} \{\sigma : \forall v \in \mathit{Vars}_{ed} \cdot \sigma(v) \in \mathit{typedecl}(v)\}$$

Der semantische Bereich einer $\mathrm{TPTL}^{\mathrm{C}}$-Formel ist eine *Timed State Sequence* $tss = (\sigma, \tau)$, wobei σ eine Sequenz von Zuständen $\sigma_0\sigma_1\sigma_2\ldots$ mit $\sigma_i \in \mathit{Val}_{ed}$ für alle i und $\tau = \tau_0\tau_1\tau_2\ldots$ eine Folge der zugehörigen Zeitpunkte mit $\tau_i \in \mathbf{Time}$ ist. Dabei wird die Zeit **Time** diskret angenommen, also $\mathbf{Time} = \mathbb{N}$. Diese Einschränkung ist nötig, um das Problem der Erfüllbarkeit von Formeln berechenbar zu machen. Wir werden jedoch im nächsten Kapitel auch für TPTL^{C} $\mathbf{Time} = \mathbb{R}_{\geq 0}$ setzen, um die DC und TPTL^{C} vergleichbar zu machen.

Die Zeitfolge hat noch zwei weitere Einschränkungen:

- Monotonie: $\tau_i \leq \tau_{i+1}$ für alle $i \in \mathbb{N}$.
- Non-Zeno: Zu jedem Zeitpunkt $t \in \mathbf{Time}$ gibt es ein $i \in \mathbb{N}$, sodass $\tau_i > t$ ist.

Mit σ^i bzw. τ^i wird die Teilfolge von σ bzw. τ bezeichnet, die durch das Streichen der ersten i Elemente aus σ bzw. τ hervorgeht, d. h. $\sigma^i = \sigma_i\sigma_{i+1}\sigma_{i+2}\ldots$. Außerdem definieren wir $tss^i \mathrel{\hat{=}} (\sigma^i, \tau^i)$.

4.1.2 Uhren

Wie bereits erwähnt betrachten wir in TPTL^{C} eine Menge C von Uhrenvariablen $z \in C$. Ihre Interpretation ist durch eine Uhren-Umgebung $\mathcal{E}$ gegeben. Dabei handelt es sich um eine partielle Funktion $\mathcal{E} : C \nrightarrow \mathit{Time}$, die den Wert einiger Uhrenvariablen festlegt.

Das Laufen der Uhren wird durch die Funktion $\mathcal{E} + t$ beschrieben, die wie folgt definiert ist:

$$(\mathcal{E} + t)(z) = \begin{cases} \mathcal{E}(z) + t & \text{falls } \mathcal{E}(z) \text{ definiert} \\ \text{undefiniert} & \text{sonst} \end{cases}$$

Das Zurücksetzen einer Uhrenvariable z beschreiben wir mit $\mathcal{E}[z := 0]$. Es ist $\mathcal{E}[z := 0](z) = 0$ und $\mathcal{E}[z := 0](z') = \mathcal{E}(z')$ für $z' \in C \setminus \{z\}$.

4.1.3 Syntax

Um die Syntax von TPTL^{C}-Formeln zu beschreiben legen wir zunächst fest, welche Vergleiche auf die Uhrenvariablen angewendet werden dürfen:

$$\pi := z \leq t \mid z_1 + t_1 \leq z_2 + t_2$$

Dabei sind $z, z_1, z_2 \in C$ beliebige Uhrenvariablen und $t, t_1, t_2 \in \mathbf{Time}$. Wir erlauben also keine beliebigen arithmetischen Operationen auf den Uhrenvariablen. Die Syntax von TPTL^{C}-Formeln ist wie folgt:

$$\varphi := \mathit{Pred}_{ed} \mid \pi \mid \mathbf{false} \mid \varphi_1 \Rightarrow \varphi_2 \mid \bigcirc\varphi \mid \varphi_1\,\mathcal{U}\,\varphi_2 \mid z.\varphi$$

Dabei ist Pred_{ed} ein typkorrektes Prädikat über die Variablen aus Var_{ed}. Prädikate beschreiben den ersten Zustand einer Timed State Sequence. Ein Vergleich π zweier Uhren wird mithilfe der Uhrenumgebung ausgewertet. Die Formeln **false** und $\varphi_1 \Rightarrow \varphi_2$ haben die übliche logische Bedeutung. Der Operator $\bigcirc$, überspringt den ersten Zustand einer Timed State Sequence, d. h. $\bigcirc\varphi$ ist wahr unter tss, wenn φ unter tss^1 wahr ist. Der Operator $\mathcal{U}$ besagt, dass irgendwann in der Timed State Sequence φ_2 gilt, und für alle vorherigen Schritte muss φ_1 gelten[1]. Mit dem Reset-Quantifier $z.\varphi$ wird schließlich eine Uhr auf null gesetzt.

4.1.4 Semantik

Sei nun $tss = (\sigma, \tau)$ eine Timed State Sequence und $\mathcal{E}$ eine Uhrenumgebung. Eine TPTL$^{\mathrm{C}}$-Formel φ wird durch das Paar $(tss, \mathcal{E})$ erfüllt, gdw. $tss \models_{\mathcal{E}} \varphi$, wobei $\models_{\mathcal{E}}$ wie folgt induktiv über den Aufbau von φ definiert wird:

$$
\begin{array}{lll}
tss \models_{\mathcal{E}} Pred_{ed} & \text{gdw.} & \sigma_0 \models Pred_{ed} \\
tss \models_{\mathcal{E}} z \leq t & \text{gdw.} & \mathcal{E}(z) \leq t \\
tss \models_{\mathcal{E}} z_1 + t_1 \leq z_2 + t_2 & \text{gdw.} & \mathcal{E}(z_1) + t_1 \leq \mathcal{E}(z_2) + t_2 \\
tss \not\models_{\mathcal{E}} \mathbf{false} & & \\
tss \models_{\mathcal{E}} \varphi_1 \Rightarrow \varphi_2 & \text{gdw.} & tss \models_{\mathcal{E}} \varphi_1 \text{ impliziert } tss \models_{\mathcal{E}} \varphi_2 \\
tss \models_{\mathcal{E}} \bigcirc \varphi & \text{gdw.} & tss^1 \models_{\mathcal{E}+\tau_1-\tau_0} \varphi \\
tss \models_{\mathcal{E}} \varphi_1 \,\mathcal{U}\, \varphi_2 & \text{gdw.} & \exists i \geq 0 \cdot tss^i \models_{\mathcal{E}+\tau_i-\tau_0} \varphi_2 \\
 & & \wedge \forall j < i \cdot tss^j \models_{\mathcal{E}+\tau_j-\tau_0} \varphi_1 \\
tss \models_{\mathcal{E}} z.\varphi & \text{gdw.} & tss \models_{\mathcal{E}[z:=0]} \varphi
\end{array}
$$

Dabei bedeutet $\sigma_0 \models Pred_{ed}$, dass die übliche Interpretation des Prädikates $Pred_{ed}$ unter der Belegung σ_0 sich zu **true** auswertet.

Eine TPTL$^{\mathrm{C}}$-Formel φ wird durch eine Timed State Sequence tss erfüllt, in Zeichen $tss \models \varphi$, gdw. für alle Uhrenumgebungen $\mathcal{E}$ gilt $tss \models_{\mathcal{E}} \varphi$.

4.1.5 Abkürzungen

Wie schon bei DC gibt es einige Operatoren, die man auf die obigen Grundoperatoren reduzieren kann:

- Die logische Negation lässt sich durch $\neg\varphi \mathrel{\hat{=}} \varphi \Rightarrow \mathbf{false}$ ausdrücken.
- Nun lässt sich $\mathbf{true} \mathrel{\hat{=}} \neg\mathbf{false}$ definieren.
- Logisches ODER und UND kann man mit $\neg$ und $\Rightarrow$ definieren:

  $$\varphi_1 \vee \varphi_2 \mathrel{\hat{=}} \neg\varphi_1 \Rightarrow \varphi_2 \quad \varphi_1 \wedge \varphi_2 \mathrel{\hat{=}} \neg(\varphi_1 \Rightarrow \neg\varphi_2)$$

- Mit den Operator $\Diamond$ und $\Box$ kann über die Zeit „quantifiziert“ werden. Die Formel $\Diamond\varphi$ bedeutet, dass es einen (späteren) Zeitpunkt gibt, an dem φ gilt. Dies lässt sich mit dem $\mathcal{U}$-Operator leicht ausdrücken:

  $$\Diamond\varphi \mathrel{\hat{=}} \mathbf{true}\,\mathcal{U}\,\varphi$$

  Die Formel $\Box\varphi$ bedeutet, dass φ zu jedem Zeitpunkt gilt. Es gibt eine analoge Definition, wie in DC:

  $$\Box\varphi \mathrel{\hat{=}} \neg\Diamond\neg\varphi$$

[1] In [Fey96] wird statt des $\mathcal{U}$-Operators der **until**-Operator mit einer etwas anderen Semantik verwendet. Ich habe mich hier an [AH94] gehalten. Später wird der **until**-Operator mithilfe von $\mathcal{U}$ und $\bigcirc$ definiert

- Für die Beschreibung der Semantik von RTST-Diagrammen werden wir später den **until** und **unless** Operator brauchen. Die Formel φ_1 **until** φ_2 hat eine ähnliche Bedeutung, wie $\varphi_1 \, \mathcal{U} \, \varphi_2$, fordert allerdings, dass φ_1 auch tatsächlich einen Schritt lang Anfang gilt. Bei φ_1 **unless** φ_2 wird nicht gefordert, dass der Wechsel auf φ_2 irgendwann einmal stattfindet:

$$\varphi_1 \textbf{ until } \varphi_2 \mathrel{\hat{=}} \varphi_1 \wedge \bigcirc(\varphi_1 \, \mathcal{U} \, \varphi_2)$$
$$\varphi_1 \textbf{ unless } \varphi_2 \mathrel{\hat{=}} (\Box\varphi_1) \vee (\varphi_1 \textbf{ until } \varphi_2)$$

- Wir erlauben auch mehrere Uhren gleichzeitig auf null zu setzen:

$$\{z_1, \ldots, z_n\}.\varphi \mathrel{\hat{=}} z_1. \cdots z_n.\varphi$$

4.2 Real-Time Symbolic Timing Diagrams

Um die Semantik eines Real-Time Symbolic Timing Diagram zu berechnen, wird zuerst die sogenannte *Unwinding Structure* dieses Diagramms ermittelt. Dabei handelt es sich um einen *Timed Büchi Automat*, der dieses Diagramm Schritt für Schritt abarbeitet.

Um diesen Automaten zu definieren, müssen wir die Abschnitt 1.3.2 erwähnten Begriffe präzisieren. Ein Real-Time Symbolic Timing Diagramm *td* ist ein Tupel $td = (A, BW, e_{act}, RT)$, bestehend aus einem Aktivierungsmodus A, einem Aktivierungsereignis e_{act} einem *Bündel von Waveforms BW*(einer Menge von *Waveforms*) und einer Menge von *Constraints RT*. Der Aktivierungsmodus $A \in \{Initial, Invariant\}$ bestimmt, ob das Diagramm nur am Anfang gilt (*Initial*), oder ob es zu jedem Zeitpunkt anwendbar ist (*Invariant*).

4.2.1 Waveform

Eine *Waveform* $\rho \in BW$ besteht wiederum aus mehreren Teilen (vgl. Abbildung 4.1).

$$\rho = (SE, \rightarrow, actc, contc, exitc)$$

Dabei ist *SE* eine endliche Menge von Ereignissen, z. B. $\{e_1, e_2, e_\top\}$. Die Nachfolgerfunktion $\rightarrow$ ist eine Funktion $\rightarrow: SE \rightarrow SE$, die jedem Ereignis, das nachfolgende Ereignis zuordnet. Der reflexive und transitive Abschluss dieser Funktion $\leq \hat{=} \rightarrow^*$ soll dabei eine totale Ordnung auf *SE* induzieren.

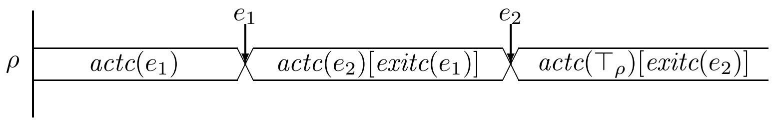


Abbildung 4.1: Allgemeiner Aufbau einer Waveform

Die Funktionen *actc*, *contc* und *exitc* ordnen jedem Ereignis eine Aktivierungs-, Fortsetzungs- bzw. Ausstiegsbedingung zu. Dabei ist die Fortsetzungsbedingung gleich der Aktivierungsbedingung des bzgl. $\rightarrow$ nachfolgenden Ereignisses:

$$e_i \rightarrow e_j \Rightarrow contc(e_i) = actc(e_j)$$

Das bezüglich der Ordnung $\leq = \rightarrow^*$ maximale Ereignis wird mit $\top_\rho$ bezeichnet und dessen Fortsetzungs- und Ausstiegsbedingung ist immer erfüllt:

$$contc(\top_\rho) = exitc(\top_\rho) = \textbf{true}$$

4.2.2 Bündel von Waveforms

In einem Diagramm werden mehrere Waveforms zu einem Bündel BW zusammengefasst. Einzige Bedingung ist, dass die Ereignisse verschiedener Waveformen disjunkt sind:

$$\rho_i, \rho_j \in BW, i \neq j \Rightarrow SE_{\rho_i} \cap SE_{\rho_j} = \emptyset$$

Die Phasen eines Bündels sind dann Mengen von Ereignissen verschiedener Waveformen, wobei von jeder Waveform genau ein Ereignis vertreten ist. Diese Phasen werden mit $Front_{BW}$ bezeichnet:

$$Front_{BW} \mathrel{\widehat{=}} \left\{ \zeta \subset \bigcup_{\rho \in BW} SE_\rho : \forall \rho \in BW \cdot \exists^1 e \in SE_\rho : e \in \zeta \right\}$$

Die Ereignisse $e \in \zeta$ werden die in Phase ζ aktivierten Ereignisse genannt.

Die Nachfolgerfunktionen $\rightarrow_\rho$ induzieren eine Relation $\rightarrow_\zeta$ auf $Front_{BW}$ wie folgt: Zwei Phasen ζ_i und ζ_j stehen in der Relation, falls ζ_j aus ζ_i durch Ersetzung ein oder mehrerer Ereignisse durch ihre Nachfolger hervorgeht:

$$\zeta_i \rightarrow_\zeta \zeta_j \iff \exists E \subset \zeta_i, E \neq \emptyset \cdot \zeta_j = (\zeta_i \setminus E) \cup \{e' : \exists e \in E, \rho \in BW \cdot e \rightarrow_\rho e'\}$$

Die bezüglich der neuen Nachfolgerfunktion $\rightarrow_\zeta$ letzte Phase ist

$$\top_{BW} \mathrel{\widehat{=}} \{\top_\rho : \rho \in BW\}$$

Während eine Phase aktiv ist, muss die Aktivierungsbedingung aller Ereignisse dieser Phase aktiv bleiben, bis sie abgearbeitet werden. Dies gilt jedoch nicht für das letzte Ereignis $\top_\rho$:

$$cond(\zeta) = \bigwedge_{e \in \zeta \setminus \top_{BW}} actc(e)$$

4.2.3 Unwinding Structure

Um die Semantik eines Real-Time Symbolic Timing Diagram td zu bestimmen, wird zu diesem Diagramm zunächst die *Unwinding Structure* $US(td)$ berechnet. Dabei handelt es sich um einen zeitbehafteten Büchi-Automaten $(S, \zeta_0, C, \mathcal{T}, F)$ mit folgenden Eigenschaften:

- Die Zustände S des Automaten sind die Phasen $\zeta \in Front_{BW}$ plus einem zusätzlichen Zustand ζ_{exit}.
- Der Startzustand ζ_0 ist die erste Phase des Diagramms $\zeta_0 := \{\perp_\rho : \rho \in BW\}$, wobei $\perp_\rho$ das kleinste Element bzgl. $\leq$ ist.

- Die Uhren C umfassen eine Uhr z_e für jedes Ereignis $e \in \bigcup SE_\rho$ und eine Uhr z_{act} für das Aktivierungsereignis e_{act}. Jedesmal, wenn der Automat von einem Zustand ζ_i nach ζ_j wechselt, werden die Uhren der zugehörigen Ereignisse $E := \zeta_i \setminus \zeta_j$ auf null gesetzt.

- Die Transitionen $\mathcal{T}$ entsprechen der schrittweisen Abarbeitung des Diagramms. Wenn $\zeta_i \rightarrow_\zeta \zeta_j$ gilt, so kann der Automat von ζ_i nach ζ_j wechseln, sofern dabei alle weak und strong Constraints eingehalten werden.

  Gibt es einen weak Constraint, der in der Phase ζ_i verletzt werden kann, so existiert eine entsprechende Transition von ζ_i zum Zustand ζ_{exit}. Das gleiche gilt auch, wenn die Phase eine Ausstiegsbedingung besitzt.

  Die genaue Definition der Transitionen findet sich in [Fey96]. Es ist dabei zu beachten, dass, falls mehrere Constraints in einem Schritt verletzt wird, anhand der Uhren getestet werden muss, welcher Constraint (strong oder weak) zuerst verletzt wurde.

- Die akzeptierenden Zustände $F \subset S$, sind alle Phasen, die keine strong *leads-to* Constraints besitzen. Das heißt für alle strong leads-to Constraints, sind entweder beide beteiligten Ereignisse bereits abgearbeitet, oder das erste Ereignis wurde noch nicht abgearbeitet. Ein Ereignis wird abgearbeitet, wenn eine Transition $\zeta_i \rightarrow_{BW} \zeta_j$ durchgeführt wird, wobei $e \in \zeta_i \setminus \zeta_j$ liegt.

Die Abbildung 4.2 zeigt die *Unwinding Structure* des Senders von Abbildung 1.7 auf Seite 9. Wir haben den Automaten hier dahingehend abgewandelt, dass wir die Transitionen von einem Zustand in sich selbst hinzugefügt haben. Diese Transitionen werden in [Fey96] mit $\mathcal{T}_{stut}$ bezeichnet und extra behandelt.

4.2.4 TPTL$^\mathrm{C}$-Semantik

Die im letzten Abschnitt erwähnt Unwinding Structure wird in einem letzten Schritt nach TPTL$^\mathrm{C}$ übersetzt. Dazu definieren wir eine Übersetzungsfunktion *TL* wie folgt:

- Ein Zustand ζ, der keine weiterführende Transitionen hat, schränkt den weiteren Verlauf nicht ein:

  $$TL(S, \zeta, C, \mathcal{T}, F) = \textbf{true} \text{ falls } \nexists \zeta_j \neq \zeta \cdot (\zeta, \zeta_j, \_, \_, \_) \in \mathcal{T}$$

  Dabei steht _ für ein nicht näher spezifiziertes Element. Nur der spezielle Zustand ζ_{exit} und der letzte Zustand $\{\top_\rho : \rho \in BW\}$ erfüllen diese Eigenschaft.

- Andernfalls, wenn zusätzlich der Zustand $\zeta \in F$ liegt, so wird der Automat mit einem **unless**-Operator übersetzt:

  $$\begin{aligned} TL(S, \zeta, C, \mathcal{T}, F) = &\ cond(\zeta) \textbf{ unless} \\ &\bigvee_{(\zeta, \zeta_j, trcond, reset, timing) \in \mathcal{T}, \zeta_j \neq \zeta} trcond \wedge timing \wedge reset.TL(S, \zeta_j, C, \mathcal{T}, F) \end{aligned}$$

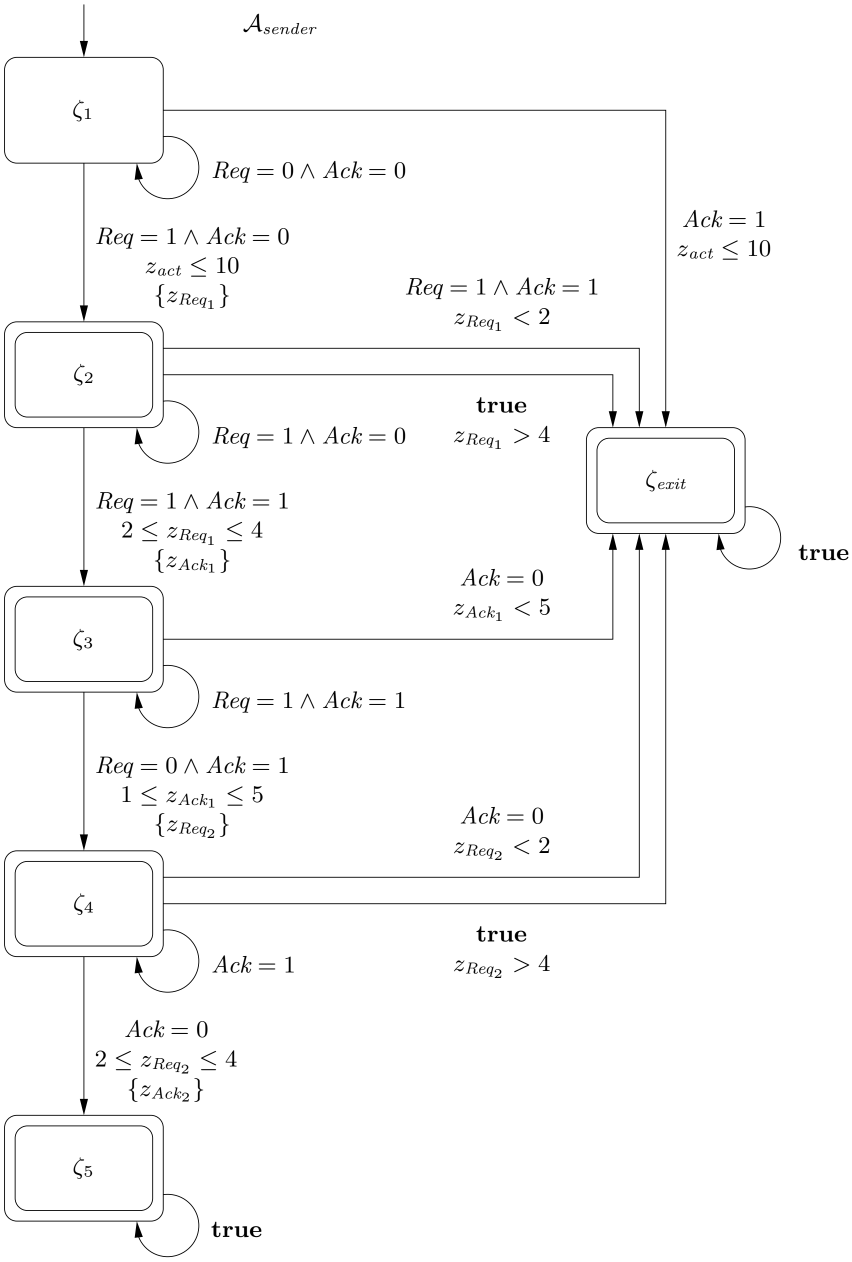


Abbildung 4.2: Unwinding Structure des Senders

- Wenn der Zustand $\zeta \in S \setminus F$ liegt, so wird statt des **unless** eine **until**-Operator benutzt:

$$TL(S, \zeta, C, \mathcal{T}, F) = cond(\zeta) \textbf{ until} \bigvee_{(\zeta, \zeta_j, trcond, reset, timing) \in \mathcal{T}, \zeta_j \neq \zeta} trcond \wedge timing \wedge reset.\, TL(S, \zeta_j, C, \mathcal{T}, F)$$

Weil es in der Unwinding Structure keine Zyklen gibt (sofern man von den Stotterschritten absieht), lässt sich die Übersetzung *TL* auflösen.

Anschließend wird die TL-Formel der Unwinding Structure noch in eine weitere Formel eingepackt, um die Semantik des RTSTDs *td* zu erhalten:

- Falls *td* ein initiales Diagramm ist (d. h. Aktivierungsmodus $A = Initial$), muss im ersten Zustand die Bedingung der ersten Phase gelten und anschließend der Automat den weiteren Verlauf akzeptieren:

$$TL(td) = cond(\zeta_0) \wedge z_{act}.\, TL(S, \zeta_0, C, \mathcal{T}, F)$$

- Andernfalls (Aktivierungsmodus $A = Invariant$) muss für alle Zeitpunkte, zu denen die Bedingung der ersten Phase erfüllt ist, der Automat den weiteren Verlauf akzeptieren.

$$TL(td) = \Box(cond(\zeta_0) \Rightarrow z_{act}.\, TL(S, \zeta_0, C, \mathcal{T}, F))$$

Kapitel 5

Angleichung der Semantiken

Ziel dieser Arbeit ist es, eine Teilklasse von Constraint Diagrams in Real-Time Symbolic Timing Diagrams zu übersetzten. Die in den vorherigen Kapiteln definierten Semantiken werfen allerdings einige Schwierigkeiten auf:

- Im Duration Calculus ist die Interpretation einer Observablen X eine Funktion $\mathcal{I}(X) : \mathbf{Time} \to D$, wobei D der Datenbereich der Observablen ist.

  In TPTL dagegen ist die Interpretation indirekt durch eine Timed State Sequence $tss = (\sigma, \tau)$, d. h. durch eine Folge von Zuständen σ und einer Folge von Zeitpunkten τ gegeben. Der Wert einer Observablen X ist dort durch die Folge $(\sigma_i(X))_{i\in\mathbb{N}}$ gegeben, wobei die zugehörige Folge $(\tau_i)_{i\in\mathbb{N}}$ den Zeitpunkt angibt, zu dem der Wert angenommen wird. Über den Verlauf der Observablen zwischen zwei Messpunkten τ_i und τ_{i+1} wird keine Aussage gemacht. Man darf aber annehmen, dass der in τ_i angenommene Wert $\sigma_i(X)$ bis zum Zeitpunkt τ_{i+1} stabil bleibt.

- Wenn bei einer Timed State Sequence $tss = (\sigma, \tau)$ ein Zustand mehrmals wiederholt wird, etwa $\sigma_i = \sigma_{i+1}$ für ein $i \in \mathbb{N}$, ändert sich das Verhalten der Observablen nicht. Man spricht hierbei auch von *Stotterschritten*. Wenn man das Folgenelement σ_{i+1} und den dazugehörigen Zeitpunkt τ_{i+1} streicht, erhält man eine Timed State Sequence, die genau das gleiche Verhalten der Observablen beschreibt. Das Problem ist nun, dass es RTSTDs gibt, die eine der beiden Timed State Sequences akzeptieren, nicht jedoch die andere.

- Weil bei TPTL nur schwache Monotonie $\tau_i \leq \tau_{i+1}$ für alle $i \in \mathbb{N}$ gefordert wird, kann eine Observable zu einem Zeitpunkt mehrfach den Wert wechseln. In DC ist das nicht möglich.

- Ein weiterer Unterschied ist die Auffassung der Zeit selbst. Im Duration Calculus gilt $\mathbf{Time} = \mathbb{R}_{\geq 0}$, in TPTL dagegen $\mathbf{Time} = \mathbb{N}$.

Wir wenden uns zunächst dem letzten Punkt zu.

5.1 TPTL mit reeller Zeit

Es spricht nichts dagegen, auch in TPTL **Time** $= \mathbb{R}_{\geq 0}$ zu setzen. Die im letzten Kapitel definierte Semantik kann unverändert übernommen werden. Es stellt sich daher die Frage, warum TPTL überhaupt mit diskreter Zeit **Time** $= \mathbb{N}$ definiert wurde. Die Antwort findet sich in [AH94]. Dort wird nämlich gezeigt, dass bereits für rationale Zeit das Problem unentscheidbar ist, ob eine TPTL-Formel erfüllbar, bzw. allgemeingültig ist. Für diskrete Zeit wird dagegen ein Verfahren angegeben, mit der sich die Erfüllbarkeit einer Formel nachrechnen lässt.

Wir sind in dieser Arbeit nicht auf eine automatische Verifikation der TPTL-Formeln angewiesen, daher können wir die Zeit reell setzen.

Die Alternative wäre natürlich DC mit diskreter Zeit zu betrachten [HZ97]. Die Semantik eines CDs kann dann in diskretem Duration Calculus unverändert übernommen werden. Es müssen nur alle Zeitintervalle ganzzahlige Grenzen haben. Mit dieser Einschränkung verlangt man aber auch, dass sich Umgebungsvariablen nur zu diskreten Zeitpunkten ändern, oder man gibt sich mit einer Annäherung des Modells an den tatsächlichen Verlauf zufrieden.

5.2 Einschränkung der Timed State Sequences

Timed State Sequences *tss* erlauben, dass verschiedene Zustände zur gleichen Zeit angenommen werden, falls die Zeitfolge $\tau_0\tau_1\ldots$ nicht *streng* monoton wachsend ist. Dies hat zur Folge, dass eine Observable zu einem Zeitpunkt mehrfach den Zustand ändern kann. Im Duration Calculus kann auf solche Änderungen nicht reagiert werden, denn der einzige Operator den man auf Zustandsausdrücken anwenden kann, ist der Intergral-Operator und dieser berücksichtigt punktförmige Änderungen nicht.

Die im letzten Kapitel vorgestellte Unwinding Structure eines RTSTD reagiert aber sehr wohl auf solche Zustandsänderungen. Dies führt zu merkwürdigen Effekten: Zum Beispiel würde ein Constraint, der Gleichzeitigkeit fordert, verletzt, wenn die zugehörigen Ereignisse durch zwei verschiedene Zustandsänderungen abgearbeitet werden, selbst wenn sie zum gleichen Zeitpunkt geschehen. Auch würde ein Zustand, der nur für einen Punkt anliegt und dann durch den vorherigen ersetzt wird, von Duration Calculus Formeln und damit von einem CD überhaupt nicht „bemerkt", während die Unwinding Structure eines RTSTD zwei Ereignisse abarbeiten kann.

Diese Effekte würden eine korrekte Übersetzung unmöglich machen. Deshalb werde ich mich im Folgenden auf Timed State Sequences $tss = (\sigma, \tau)$ beschränken, bei denen die Zeitfolge τ streng monoton wächst. Wir fordern also $\tau_i \lneqq \tau_{i+1}$ für alle $i \in \mathbb{N}$.

Eine Alternative wäre, die Semantik von RTSTDs abzuändern, dass diese Effekte keine Auswirkung mehr haben. Die Unwinding Structure könnte zum Beispiel bei jeder Änderung einer Observablen eine zusätzliche Uhr auf null setzen und erst den Zustand verlassen, wenn diese Uhr einen Wert größer als null hat. Siehe dazu auch [DFMV98], wo dieses Verfahren für die Timed Automata Semantik von SPS-Automaten eingesetzt wurde.

Eine andere Alternative, wäre den DC zu erweitern, sodass Abweichungen der Länge null erkannt werden können (siehe dazu auch [LRL98]). Anschließend müsste man dann auch die Semantik von CDs abändern, dass sie die Erweiterung des DC nutzen.

5.3 Zusammenhang der Observablen in DC und TPTL

Wie bereits am Anfang dieses Kapitels erwähnt ist die Semantik einer Observablen im DC eine Funktion $\mathbf{Time} \to D$, während in TPTL die Semantik indirekt durch zwei Folgen $\sigma = \sigma_0\sigma_1 \dots$ und $\tau = \tau_0\tau_1 \dots$ gegeben ist, wobei σ_i jeder Observablen V einen Wert zuweist und τ_i den Zeitpunkt bestimmt zu dem in den Zustand σ_i gewechselt wird.

Wir verlangen jetzt und im Folgenden, dass wir in DC und TPTL^{C} die gleichen Observablen mit den gleichen Datentypen haben. Ferner sei in DC die Menge von Konstanten, Funktions- und Prädikatssymbolen, sowie deren Interpretation fixiert. Nun wollen wir den Zusammenhang zwischen den Observablen in TPTL^{C} und DC herstellen, indem wir definieren, wann eine Interpretation $\mathcal{I}$ zugehörig zu einer Timed State Sequence $tss = (\sigma, \tau)$ ist.

Definition 5.3.1. *Sei $tss = (\sigma, \tau)$ eine Timed State Sequence. Eine Interpretation $\mathcal{I}$ heißt* zugehörig zu *tss, falls für alle Observablen X, die Interpretation $\mathcal{I}(X)$ durch folgende Vorschrift gegeben ist:*

$$\mathcal{I}(X)(t) = \sigma_i(X) \quad \textit{für alle } t \in (\tau_i, \tau_{i+1}), \textit{ für alle } i \in \mathbb{N}$$

Diese Definition folgt der am Anfang des Kapitels erwähnten Annahme, dass der zum Zeitpunkt τ_i angenommene Zustand σ_i bis zum nächsten Zeitpunkt τ_{i+1} stabil bleibt.

Eine Timed State Sequence *tss* kann mehrere zugehörige Interpretationen besitzen, denn der Wert der Observablen in den Zeitpunkten τ_i für $i \in \mathbb{N}$ wird nicht durch die obige Definition festgelegt. Allerdings hängt der Wahrheitswert einer DC-Formel nicht von den Werten der Observablen in τ_i ab. Wir können die Äquivalenzklasse der Interpretationen bilden, die sich nur an endlich vielen Zeitpunkten in jedem endlichen Intervall unterscheiden. Dann liegen alle zugehörigen Interpretationen einer Timed State Sequence *tss* in der gleichen Äquivalenzklasse. Den Vertreter dieser Klasse bezeichnen wir mit *der zugehörigen Interpretation* $\mathcal{I}_{tss}$[1]. Der Vertreter erfüllt eine DC-Formel, genau dann wenn alle zugehörigen Interpretationen die Formel erfüllen, genau dann wenn irgendeine zugehörige Interpretation die Formel erfüllt.

Es stellt sich noch die Frage, ob auch jede Interpretation zugehörig zu einer Timed State Sequence ist. Diese Frage wird vom folgenden Satz geklärt:

Satz 5.3.2. *Zu jeder Interpretation $\mathcal{I}$ gibt es eine Timed State Sequence tss, sodass $\mathcal{I}$ zugehörig zu tss ist.*

Beweis. Die gesuchte Timed State Sequence *tss* kann man auf folgende Weise erhalten: Weil die Observablen einer Interpretation von endlicher Variabilität sind, gibt es in einem Intervall $[0, t]$, $t \in \mathbf{Time}$, nur endlich viele Zeitpunkte

[1] Es kann passieren, dass der Vertreter $\mathcal{I}_{tss}$ keine zugehörige Interpretation ist. Durch geeignete Wahl des Vertreters kann man das aber ausschließen.

$\tau_0 < \tau_1 < \ldots < \tau_n$ (mit $\tau_0 = 0$), an denen sich die Interpretation einer Observablen ändert. Zwischen diesen Zeitpunkten behalten alle Observablen einen konstanten Wert. Setze nun für $0 \leq i < n$ und alle Observablen X:

$$\sigma_i(X) := \mathcal{I}(X)(\tau), \text{ wobei } \tau \in (\tau_i, \tau_{i+1}) \text{ beliebig}$$

Man beachte, dass der Wert von $\sigma_i(X)$ nicht von der Wahl von τ abhängt. Auf diese Weise erhält man das Anfangsstück einer Timed State Sequence. Vergrößert man t, so erhält man ein längeres Anfangsstück, das auf den ersten n Elementen übereinstimmt. Lässt man t gegen unendlich laufen, so erhält man schließlich die gesuchte Folge *tss*. $\square$

$$\begin{tikzcd}
\begin{array}{c}\text{RTSTD}\\ td\end{array} \arrow[r, leftrightarrow, "\cong"] \arrow[d, "\text{übersetzt in}"'] & \begin{array}{c}\text{CD}\\ cd\end{array} \arrow[dd, "\text{übersetzt in}"] \\
\begin{array}{c}\text{Unwinding Structure}\\ US(td)\end{array} \arrow[d, "\text{übersetzt in}"'] & \\
\begin{array}{c}\text{TPTL}\\ TL(td)\end{array} \arrow[d] & \begin{array}{c}\text{DC}\\ [\![cd]\!]\end{array} \arrow[d] \\
\begin{array}{c}\text{Timed State Sequence}\\ tss\end{array} \arrow[r] & \begin{array}{c}\text{Interpretation}\\ \mathcal{I}_{tss}\end{array}
\end{tikzcd}$$

Abbildung 5.1: Zusammenhang zwischen RTSTD und CD Semantik

Nachdem wir Timed State Sequences mit Interpretationen in Zusammenhang gebracht haben, können wir nun definieren, wann ein Real-Time Symbolic Timing Diagram *td* äquivalent zu einem Constraint Diagram *cd* ist. Dazu betrachten wir die Abbildung 5.1. Nach Übersetzung eines RTSTDs in eine TPTL-Formel lässt sich die Semantik als die Menge von Timed State Sequences *tss* auffassen, die die Formel erfüllen. Analog lässt sich die Semantik eines CDs als Menge von Interpretationen $\mathcal{I}$ auffassen. Eine naheliegende Definition ist nun:

Definition 5.3.3. *Ein RTSTD td und ein CD cd sind äquivalent, in Zeichen $td \cong cd$, wenn für alle Timed State Sequences tss gilt:*

$$tss \models TL(td) \iff \mathcal{I}_{tss} \models_0 [\![cd]\!]$$

5.4 Stotterschritte

Es gibt mehrere Timed State Sequences, die die gleiche zugehörige Interpretation besitzen. Wie bereits am Anfang dieses Kapitels erwähnt, passiert das durch Einfügen bzw. Entfernen von Stotterschritten. Wir wollen in diesem Fall die beiden Timed State Sequences als äquivalent bezeichnen:

Definition 5.4.1. *Zwei Timed State Sequences tss, tss' sind* äquivalent*, in Zeichen $tss \cong tss'$, falls sie die gleiche zugehörige Interpretation besitzen:*

$$tss \cong tss' \iff \mathcal{I}_{tss} = \mathcal{I}_{tss'}$$

Betrachten wir jetzt noch einmal die Definition der Äquivalenz von Diagrammen. Wenn wir zwei Diagramme td und cd mit $td \cong cd$ haben, sowie zwei äquivalente Timed State Sequences $tss \cong tss'$, so gilt:

$$tss \models td \iff \mathcal{I}_{tss} \models [\![cd]\!] \underset{\mathcal{I}_{tss}=\mathcal{I}_{tss'}}{\iff} \mathcal{I}_{tss'} \models [\![cd]\!] \iff tss' \models td$$

Es können also nur solche RTSTDs äquivalent zu einem CD sein, die für äquivalente Timed State Sequences das gleiche Ergebnis liefern. Insbesondere muss so ein RTSTD invariant gegenüber Stotterschritten sein.

Es ist aber nicht jedes RTSTD invariant gegenüber Stotterschritten, wenn nämlich die Aktivierungsbedingung zweier aufeinanderfolgender Ereignisse im RTSTD nicht disjunkt sind. Betrachten wir zum Beispiel das Diagramm in Abbildung 5.2. Dieses Diagramm kann wie folgt interpretieren werden: Zu jedem Zeitpunkt gilt, dass innerhalb von 5 Sekunden die Observable X den Wert 1 annimmt. Die TPTL$^{\mathrm{C}}$-Formel dieses Diagramms lautet:

$$\Box(\mathbf{true} \Rightarrow z.(\mathbf{true\ until}\ (z \leq 5 \wedge X = 1)))$$

oder vereinfacht und die Definition des **until**-Operators eingesetzt:

$$\Box(z.\bigcirc(\mathbf{true}\,\mathcal{U}\ (z \leq 5 \wedge X = 1)))$$

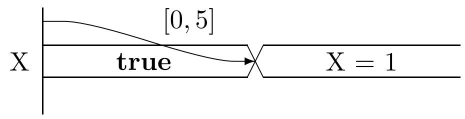


Abbildung 5.2: Beispiel für die Wirkung von Stotterschritten

Wenn nun die Observable X zum Zeitpunkt τ_i den Wert $\sigma_i(X) = 1$ hat und ihn für länger als 5 Sekunden beibehält, so wird das Diagramm nur akzeptiert, wenn in dieser Zeit ein Stotterschritt eingelegt wird. Wenn nämlich $\tau_{i+1} - \tau_i > 5$, so ist bereits nachdem ersten Schritt die Uhr z zu weit gelaufen. Das Diagramm ist also nicht invariant gegenüber Stotterschritten und kommt daher nicht als Übersetzung eines CDs in Frage.

Um solche Probleme zu vermeiden sollte man darauf achten, dass die Aktivierungsbedingungen aufeinanderfolgender Ereignisse immer disjunkt sind. Insbesondere sollte man auf **true**-Phasen generell verzichten. Bei der Übersetzung von CDs in RTSTDs im nächsten Kapitel, werden noch weitere Beispiele dafür gegeben, dass **true**-Phasen nicht verwendet werden sollen.

Kapitel 6

Übersetzung von CDs in RTSTDs

Nachdem wir im letzten Kapitel den Zusammenhang zwischen DC-Observablen und Timed State Sequences definiert haben, können wir nun eine korrekte Übersetzung von CDs in RTSTDs geben. Wir werden uns dabei auf Aktionsdiagramme und die Diagramme der Fallstudie beschränken.

6.1 Übersetzung der Aktionsdiagramme

Das erste Aktionsdiagramm ist das Initialisierungsdiagramm. Dieses Diagramm lässt sich leicht in ein äquivalentes RTSTD übersetzen, wie in Abbildung 6.1 gezeigt. Natürlich benötigt man dafür ein initiales RTSTD.

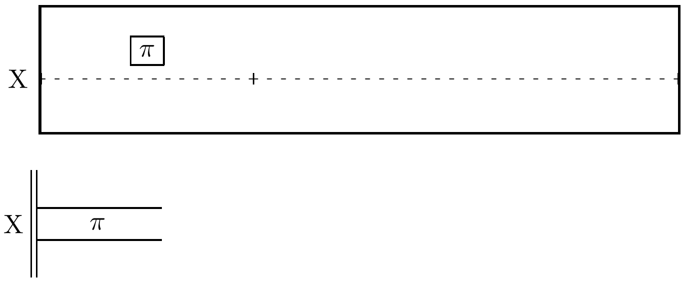


Abbildung 6.1: Übersetzung des Initialisierungsdiagramm

Auch das Sequenzdiagramm lässt sich sehr leicht übersetzen. Die Übersetzung sieht dabei sogar einfacher aus als das ursprüngliche Constraint Diagram, siehe Abbildung 6.2. Um zu verstehen warum dieses RTSTD die gleiche Semantik hat, muss man sich vergegenwärtigen, dass es nur aktiviert wird, wenn die Aktivierungsbedingung des ersten Ereignisses (in diesem Fall π) gilt. Anschließend fordert das Diagramm, dass wenn das erste Diagramm jemals abgearbeitet wird, die Aktivierungsbedingung des zweiten Ereignisses $\pi_1 \vee \cdots \vee \pi_n$ gilt.

Die Übersetzung des Stabilitätsdiagramms (siehe Abbildung 1.4) ist ein wenig komplizierter: Eine direkte Übersetzung des CDs würde nahelegen, dass

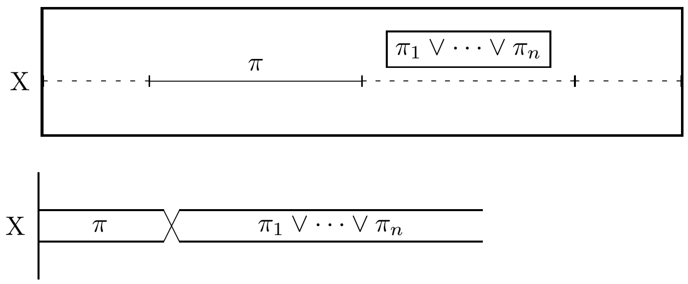
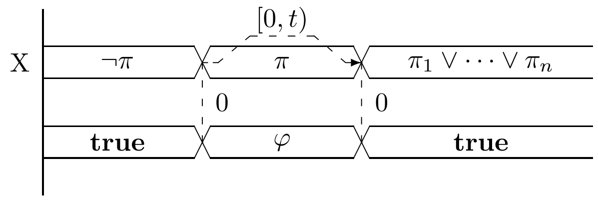


Abbildung 6.2: Übersetzung des Sequenzdiagramms

man einfach **true** als Aktivierungsbedingung des ersten Ereignisses setzt. Mithilfe von weak Constraints wird sichergestellt, dass φ während der gesamten Phase π gilt. Das ganze könnte wie folgt aussehen.



Das funktioniert allerdings nicht, aus einem ziemlich subtilen Grund: Der im Abschnitt 4.2 erwähnte zeitbehaftete Automat (die Unwinding Structure) hätte die Wahl, wann genau er das erste Ereignis der zweiten Waveform (Wechsel von **true** nach φ) durchführt. Weil **true** immer gilt, kann er diesen Wechsel in jedem Fall über den Beginn der Phase π hinausschieben und dadurch den weak Constraint verletzen. Das Diagramm wird dann akzeptiert, selbst wenn die Annahmen erfüllt sind und die Zusicherung nicht getestet wurde. Das gleiche gilt auch für den Wechsel von φ nach **true**, falls φ auch weiterhin erfüllt ist.

Daher sollte man auf mit **true** beschriftete Phasen verzichten. Um nun das Stabilitätsdiagramm zu übersetzen benötigen wir nun zwei Diagramme, siehe Abbildung 6.3. Eines, falls bereits vor der π-Phase φ galt, und eines, falls der Wechsel von $\neg\varphi$ auf φ gleichzeitig mit dem Wechsel von $\neg\pi$ nach π stattfindet.

Bei der Übersetzung fällt auf, dass auf die φ-Phase eine $\neg\varphi$-Phase angefügt wurde und die φ-Phase nach rechts verlängert wurde, d. h. man erlaubt dass der Wechsel nach $\neg\varphi$ auch später stattfinden darf. Mithilfe des weak Constraints $[0, t)$ wird dafür gesorgt, dass wenn π für t Sekunden stabil bleibt, das Diagramm auf jeden Fall akzeptiert wird. Die einzige Zusicherung des Diagramms ist, dass auf die π-Phase eine $\pi_1 \vee \cdots \vee \pi_n$-Phase folgt, sofern die Voraussetzungen alle erfüllt sind.

In Abbildung 6.4 ist das Synchronisationsdiagramm mit seiner Übersetzung nach RTSTD zu sehen. Wieder wird an die φ-Phase eine $\neg\varphi$-Phase angehängt und erlaubt, dass der Wechsel nach $\neg\varphi$ erst später stattfindet. Durch den weak Constraint wird sichergestellt, dass wenn der Wechsel von φ nach $\neg\varphi$ zu früh stattfindet, d. h. bevor t Sekunden verstrichen sind, dieses Diagramm akzeptierend verlassen wird. Andernfalls fordert das Diagramm, dass der Zustand π innerhalb von t Sekunden verlassen wird.

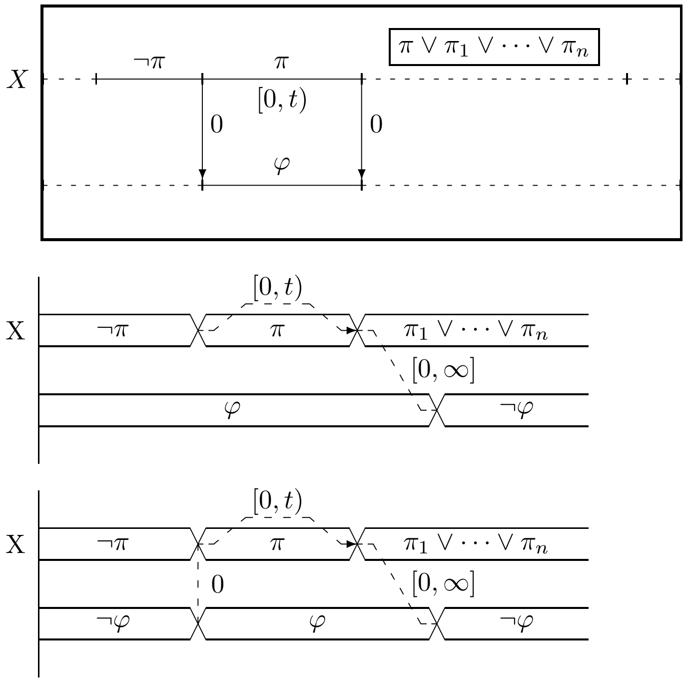


Abbildung 6.3: Übersetzung des Stabilitätsdiagramms

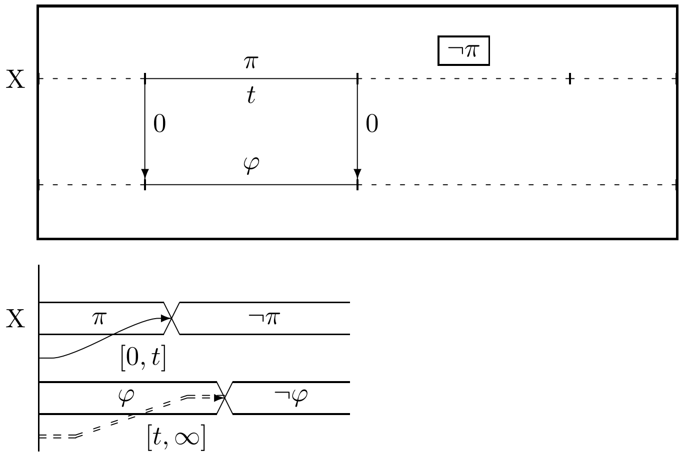
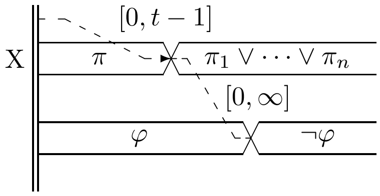


Abbildung 6.4: Übersetzung der Synchronisation

Als letztes wenden wir uns der initialen Stabilität zu. Diese lässt sich nicht direkt übersetzen. Es dürfte klar sein, dass eine Übersetzung ein initiales RTSTD sein muss. Eine mögliche Übersetzung wäre etwa folgendes Diagramm:



Das Problem ist allerdings, dass initiale RTSTDs fordern, dass die Aktivierungsbedingung des ersten Ereignisses jeder Waveform, in diesem Fall π und φ initial gelten. Das Constraint Diagram fordert dieses jedoch nicht. Zum Glück treten initiale Stabilitätsdiagramme im Allgemeinen zusammen mit den entsprechenden Initialisierungsdiagrammen auf. Obiges Diagramm ist eine korrekte Übersetzung der drei Diagramme in Abbildung 6.1.

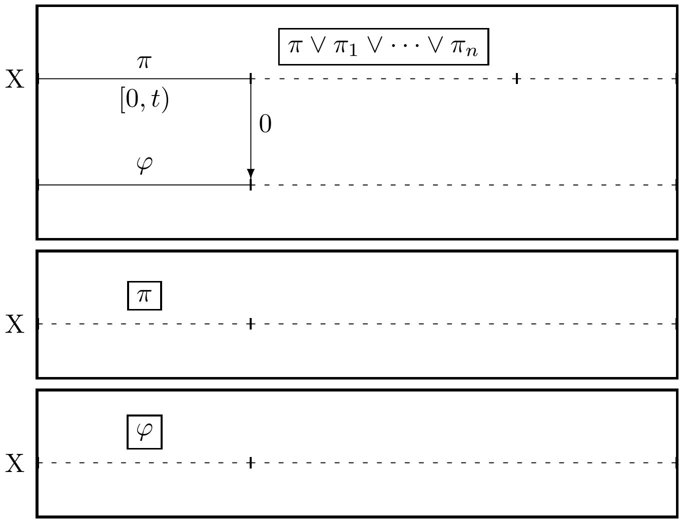


Abbildung 6.5: Initiales Stabilitätsdiagramm mit zugehörigen Initialisierungsdiagrammen

6.2 Korrektheit der Übersetzung

In diesem Abschnitt zeigen wir die Korrektheit der Übersetzung am Beispiel der Synchronisation:

Satz 6.2.1. *Das Constraint Diagramm und das Real-Time Symbolic Timing Diagram aus Abbildung 6.4 (die Synchronisationsdiagramme) sind äquivalent gemäß Definition 5.3.3 auf Seite 47.*

Um diesen Satz zu beweisen sehen wir uns zunächst die DC bzw. TPTL Semantik der beiden Diagramme an. In [Die97] wurde bereits gezeigt, dass das

Synchronisationsdiagramm semantisch äquivalent zu der DC-Formel

$$\lceil \pi \wedge \varphi \rceil \xrightarrow{t} \lceil \neg\pi \rceil$$

ist. Laut Definition gilt:

$$\lceil \pi \wedge \varphi \rceil \xrightarrow{t} \lceil \pi \rceil \iff \Box\neg((\lceil \pi \wedge \varphi \rceil \wedge \ell = t); \lceil \pi \rceil)$$

Im Folgenden werden wir daher nur noch die rechte Formel betrachten, die wir mit DC_{synch} bezeichnen.

In Abschnitt 4.2 wurde erwähnt, dass die Semantik von Real-Time Symbolic Timing Diagrams über die Unwinding Structure erklärt wird. Die zu dem Diagramm in Abbildung 6.4 gehörige Unwinding Structure US_{synch} ist in Abbildung 6.6 gezeigt. Sie wurde getreu der in [Fey96] angegebenen Vorschrift erzeugt; es wurden lediglich einige Transitionen zusammengefasst und triviale logische Vereinfachungen vorgenommen, wie zum Beispiel $z \leq t \wedge z < t$ zu $z < t$. Die $TPTL^C$-Formel, die die Semantik des RTSTDs erklärt ist nun:

$$TL_{synch} := \Box(\pi \wedge \varphi \Rightarrow z.TL(US_{synch}))$$

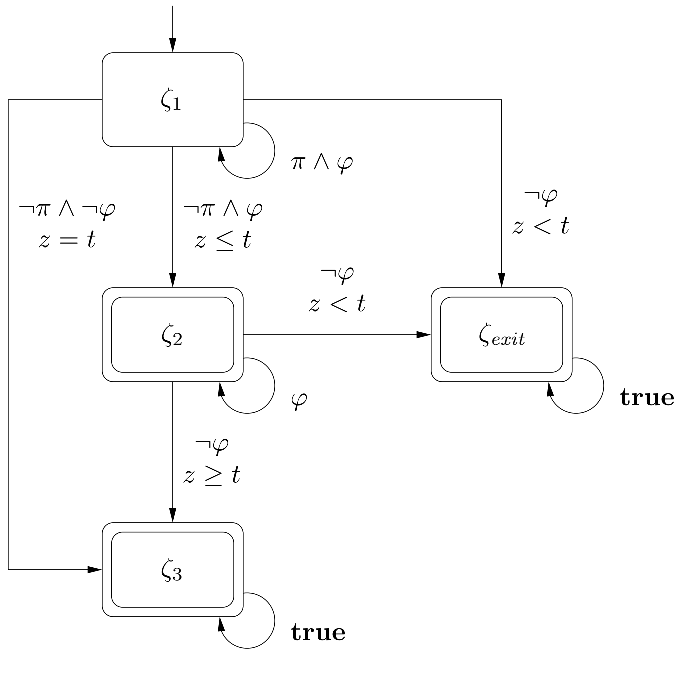


Abbildung 6.6: Unwinding Structure US_{synch}

Für den obigen Satz genügt es, folgendes Lemma zu beweisen:

Lemma 6.2.2. *Seien TL_{synch}, DC_{synch} wie oben. Dann gilt für alle Timed State Sequences $tss = (\sigma, \tau)$ und den zugehörigen Interpretationen $\mathcal{I}_{tss}$:*

$$tss \models TL_{synch} \iff \mathcal{I}_{tss} \models_0 DC_{synch}$$

Beweis. „$\Rightarrow$“: Es gilt $tss \models TL_{synch}$. Wir zeigen:

$$\forall [b,e] \subseteq \mathbf{Time} \cdot \mathcal{I}_{tss}, [b,e] \not\models (\ulcorner \pi \wedge \varphi \urcorner) \wedge \ell = t); \ulcorner \pi \urcorner$$

Für alle Intervalle $[b,e]$ gibt es ein $i \in \mathbb{N}$ mit $\tau_i \leq b < \tau_{i+1}$ wegen der *Non-Zeno*-Eigenschaft der Zeitfolge τ. Also genügt es zu zeigen, dass

$$\forall i \in \mathbb{N} \cdot \forall b \in [\tau_i, \tau_{i+1}) \cdot \forall e > b \cdot \mathcal{I}_{tss}, [b,e] \not\models (\ulcorner \pi \wedge \varphi \urcorner) \wedge \ell = t); \ulcorner \pi \urcorner \tag{6.1}$$

Sei nun $i \in \mathbb{N}$.

Fall 1: $\sigma_i \models \neg(\pi \wedge \varphi)$, dann gilt wegen $b < \tau_{i+1}$ und Definition von $\mathcal{I}_{tss}$:

$$\mathcal{I}_{tss}, [b, \tau_{i+1}] \models \ulcorner \neg(\pi \wedge \varphi) \urcorner$$

Daher gilt für alle $e > b$: $\mathcal{I}_{tss}, [b,e] \not\models \ulcorner \pi \wedge \varphi \urcorner; \mathbf{true}$, also erst recht (6.1).

Fall 2: $\sigma_i \models \pi \wedge \varphi$. Nach Voraussetzung muss der Automat US_{synch} zum Zeitpunkt τ_i startend tss akzeptieren, insbesondere den ersten Zustand ζ_1 verlassen. Dies geschehe zum Zeitpunkt τ_j, $j > i$. Zum Startzeitpunkt τ_i wurde die Uhr z in der Formel TL_{synch} zurückgesetzt und hat daher zum Zeitpunkt τ_j den Wert $\tau_j - \tau_i$. Wir unterscheiden nun folgende Fälle:

- wechselt US_{synch} in den Zustand ζ_{exit}, so gilt $\tau_j - b \leq \tau_j - \tau_i < t$ und $\sigma_j \models \neg\varphi$. Es folgt (mit Chop zum Zeitpunkt τ_j):

  $$\mathcal{I}_{tss}, [b, \tau_{j+1}] \models \ell < t; \ulcorner \neg\varphi \urcorner$$

  Also gilt für alle $e > b$: $\mathcal{I}_{tss}, [b,e] \not\models ((\ulcorner \varphi \urcorner \wedge \ell = t); \mathbf{true})$, und damit folgt (6.1)
- wechselt US_{synch} in den Zustand ζ_2, so gilt: $\tau_j - \tau_i \leq t$ und $\sigma_j \models \neg\pi$. Analog zum obigen Fall folgt:

  $$\mathcal{I}_{tss}, [b, \tau_{j+1}] \models \ell \leq t; \ulcorner \neg\pi \urcorner$$

  Damit gilt für alle $e > b$: $\mathcal{I}_{tss}, [b,e] \not\models \ulcorner \pi \urcorner \wedge \ell > t$, und das ist äquivalent zu $\mathcal{I}_{tss}, [b,e] \not\models (\ulcorner \pi \urcorner \wedge \ell = t); \ulcorner \pi \urcorner$. Wieder folgt (6.1)
- wechselt US_{synch} in den Zustand ζ_3, gilt $\tau_j - \tau_i = t$ und $\sigma_j \models \neg\pi \wedge \neg\varphi$. Wie im obigen Fall folgt nun (6.1).

„$\Leftarrow$“: Es gelte nun $\mathcal{I}_{tss} \models \Box\neg((\ulcorner \pi \wedge \varphi \urcorner \wedge \ell = t); \ulcorner \pi \urcorner)$. Zu zeigen ist

$$tss \models \Box(\pi \wedge \varphi \Rightarrow z.\, TL(US_{synch}))$$

Sei $i \in \mathbb{N}$, $\sigma_i \models (\pi \wedge \varphi)$. Wir zeigen: US_{synch} startend zum Zeitpunkt τ_i akzeptiert. Dazu genügt es zu zeigen, dass der Automat den Startzustand verlässt, denn aus den übrigen Zuständen sind nur akzeptierende Zustände erreichbar und es ist immer eine Transition möglich.

Würde nun für alle $j > i$, $\sigma_j \models \pi \wedge \varphi$ gelten, so würde auch

$$\mathcal{I}_{tss}, [\tau_i, \tau_i + t + 1] \models (\ulcorner \pi \wedge \varphi \urcorner \wedge \ell = t); \ulcorner \pi \urcorner$$

gelten, im Widerspruch zur Voraussetzung. Also gibt es ein kleinstes $j > i$ mit $\sigma_j \not\models \pi \wedge \varphi$. Bis zum Schritt j kann der Automat im Anfangszustand verbleiben. Nun kommen noch folgende Fälle in Betracht:

- $\tau_j - \tau_i > t$: Dann gilt mit Chop zum Zeitpunkt $\tau_i + t$ im Widerspruch zur Voraussetzung: $\mathcal{I}_{tss}, [\tau_i, \tau_j] \models (\ulcorner \pi \wedge \varphi \urcorner \wedge \ell = t); \ulcorner \pi \urcorner$.

- $\sigma_j \models \pi \wedge \neg\varphi$ und $\tau_j - \tau_i = t$: Mit Chop bei τ_j gilt dann erneut im Widerspruch zur Voraussetzung: $\mathcal{I}_{tss}, [\tau_i, \tau_{j+1}] \models (\ulcorner \pi \wedge \varphi \urcorner \wedge \ell = t); \ulcorner \pi \urcorner$.

- $\sigma_j \models \neg\pi \wedge \neg\varphi$ und $\tau_j - \tau_i = t$: US_{synch} wechselt nach ζ_3.

- $\sigma_j \models \neg\varphi$ und $\tau_j - \tau_i < t$: US_{synch} wechselt nach ζ_{exit}.

- $\sigma_j \models \neg\pi \wedge \varphi$ und $\tau_j - \tau_i \leq t$: US_{synch} wechselt nach ζ_2. $\square$

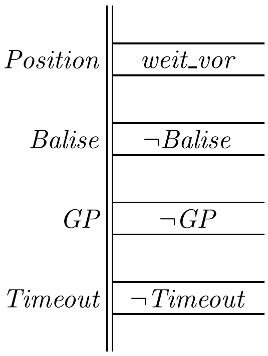
Man beachte, dass im obigen Beweis an mehreren Stellen die strenge Monotonie der Zeitfolge τ benötigt wird, vor allem um aus $\sigma_i \models \pi$ zu schließen, dass $\mathcal{I}_{tss}, [\tau_i, \tau_{i+1}] \models \ulcorner \pi \urcorner$ gilt.

6.3 Übersetzung der Diagramme der Fallstudie

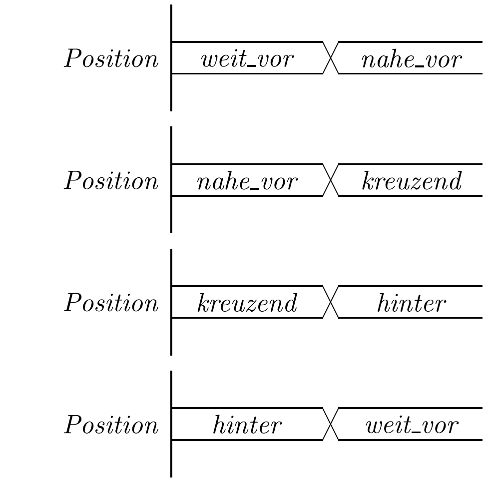
Nun wollen wir die im Kapitel 2 aufgeführten Diagramme übersetzen. Die vorgestellte Übersetzung der Aktionsdiagramme liefert uns auf einen Schlag RTSTDs für beinahe alle CDs.

6.3.1 Anwenden der Übersetzung der Aktionsdiagramme

Die Initialisierung ist das einfachste Aktionsdiagramm. Das Diagramm aus Abbildung 2.3 kann man problemlos in ein RTSTD übersetzen, auch wenn gleich mehrere Initialisierungen zusammengefasst wurden:



Die Sequenzdiagramme aus Abbildung 2.4 auf Seite 14 lassen sich streng nach der Regel aus Abschnitt 6.1 übersetzen.



Die Diagramme in Abbildung 2.5 (Setzen des Gefahrenpunktes) sind Synchronisationsdiagramme, die sich ebenfalls streng nach der Regel übersetzenlassen, siehe Abbildung 6.7.

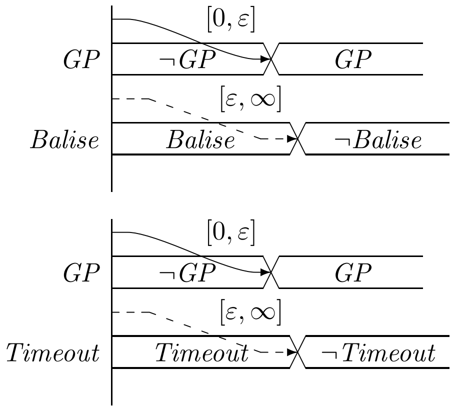


Abbildung 6.7: Übersetzung von Setzen des Gefahrenpunktes

Bei den Stabilitätsdiagrammen in Abbildung 2.6 (Stabilität von $\neg GP$) gibt es eine Besonderheit: Dort werden über drei verschiedene Observablen Aussagen gemacht. Der Zustandsausdruck φ des allgemeinen Stabilitätsdiagramm darf auch über mehrere Observablen Aussagen machen. Bei der Übersetzung wurde aber φ wie ein Zustandsausdruck über eine Observable im Diagramm dargestellt. Wir lösen das Problem, in dem wir die Observablen *Balise* und *Timeout* zusammenfassen; in RTSTD darf nämlich eine Zeile auch über mehrere Observablen Aussagen machen. Außerdem haben wir den Fall, dass keine weiteren Folgezustände $\pi_1 \vee \cdots \vee \pi_n$ erlaubt sind, d. h. $n = 0$. In diesem Fall ist $\pi_1 \vee \cdots \vee \pi_n \iff \mathbf{false}$. Auch die Zeitbeschränkung fällt komplett weg ($t = \infty$). Das Stabilitätsdiagramm wird wie erwähnt in zwei Diagramme übersetzt, siehe Abbildung 6.8.

Das zweite Diagramm in Abbildung 2.6 ist ein initiales Stabilitätsdiagramm. Weil die Initialisierungen der Observablen stimmen, lässt sich dieses Diagramm

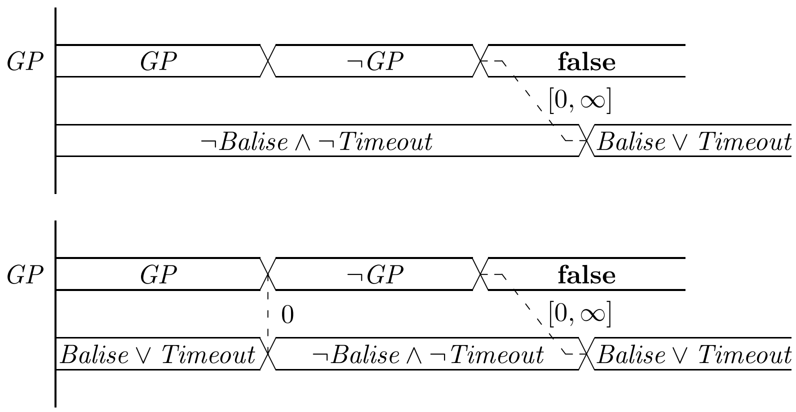


Abbildung 6.8: Übersetzung von Stabilität von $\neg GP$ (Abbildung 2.6)

wie im Abschnitt 6.1 beschrieben übersetzen, siehe Abbildung 6.9.

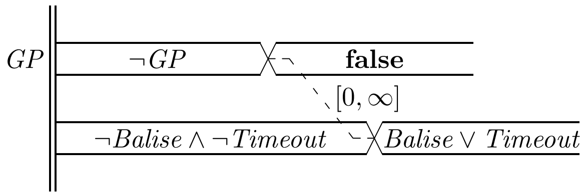


Abbildung 6.9: Übersetzung von initialerStabilität von $\neg GP$ (Abbildung 2.6)

Beim Diagramm in Abbildung 2.7 (Funktion der Bremssteuerung) handelt es sich wieder um ein Stabilitätsdiagramm. Es besagt, dass die Position *nahe_vor* stabil bleibt, wenn der Gefahrenpunkt gesetzt ist. Die Übersetzung läuft wieder analog ab, siehe Abbildung 6.10. Es gibt wieder keine Zeitbeschränkung, oder alternative Folgezustände.

Bei den Diagrammen in Abbildung 2.8 (Löschen des Gefahrenpunktes) und Abbildung 2.9 (Der Stellbefehl) handelt es sich wieder um Stabilitäts- und Synchronisationsdiagramme, die sich genauso wie die anderen Diagramme übersetzen lassen. Das Diagramm in Abbildung 2.10 ist kein Aktionsdiagramm und wird daher im nächsten Abschnitt behandelt. Das Diagramm in Abbildung 2.11 ist ein sehr einfaches Stabilitätsdiagramm. Weil hier $\varphi \iff \textbf{true}$ ist, brauchen wir bei der Übersetzung in RTSTD nur ein Diagramm, siehe Abbildung 6.11.

Die letzten Diagramme in Abbildung 2.13 (Balise korrekt verlegt) sind keine Aktionsdiagramme und werden daher ebenfalls erst im nächsten Abschnitt betrachtet.

6.3.2 Übersetzung der verbleibenden Diagramme

Das Diagramm „Setzen des Timeouts“, in Abbildung 6.12 nochmals dargestellt, wollen wir jetzt per Hand nach RTSTD übersetzen. Es beinhaltet eine besondere Schwierigkeit: In diesem Diagramm ist der Verlauf der Variable Timeout beliebig, bis er gesetzt wird. Im Gegensatz zum Stabilitätsdiagramm kann hier auch

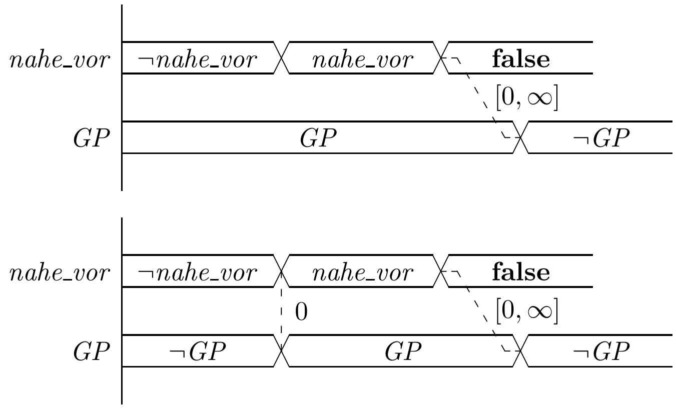


Abbildung 6.10: Übersetzung von Abbildung 2.7

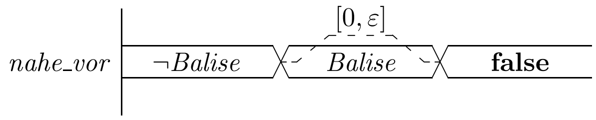


Abbildung 6.11: Übersetzung von Abbildung 2.11

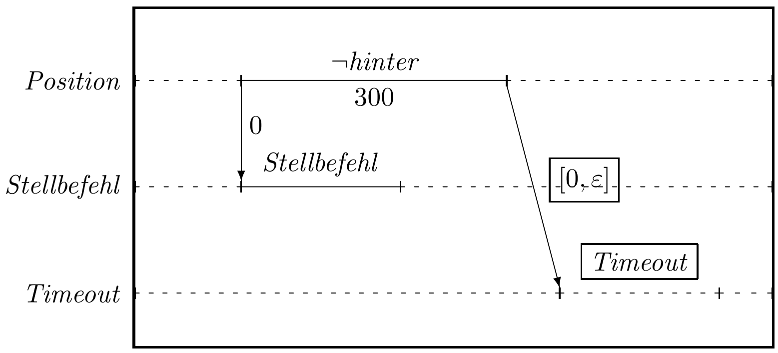
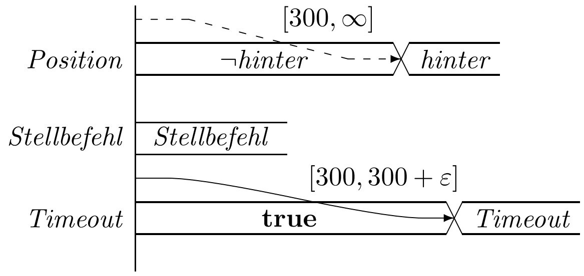


Abbildung 6.12: Setzen des Timeouts

keine Fallunterscheidung unternommen werden, da *Timeout* beliebig oft seinen Wert wechseln darf. Wenn man aber im RTSTD eine **true**-Phase einsetzt, ergeben sich wieder Schwierigkeiten. Betrachten wir dafür folgendes Diagramm:



In diesem Fall scheint es auf dem ersten Blick möglich eine **true**-Phase zu benutzen, weil der Wechsel nach *Timeout* eine Zusicherung ist. Der Automat wird also das Diagramm genau dann akzeptieren, wenn der Wechsel nach *Timeout* innerhalb des angegebenen Zeitintervalls geschehen *kann*, d. h. wenn *Timeout* wie gewünscht zum richtigen Zeitpunkt gesetzt ist.

Leider funktioniert es aber doch nicht. Betrachten wir dazu folgenden Verlauf der Observablen in Abbildung 6.13. Dieser Verlauf wird vom Constraint Diagramm aus Abbildung 2.10 akzeptiert. Das Timeout ist wie gewünscht 300 Sekunden nach Absenden des Stellbefehls gesetzt, sogar schon ein wenig früher.

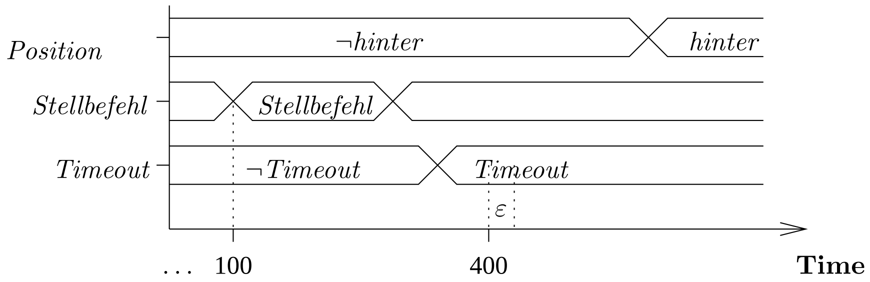


Abbildung 6.13: Verlauf der Observablen

Die durch den obigen Verlauf der Observablen gegebene Interpretationist ist zugehörig zur folgenden Timed State Sequence *tss*:

$$
tss = \left(\ldots, \begin{matrix} \left\{ \begin{matrix} Position \neq hinter, \\ Stellbefehl = \mathbf{true}, \\ Timeout = \mathbf{false}, \end{matrix} \right\}, \\ 100, \end{matrix} \begin{matrix} \left\{ \begin{matrix} Position \neq hinter, \\ Stellbefehl = \mathbf{false}, \\ Timeout = \mathbf{false}, \end{matrix} \right\}, \\ 300, \end{matrix} \right.
$$
$$
\left. \begin{matrix} \left\{ \begin{matrix} Position \neq hinter, \\ Stellbefehl = \mathbf{false}, \\ Timeout = \mathbf{true}, \end{matrix} \right\}, \\ 350, \end{matrix} \begin{matrix} \left\{ \begin{matrix} Position = hinter, \\ Stellbefehl = \mathbf{false}, \\ Timeout = \mathbf{true}, \end{matrix} \right\}, \\ 600 \end{matrix} \ldots \right)
$$

Dieser Verlauf wird von dem auf der letzten Seite gegebenen RTSTD jedoch nicht akzeptiert, wenn der Automat zum Zeitpunkt 100 startet: Dort wird nämlich verlangt, dass nach 300 bis $300+\varepsilon$ Sekunden ein *Wechsel* von **true** nach

Timeout stattfindet. In diesem Zeitraum findet jedoch überhaupt kein Ereignis statt. Also wird der strong Constraint verletzt und *tss* nicht akzeptiert.

Statt nun eine direkte Übersetzung des Timeout-Diagramms anzugeben, werden wir hier das Diagramm *verfeinern*. Das heißt wir werden ein RTSTD angeben, das nur Verläufe akzeptiert, die auch von dem CD akzeptiert werden. Es gilt aber nicht die andere Richtung. Dabei überlegen wir uns genauer, wie der Verlauf von *Timeout* aussehen kann.

Wir werden jetzt verlangen, dass das Timeout *spätestens* 300 Sekunden (plus Reaktionszeit ε) nach Absenden des Stellbefehls gesetzt wird, sofern der Bahnübergang nicht passiert wurde, und aktiviert bleibt bis $Position = hinter$ ist. Das zugehörige RTSTD ist in Abbildung 6.14 zu sehen.

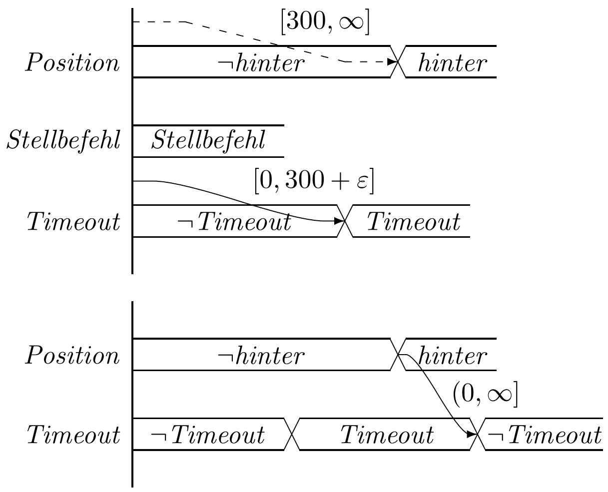


Abbildung 6.14: Setzen des Timeouts als RTSTD

Ähnlich wie schon vorher haben wir im ersten Diagramm die Phase $\neg hinter$ verlängert und die Phase *hinter* angeschlossen. Wir fordern, dass $Position = hinter$ frühestens nach 300 Sekunden gelten darf, andernfalls wird dieses Diagramm verlassen und Timeout braucht nicht gesetzt zu werden. Ist der weak Constraint erfüllt sichert uns der strong Constraint zu, dass innerhalb von $300 + \varepsilon$ Sekunden *Timeout* gesetzt wird. Das zweite Diagramm fordert, dass wenn *Timeout* gesetzt ist, es so lange gesetzt bleibt, bis der Zug den Bahnübergang passiert hat ($Position = hinter$). Beide Diagramme zusammen garantieren das von dem Constraint Diagramm in Abbildung 6.14 geforderte Verhalten.

Die beiden Diagramme aus Abbildung 6.15 (Balise korrekt verlegt) sind ebenfalls keine Aktionsdiagramme. Sie haben zwar Ähnlichkeit mit Stabilitätsdiagrammen, aber wegen des Intervalls nach der Phase $\neg Balise$ sind sie es nicht. Ihre Übersetzung ist in Abbildung 6.16 zu sehen. Das zweite Diagramm wird wie ein Stabilitätsdiagramm in zwei Diagramme übersetzt.

Betrachten wir das letzte Diagramm. Dort wird zunächst gefordert, dass initial $Position = weit\_vor$ und $\neg Balise$ gilt, wie es auch von den initialen Constraint Diagrams der Fallstudie (Abbildung 2.3) gefordert wurde. Wenn nun irgendwann $Position \neq weit\_vor$ gilt, so wird der Automat das Verhalten nicht akzeptieren (wegen der **false**-Phase), es sei denn, es wurde vorher der weak

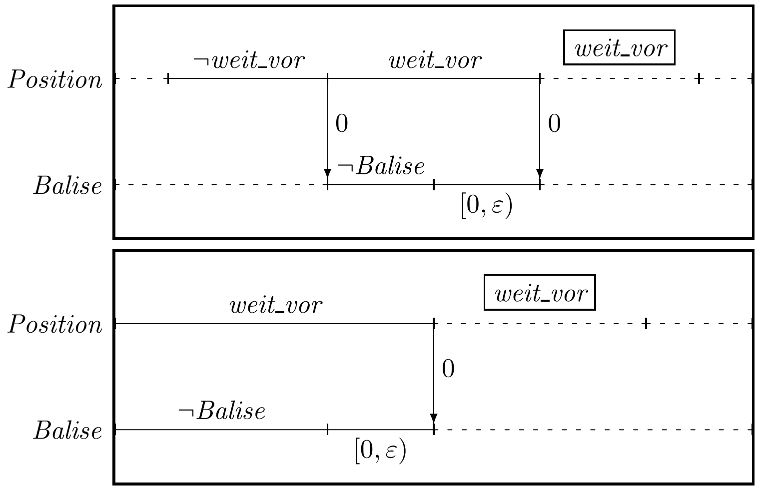


Abbildung 6.15: Balise ist korrekt verlegt (Abbildung 2.13)

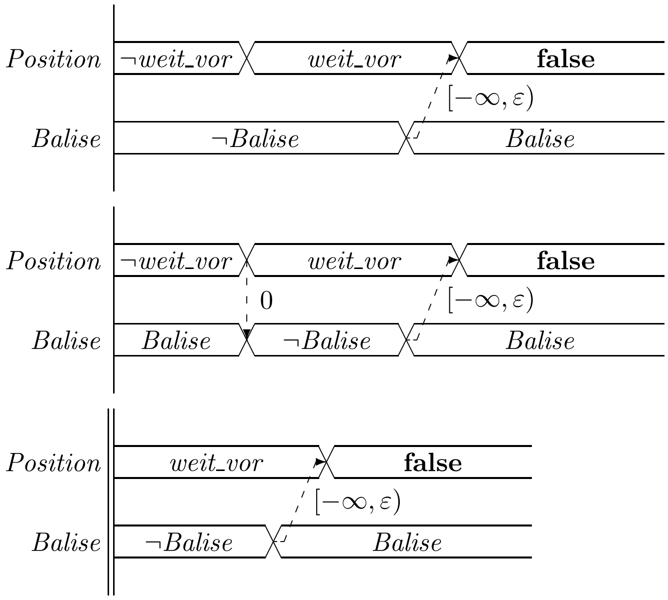


Abbildung 6.16: Balise korrekt verlegt als RTSTD

Constraint verletzt. Der weak Constraint wird genau dann verletzt, wenn der Wechsel von *weit_vor* frühestens ε Sekunden nach dem Wechsel von $\neg Balise$ nach *Balise* stattfindet. Genau in diesem Fall akzeptiert aber auch das zweite Constraint Diagram in Abbildung 2.13 den Verlauf.

Die Übersetzung des ersten Diagramms in Abbildung 6.15 ist analog.

6.4 Grenzen der Übersetzbarkeit

In dieser Arbeit wurde nur eine Teilklasse von CDs in RTSTDs übersetzt. Es wäre natürlich schöner, einen Algorithmus zu finden, der jedes beliebige Constraint Diagram übersetzen kann. Doch schon die Schwierigkeiten, die im letzten Abschnitt auftraten, zeigen, dass das ein äußerst schwieriges Unterfangen ist. Einige Constraint Diagrams müssen in mehrere RTSTDs übersetzt werden, einige lassen sich überhaupt nicht übersetzen (zum Beispiel die initiale Stabilität). Die Schwierigkeiten liegen in mehreren Punkten:

- In CDs werden Phasen häufig mit **true** beschriftet. Das führt bei RTSTDs jedoch fast zwangsläufig zu Problemen. Dort sollten aufeinanderfolgende Phasen sich gegenseitig ausschließen.

  Dieses Problem lässt sich teilweise dadurch lösen, dass man an eine Phase, auf die im CD eine **true**-Phase folgt, im RTSTD eine Phase mit negierter Bedingung anhängt und erlaubt, dass der Übergang in diese Phase später passiert. So wurde es zum Beispiel bei der φ-Phase des Stabilitätsdiagramms gelöst.

- Obiger Punkt führt noch zu einer weiteren Schwierigkeit. Während bei Constraint Diagrams es möglich ist, über den Wert einer Observablen zum Anfang der Betrachtung keine Aussagen zu machen, ist dies bei RTSTD nur über eine **true**-Phase möglich, die unter allen Umständen vermieden werden sollte.

  Dieses Problem lässt sich umgehen, in dem man mehrere Diagramme für die verschiedenen Verläufe angibt, wie zum Beispiel beim Stabilitätsdiagramm. Das geht allerdings nur, wenn es nur endlich viele verschiedene Verläufe gibt. Bei dem Timeout-Diagramm im letzten Abschnitt ist das nicht möglich, denn die Observable *Timeout* kann während der **true**-Phase beliebig oft ihren Wert ändern. Das ist der Grund, warum wir den erlaubten Verlauf von *Timeout* für die Übersetzung weiter eingeschränkt haben.

- In initialen Real-Time Symbolic Timing Diagrams, lassen sich keine Voraussetzungen über den initialen Zustand unterbringen. Es wird immer implizit *gefordert*, dass die erste Phase wie beschrieben aussieht. Das ist der Grund warum sich die initiale Stabilität alleine nicht übersetzen lässt.

- Bei CDs können Voraussetzungen und Zusicherungen an beliebigen Stellen im Diagramm stehen. Eine Zusicherung kann sogar ihrer Voraussetzung zeitlich vorausgehen. RTSTDs dagegen werden von links nach rechts abgearbeitet. Steht eine Voraussetzung zeitlich hinter der Zusicherung, so wird der Automat das Diagramm nicht akzeptieren, wenn die Zusicherung verletzt ist, obwohl er gar nicht die Voraussetzung geprüft hat. Bei den Aktionsdiagrammen und den Diagrammen der Fallstudie waren Voraussetzung und Zusicherung allerdings immer in der richtigen zeitlichen Reihenfolge.

Dieses Problem lässt sich durch Umformulieren der Aussage ermöglichen. Wenn wir zum Beispiel die Aussage haben: „Wann immer B gilt, muß vorher A gegolten haben“, so können wir sie wie folgt umformulieren: „Solange A nie gilt, kann B auch nicht gelten.“

Kapitel 7

Zusammenfassung

In dieser Arbeit wurden zwei graphische Spezifikationssprachen betrachtet und in Zusammenhang gebracht. Die Schwierigkeiten lagen dabei vor allem im unterschiedlichen semantischen Bereich. Daher wurde im Kapitel 5 die semantischen Bereiche der beiden zugrundeliegenden Formalismen TPTL und DC in Zusammenhang gebracht. Anschliessend konnte definiert werden, wann ein CD und ein RTSTD äquivalent sind. Die Definition der Äquivalenz ermöglichte uns in Kapitel 6 die Übersetzung einer Teilklasse von CDs, den Aktionsdiagrammen. Lediglich bei der Übersetzung der initialen Stabilität musste eine weitere Annahme über den initialen Zustand einiger Observablen gemacht werden, doch wie in der Fallstudie sind diese meistens erfüllt.

Es stellte sich jedoch auch heraus, dass sich ein Diagramm der Fallstudie überhaupt nicht übersetzen ließ und nur für die Aktionsdiagramme konnte eine Übersetzungsvorschrift gefunden werden. Allerdings stellen die Aktionsdiagramme eine weitreichend verwendbare Teilklasse von Constraint Diagrams dar.

Index

Literaturverzeichnis

[AH94] ALUR, RAJEEV and THOMAS HENZINGER: *A really temporal logic.* Journal of the ACM, 41(1):181–204, 1994.

[DFMV98] DIERKS, H., A. FEHNKER, A. MADER, and F.W. VAANDRAGER: *Operational and logical semantics for polling real-time systems.* Technical Report CSI-R9813, Katholieke Universiteit Nijmegen, April 1998.

[Die96] DIETZ, CHERYL: *Graphical formalization of real-time requirements.* In BENGT JONSON and JOACHIM PARROW (editors): *Formal Techniques in Real-Time and Fault-Tolerant Systems*, number 1135 in *Lecture Notes in Computer Science*, pages 366–384. Springer Verlag, 1996.

[Die97] DIETZ, CHERYL: *Action diagrams.* In MARANZANA, MATHIEU (editor): *Twenty Second IFAC/IFIP Workshop on Real-Time Programming*, pages 51–56. IFAC Preprints, 1997.

[Die98] DIETZ, CHERYL: *Advances in graphical yet formal real-time system development.* Electronic Summer School of Logic, Language and Information, 1998.

[Fey96] FEYERABEND, KONRAD: *Real time symbolic timing diagrams.* Technical report, Carl von Ossietzky Universität Oldenburg, Germany, November 1996.

[FJ] FEYERABEND, KONRAD and BERNHARD JOSKO: *A visual formalism for real time requirement specifications.* Technical report, Carl von Ossietzky Universität Oldenburg, Germany.

[HZ97] HANSEN, M.R. and ZHOU CHAOCHEN: *Duration calculus: logical foundations.* Formal Aspects of Computing. 1997.

[Jan97] JANSEN, LARS: *Spezifikation der Steuerung und Sicherung von Fahrzeug und Fahrweg für zustandsvariable Fahrwegelemente beim funkbasierten Fahrbetrieb im spurgebundenen Verkehr.* http://www.ifra.ing.tu-bs.de/~m33/spezi/, 1997.

[LRL98] LIU ZHIMING, ANDERS P. RAVN, and LI XIAOSHAN: *Unifying proof methodologies of DC and LTL.* In *Duration Calculus Workshop*, Electronic Summer School of Logic, Language and Information, pages 99–109, 1998.

[MP93] MANNA, ZOHAR and AMIR PNUELI: *Verifying hybrid sytems.* In GROSSMAN, ROBERT L., ANIL NERODE, ANDERS P. RAVN, and HANS RISCHEL (editors): *Hybrid Systems*, number 736 in *Lecture Notes in Computer Science*, pages 4–35, 1993.

[SD93] SCHLÖR, RAINER and WERNER DAMM: *Specification and verification of system-level hardware designs using timing diagrams.* The European Conference on Design Automation with the European Event in ASIC Design, pages 518–524, 1993.

[ZHR91] ZHOU CHAOCHEN, C.A.R. HOARE, and A.P. RAVN: *A calculus of durations.* Information Processing Letters. 1991.

Ich versichere, die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Hilfsmittel verfaßt zu haben.

Oldenburg, den 14. Juni 1999

Jochen Hoenicke