ALBERT-LUDWIGS-UNIVERSITÄT
FREIBURG
INSTITUT FÜR INFORMATIK

Lehrstuhl für Rechnerarchitektur
Prof. Dr. Bernd Becker

Bounded Model Checking für unvollständige Netzwerke von Zeitautomaten

Diplomarbeit

Karina Gitina

18. Oktober 2010

Erstgutachter:	Prof. Dr. Bernd Becker
Zweitgutachter:	Prof. Dr. Christoph Scholl
Betreuer:	Dipl.-Inf. Christian Miller

Erklärung

Hiermit erkläre ich, daß ich die vorliegende Arbeit selbständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt und alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder unveröffentlichten Schriften entnommen wurden, als solche kenntlich gemacht habe. Außerdem erkläre ich, daß diese Diplomarbeit nicht, auch nicht auszugsweise, bereits für eine andere Prüfung angefertigt wurde.

Karina Gitina
Freiburg, 18. Oktober 2010

Inhaltsverzeichnis

1 Einleitung

Technik begegnet uns jeden Tag fast überall. Sie hält immer mehr Einzug in unser Leben, immer mehr sind wir auf sie angewiesen. Vor allem eingebettete Systeme begleiten uns in vielen Lebensbereichen, manchmal völlig unbemerkt. Wir stellen durch unsere Bedürfnisse sehr hohe Anforderungen an derartige Systeme: Sie müssen unter Echtzeitbedingungen korrekt ihre Aufgabe erfüllen. Die Verletzung solcher Echtzeitanforderungen kann störende, aber tolerierbare Auswirkungen zur Folge haben, wie z. B. das Ruckeln eines Videos beim Abspielen. In sicherheitskritischen Bereichen allerdings müssen wir uns unbedingt auf Echtzeitsysteme verlassen können; ansonsten können auch Menschenleben in Gefahr geraten mit fatalen Folgen: Ein Airbag muss nach einem Aufprall innerhalb einer gewissen Zeitspanne aufgeblasen werden. Bei einem Störfall in einer chemischen Anlage müssen die Sicherungssysteme schnell genug reagieren. Ein Herzschrittmacher, der den Herzschlag nicht rechtzeitig auslöst, kann zum Herzstillstand führen. Die Motorsteuerung eines Autos, die den Motor nicht im richtigen Takt mit Treibstoff versorgt, kann schnell schwere Unfälle auslösen.

Üblicherweise modelliert man Echtzeitsysteme mit Hilfe von *Zeitautomaten* [3]. Sie ermöglichen es, sowohl die Aktivitäten als auch das zeitliche Verhalten von Echtzeitsystemen zu beschreiben. Sie stellen eine Erweiterung von endlichen Automaten dar, die zur Modellierung von digitalen Systemen verwendet werden. Die Erweiterung besteht in einer Menge von reell-wertigen Variablen (Uhrenvariablen), durch die Echtzeit modelliert werden kann. Mit Hilfe dieser Uhren ist es möglich, die Verweildauer in einem Zustand und die Zeitpunkte der Zustandsübergänge zu spezifizieren.

Wie stellt man die Korrektheit solchen Systems sicher? Durch die Uhren, die beliebige nicht-negative reelle Werte annehmen können, ist der Zustandsraum eines Zeitautomaten unendlich. Durch Simulation und Tests kann man deswegen nur einen geringen Anteil der möglichen Systemabläufe abdecken. Dabei können zwar Fehler gefunden werden, aber die Fehlerfreiheit kann dadurch nicht gezeigt werden. Somit muss man auf die formale Verifikation zurückgreifen. Dabei werden die Anforderungen, die an das Echtzeitsystem gestellt werden (z. B. „Von jedem Zustand aus muss spätestens innerhalb von 25 Millisekunden, nachdem ein Aufprall festgestellt wurde, der Zustand erreicht werden, der das Aufblasen des Airbags auslöst“) als Formeln einer Logik spezifiziert. Für Zeitautomaten wird in der Regel dafür die Logik TCTL (engl.: timed computation tree logic) [4] verwendet.

Die algorithmische Überprüfung solcher Eigenschaften wird als Modellprüfung (engl.: model checking) bezeichnet. Für Zeitautomaten beruht sie auf der Konstruktion des sogenannten Regionenautomaten [2]. Dies ist ein endlicher Automat, in dem alle Systemzustände des Zeitautomaten, die sich äquivalent verhalten, in einer Region zusammengefasst sind. Man kann zeigen, dass die unendlich vielen Zustände immer in endlich viele Regionen zusammengefasst werden können. Darauf können die spezifizierten Eigenschaf-

ten mit Hilfe der Algorithmen für das Model Checking für Systeme mit einer endlichen Zahl von Zuständen überprüft werden. Dieser Regionenautomat ist zwar endlich, besteht aber im Allgemeinen aus exponentiell vielen Regionen (in der Größe des ursprünglichen Zeitautomaten). Dies führt zu einem hohen Speicherverbrauch und großen Laufzeiten, was die Verifikation realer Systeme mittels Modellprüfung erschweren kann.

Als Alternative wurde die SAT-basierte beschränkte Modellprüfung (engl.: bounded model checking, BMC) entwickelt [6, 7]. Typischerweise will man damit Invarianteneigenschaften widerlegen – also Eigenschaften, die in jedem erreichbaren Zustand gelten müssen. Dabei wird ein Ablauf des Systems, der nach einer vorgegebenen Anzahl von Schritten in einen Zustand gelangt, in dem die Invarianteneigenschaft verletzt ist, als Erfüllbarkeitsproblem formuliert. Ist diese Formel erfüllbar, so entspricht die erfüllende Belegung der Variablen einem Gegenbeispiel für die Invarianteneigenschaft. Mit Hilfe dieses Gegenbeispiels kann der Systementwickler den Fehler reproduzieren und lokalisieren. Wird für eine bestimmte Abwicklungstiefe kein Fehler gefunden, so fährt man mit der nächsten Tiefe fort.

Der Vorteil von BMC ist, dass die Länge der Formel mit steigender Abwicklungstiefe linear in der Systemgröße ist und nicht der vollständige Zustandsraum des Systems exploriert werden muss. BMC ist jedoch nicht vollständig, da die Überprüfung nur bis zu einer gewissen Tiefe erfolgt.

BMC lässt sich auch auf Zeitautomaten anwenden [18, 15]. Die resultierende Formel ist allerdings kein SAT-Problem, sondern ein sogenanntes SMT-Problem. Anstelle boolescher Literale dürfen SMT-Formeln Atome einer anderen Logik wie beispielweise Differenzenlogik [16] oder linearer Arithmetik enthalten. Für solche Formeln wurden in den letzten Jahren effiziente Entscheidungsverfahren entwickelt, z. B. [10, 9]. Dadurch wurde es möglich, BMC auf große Echtzeitsysteme anzuwenden.

Reale Systeme werden aufgrund ihrer Komplexität meist als Netzwerke von mehreren Zeitautomaten modelliert. Dabei wird für jede Komponente des zu untersuchenden Systems ein eigener Automat erstellt. Die Kommunikation der einzelnen Module miteinander kann durch Synchronisation mittels Kanälen oder gemeinsamen Variablen erfolgen. Dadurch können sehr große und komplexe Systeme aus kleinen, überschaubaren Teilen erzeugt werden. Während das (vollständige) Model Checking auf der Parallelkomposition arbeitet, ist bei BMC die Überprüfung auf Fehler möglich, ohne dass zuvor die Komposition explizit berechnet werden muss.

Im Rahmen dieser Arbeit wollen wir uns mit BMC für *unvollständige* Netzwerke von Zeitautomaten beschäftigen. Unvollständig heißt in diesem Zusammenhang, dass nur ein Teil der Automaten des Netzwerks vorliegt, der Aufbau der übrigen jedoch nicht bekannt ist. Lediglich ihre Schnittstelle nach außen, d. h. wie sie mit ihrer Umgebung kommunizieren, ist festgelegt. Man will für solche unvollständigen Entwürfe herausfinden, ob die vorliegenden Teile bereits fehlerhaft sind. Ein Fehler liegt vor, wenn für alle möglichen Implementierungen der fehlenden Teile die Spezifikation verletzt ist.

Unvollständige Netzwerke spielen in drei Situationen für die Verifikation eine Rolle:

- Während der Systementwicklung liegen noch nicht alle Teile des Modells vor; trotzdem soll bereits nach Fehlern in den vorliegenden Modulen gesucht werden. Je

früher im Entwurfsprozess solche Fehler gefunden werden, desto geringer sind die Kosten für ihre Beseitigung.

- Die zu untersuchende Eigenschaft hängt von Teilen des Modells nicht ab. Durch Entfernen aller nicht benötigten Teile kann das Modell verkleinert werden. Die Hoffnung ist, dass dadurch die Effizienz gesteigert werden kann und dass man dadurch Modelle untersuchen kann, deren Verifikation ansonsten an ihrer Größe scheitern würde.

- Bei der Fehlerdiagnose kann der Fehlerort lokalisiert werden, indem nach und nach Teile des Systems entfernt werden. Der Fehler muss in einem der nicht entfernten Teile liegen, wenn nachgewiesen werden kann, dass er unabhängig von den entfernten Teilen auftritt.

Die Verifikation unvollständiger digitaler Schaltkreise ist seit einigen Jahren Gegenstand der Forschung. Nopper und Scholl haben symbolische Methoden zur Modellprüfung unvollständiger Schaltkreise entwickelt [17], während Herbstritt et al. BMC darauf angewendet haben [11]. Miller et al. haben deren Methoden weiterentwickelt [14], um eine größere Genauigkeit bei der Fehlersuche zu erreichen, ohne die Laufzeit unnötig zu vergrößern. Für die Verifikation unvollständiger Netzwerke von Zeitautomaten sind uns keine Arbeiten bekannt.

Das *Ziel dieser Arbeit* ist die Entwicklung eines Verfahrens, mit dem BMC auf vollständige und unvollständige Netzwerke von Zeitautomaten angewendet werden kann, um die Gültigkeit von Invarianteneigenschaften zu widerlegen. Dabei sollen drei unterschiedliche Kommunikationsmechanismen unterstützt werden: gemeinsame Uhrenvariablen, Synchronisation über Kanäle und gemeinsame Integervariablen, die auch gleichzeitig in demselben Netzwerk verwendet werden können.

Bei der Erzeugung der BMC-Formeln soll zwischen mehreren Varianten ausgewählt werden können: Entweder lässt man in jedem Schritt zu, dass Zeit vergeht, oder, dass ein diskreter Zustandsübergang erfolgt (alternierende Transitionsrelation), oder man erlaubt in jedem Schritt beide Möglichkeiten (nicht-alternierende Transitionsrelation). Eine weitere Unterscheidung kann vorgenommen werden nach der Art der Formel, die erzeugt wird. Die eine Möglichkeit ist, eine quantorenfreie SMT-Formel aufzustellen. Für unvollständige Entwürfe, die mit Hilfe von Integervariablen kommunizieren, bieten sich als zweite Möglichkeit quantifizierte SMT-Formeln an. Wir werden Methoden für die Erzeugung beider Arten von Formeln entwickeln.

Das BMC-Verfahren, das wir im Rahmen dieser Diplomarbeit entwickeln, soll alle diese Möglichkeiten unterstützen. Sie sollen implementiert und anhand einer Fallstudie miteinander verglichen werden. In der Implementierung soll zwischen verschiedenen Werkzeugen zum Lösen der Formeln, sogenannten Solvern, ausgewählt werden können. Sofern der ausgewählte Solver dies unterstützt, soll ein inkrementeller Modus verwendet werden können, in dem der Solver Informationen, die er beim Lösen einer Abwicklungstiefe gewonnen hat, in der nächsten weiter verwendet, um das Lösen zu beschleunigen.

Diese Arbeit ist wie folgt aufgebaut: Im nächsten Kapitel werden wir zunächst die Grundlagen von Zeitautomaten und BMC legen. Dann werden wir im Kapitel 3 darauf eingehen, wie BMC auf Zeitautomaten angewendet werden kann. In Kapitel 4 beschreiben wir, wie BMC erweitert werden kann, damit unvollständige Netzwerke von Zeitautomaten untersucht werden können. Experimentelle Untersuchungen schließen sich in Kapitel 5 an. Am Ende fassen wir die Ergebnisse dieser Arbeit zusammen und zeigen Stellen auf, an denen weiterer Forschungsbedarf besteht.

2 Grundlagen

Wir werden in diesem Kapitel die Grundlagen beschreiben, die zum Verständnis dieser Arbeit notwendig sind. Dabei handelt es sich um Zeitautomaten und Netzwerke von Zeitautomaten mit verschiedenen Kommunikationsmechanismen. Darüber hinaus gehen wir allgemein auf Bounded Model Checking ein, das wir erweitern wollen, um Fehler in unvollständigen Netzwerken von Zeitautomaten zu finden. Für diese Anpassung benötigen wir anstelle von booleschen Formeln sogenannte SMT-Formeln, die wir am Ende dieses Kapitels einführen werden.

2.1 Zeitautomaten

Als erstes betrachten wir das Grundmodell von Zeitautomaten, das später auf unterschiedliche Arten erweitert wird.

2.1.1 Grundmodell von Zeitautomaten

Zeitautomaten sind Transitionssysteme, die um sogenannte *Uhrenvariablen* erweitert wurden. Uhrenvariablen messen die verstrichene Zeit seit dem Systemstart bzw. seit sie das letzte Mal auf Null zurückgesetzt wurden. Mit Hilfe der Uhrenvariablen können Bedingungen an die Verweilzeit in einem Ort bzw. an die Ausführbarkeit von Transitionen gestellt werden. Damit lassen sich Systeme mit Echtzeitbedingungen modellieren.

Der Wertebereich von Uhrenvariablen sind die nicht-negativen reellen Zahlen. Beim Start des Systems ist der Wert aller Uhrenvariablen gleich Null. Im Folgenden bezeichnen wir Uhrenvariablen mit x, y, z.

Sei X eine Menge von Uhrenvariablen. *Variablenbedingungen* kontrollieren, wie lange das System sich in einem diskreten Zustand (oder Ort) befinden darf und wann eine Transition ausgeführt werden kann. Die Menge der Variablenbedingungen über X wird durch folgende Grammatik mit Startsymbol $\mathfrak{B}$ beschrieben:

$$\mathfrak{B} ::= \top \mid x \sim c \mid x - y \sim c \mid (\mathfrak{B} \wedge \mathfrak{B})$$

für $x, y \in X$, $c \in \mathbb{N}$ und $\sim \in \{<, \leq, \geq, >\}$. Die Menge aller Variablenbedingungen über X wird mit $\Phi(X)$ bezeichnet.

Eine *Variablenbelegung* für X weist jeder Uhrenvariable in X einen reellen Wert zu. Formal ist eine Variablenbelegung also eine Funktion $\alpha : X \to \mathbb{R}^{\geq 0}$.

Jetzt können wir rekursiv definieren, wann eine Variablenbelegung α eine Variablenbedingung $\varphi \in \Phi(X)$ erfüllt (geschrieben $\alpha \vDash \varphi$). Seien dazu $c \in \mathbb{N}$ eine Konstante und

$x, y \in X$ Uhrenvariablen. Außerdem seien $\varphi_1, \varphi_2 \in \Phi(X)$ Variablenbedingungen und $\sim \in \{<, \leq, \geq, >\}$ ein Vergleichsoperator. Dann gelte:

$$\begin{aligned} \alpha &\vDash \top, \\ \alpha \vDash x \sim c \quad &\Leftrightarrow \quad \alpha(x) \sim c, \\ \alpha \vDash x - y \sim c \quad &\Leftrightarrow \quad \alpha(x) - \alpha(y) \sim c, \\ \alpha \vDash (\varphi_1 \wedge \varphi_2) \quad &\Leftrightarrow \quad \alpha \vDash \varphi_1 \text{ und } \alpha \vDash \varphi_2. \end{aligned}$$

Für eine Konstante $\delta \in \mathbb{R}$ und eine Variablenbelegung α definieren wir die Variablenbelegung $\alpha + \delta$ durch

$$(\alpha + \delta)(x) = \alpha(x) + \delta \quad \text{für alle } x \in X.$$

Nach diesen Vorbereitungen können wir nun die formale Definition von Zeitautomaten angeben.

Definition 2.1
*Ein **Zeitautomat** ist ein Tupel $TA = (L, l^0, X, \text{inv}, E)$, wobei*

- *L eine endliche, nicht-leere Menge von Orten,*
- *$l^0 \in L$ der Anfangsort und*
- *X eine Menge von Uhrenvariablen ist.*
- *Die Funktion $\text{inv} : L \to \Phi(X)$ ordnet jedem Ort l eine Variablenbedingung $\text{inv}(l)$, die Invariante von l, zu.*
- *$E \subseteq L \times \Phi(X) \times 2^X \times L$ ist eine Menge von gerichteten Kanten zwischen Orten. Eine Kante $e \in E$ ist dabei ein Tupel $e = (l, g, r, l')$ aus einem Anfangsort $l \in L$, einer Ausführungsbedingung (engl.: guard) $g \in \Phi(X)$, einer Menge $r \subseteq X$ von zurückzusetzenden Uhren sowie einem Zielort $l' \in L$.*

Anhand eine Beispiels wollen wir die Definition von Zeitautomaten veranschaulichen.

Beispiel 2.1
In Abbildung 2.1 ist ein einfacher Zeitautomat $TA = (L, l_0, X, \text{inv}, E)$ zu sehen. Die Orte aus $L = \{l_0, l_1, l_2, l_3\}$ sind durch Kreise dargestellt. Der Anfangsort l_0 ist durch einen eingehenden Pfeil markiert. Der Automat verwendet eine Uhrenvariable x. Die Invariante der Orte l_0, l_2 und l_3 ist $\top$, die von l_1 ist $x < 2$.

Der Automat enthält fünf Kanten:

- $(l_0, \top, \{x\}, l_1)$,
- $(l_0, (x < 1), \{x\}, l_2)$,
- $(l_0, (x > 1), \emptyset, l_3)$,

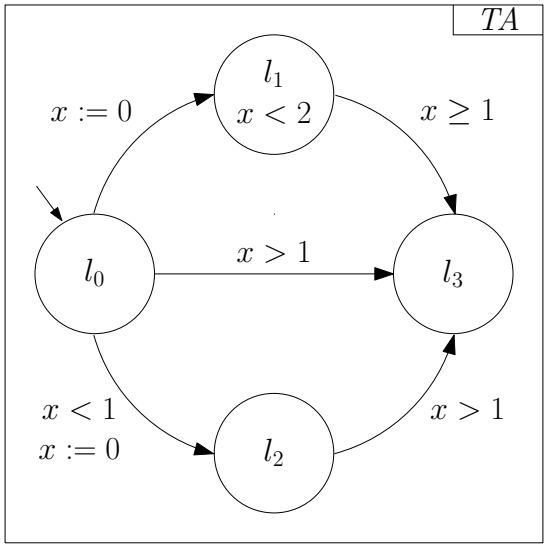

Abbildung 2.1: Beispiel für einen Zeitautomaten

- $(l_1, (x \geq 1), \emptyset, l_3)$ *und*
- $(l_2, (x > 1), \emptyset, l_3)$.

Beispielsweise darf die erste jederzeit ausgeführt werden, dabei wird x auf Null zurückgesetzt und man gelangt vom Ort l_0 in den Ort l_1. Die letzte Kante der obigen Liste hingegen darf nur ausgeführt werden, falls $x > 1$ erfüllt ist. Man gelangt dann von l_2 nach l_3, wobei die Uhrenvariable x ihren Wert behält.

Ein *Zustand* eines Zeitautomaten ist definiert durch ein Paar (l, α), wobei $l \in L$ ein Ort und α eine Variablenbelegung für die Uhrenvariablen aus X ist.

Bei Zeitautomaten kann man zwei Arten von Transitionen unterscheiden: *Verzögerungsschritte* (engl.: delay steps) und *Sprünge* (engl.: jumps). Bei einem Verzögerungsschritt bleibt der Automat in einem Ort stehen, aber die Zeit läuft weiter, d. h. die Werte aller Uhren erhöhen sich um denselben Betrag. Solange sich der Automat in einem Ort befindet, muss die Invariante dieses Ortes erfüllt sein, sonst wäre der Automat gezwungen, den Ort über eine Kante zu verlassen, deren Ausführungsbedingung gerade erfüllt ist. Existiert keine solche Kante, kann das System nicht weiterlaufen; man nennt so eine Situation einen „Time lock“. Ein solches System ist typischerweise als fehlerhaft anzusehen. Wir werden jedoch derartige Situationen in dieser Arbeit nicht weiter betrachten.

Bei Sprüngen hingegen vergeht keine Zeit, dafür geht der Automat entlang einer Kante, deren Ausführungsbedingung erfüllt ist, von einem Ort zu einem anderen über. Dabei können Uhren auf Null zurückgesetzt werden.

Formal sind Abläufe eines Zeitautomaten folgendermaßen definiert:

Definition 2.2
*Sei $TA = (L, l^0, X, \mathrm{inv}, E)$ ein Zeitautomat. Ein **Ablauf** von TA ist eine Folge $s_0 s_1 s_2 \ldots$ von Zuständen mit $s_i = (l_i, \alpha_i)$, so dass gilt:*

- $l_0 = l^0$, $\alpha_0(x) = 0$ *für alle Uhren* $x \in X$.
- *Für alle* $i \geq 0$ *gilt entweder*
  - $l_{i+1} = l_i$, *und es gibt ein* $\delta \geq 0$ *mit* $\forall x \in X : \alpha_{i+1}(x) = \alpha_i(x) + \delta$. *Für alle* $0 \leq \delta' \leq \delta$ *gilt* $\alpha_i + \delta' \vDash \text{inv}(l_i)$ *(*Verzögerungsschritt*), oder*
  - *es gibt eine Kante* $e = (l, g, r, l') \in E$ *mit* $l_i = l$, $l_{i+1} = l'$, $\alpha_i \vDash g$,

$$\alpha_{i+1}(x) = \begin{cases} 0, & \text{falls } x \in r, \\ \alpha_i(x) & \text{falls } x \in X \setminus r, \end{cases}$$

  und $\alpha_{i+1}(x) \vDash \text{inv}(l')$ *(*Sprung*).*

Beispiel 2.2
Betrachten wir nochmal den Zeitautomaten aus Abbildung 2.1. Ein möglicher Ablauf ist gegeben durch:

$$(l_0, x \mapsto 0) \to (l_0, x \mapsto 1.5) \to (l_1, x \mapsto 0) \to (l_1, x \mapsto 1.8) \to (l_2, x \mapsto 1.8) \to \ldots$$

Beginnend im Anfangszustand $(l_0, x \mapsto 0)$ *wird zuerst ein Verzögerungsschritt von 1.5 Zeiteinheiten Dauer ausgeführt; dadurch kommt man in den Zustand* $(l_0, x \mapsto 1.5)$. *Ein Sprung führt uns in den Ort* l_1*; dabei wird* x *auf* 0 *zurückgesetzt. Hier wird wiederum ein Verzögerungsschritt von 1.8 Zeiteinheiten Dauer ausgeführt. Am Ende ist der aktuelle Zustand* $(l_1, x \mapsto 1.8)$. *Jetzt kann die ausgehende Kante nach* l_3 *genommen werden; ihre Ausführungsbedingung ist erfüllt. Der resultierende Zustand ist* $(l_3, x \mapsto 1.8)$. *Nun ist nur noch ein Verzögerungsschritt beliebiger Dauer möglich.*

2.1.2 Netzwerke von Zeitautomaten

Reale Systeme sind in der Regel zu komplex, um direkt als ein einzelner Zeitautomat modelliert zu werden. Da solche Systeme aus mehreren einfacheren Komponenten zusammengesetzt sind, modelliert man zunächst jede Komponente als einen eigenen Zeitautomaten. Auf diese Weise entsteht ein *Netzwerk von Zeitautomaten*. Formal ist ein Netzwerk $\mathcal{N} = \{TA_0, \ldots, TA_{n-1}\}$ eine Menge von Zeitautomaten $TA_i = (L_i, l_i^0, X_i, \text{inv}_i, E_i)$ $(i = 0, \ldots, n-1)$. Für die Zeitautomaten im Netzwerk müssen wir Mechanismen zur Kommunikation vorsehen, damit die einzelnen Teile des Systems zusammen ihre Gesamtaufgabe erfüllen können.

Es gibt verschiedene Möglichkeiten, die Kommunikation zu realisieren. Bei Netzwerken von Zeitautomaten beschränken wir uns auf die folgenden drei gängigen Kommunikationsmechanismen:

- gemeinsame Uhrenvariablen,
- Synchronisation über Kanäle und
- gemeinsame Integervariablen,

die auch innerhalb eines Systems kombiniert werden können. Wir werden nacheinander auf diese Mechanismen eingehen, indem wir die Definition 2.1 von Zeitautomaten um die Kommunikationsmechanismen erweitern und die Regeln angeben, mit denen der Zeitautomat für das Gesamtsystem aus den Automaten für die einzelnen Komponenten erzeugt werden kann. Zunächst betrachten wir nur den Kommunikationsmechanismus mittels gemeinsamer Uhrenvariablen.

2.1.3 Netzwerke von Zeitautomaten und Kommunikation über gemeinsame Uhrenvariablen

Kommunikation über gemeinsame Uhrenvariablen bedeutet, dass die Mengen der Uhrenvariablen der Automaten im Netzwerk nicht disjunkt sein müssen. Gilt beispielsweise $x \in X_i \cap X_j$ für $i \neq j$, so können sowohl TA_i als auch TA_j auf den Wert der Uhr x zugreifen. Setzt der eine Automat x auf Null zurück, kann der andere Automat den geänderten Wert lesen und die Ausführung seiner Übergänge davon abhängig machen.

Haben wir ein Netzwerk $\mathcal{N}$ von Zeitautomaten vorliegen, so können wir daraus den Zeitautomaten $TA = (L, l^0, X, \mathrm{inv}, E)$ für das Gesamtsystem als die *Komposition* der der einzelnen Zeitautomaten mit Hilfe der folgenden Regeln konstruieren. Das Gesamtsystem hängt von den Uhrenvariablen aller Komponenten ab, X ist also die Vereinigung aller X_i. Die Orte von TA entsprechen gerade den Tupeln $l = (l_0, l_1, \ldots, l_{n-1})$ von Orten der Komponenten. TA befindet sich genau dann im Ort l, wenn sich TA_0 in l_0 befindet, TA_1 in l_1 usw. Damit dies möglich ist, müssen die Invarianten aller dieser Orte erfüllt sein. Die Invariante von l ergibt sich deshalb als Konjunktion der Invarianten von $l_0, \ldots, l_{n-1}$. Der Anfangsort von TA ist das Tupel der Anfangsorte der Komponenten, also $(l_0^0, l_1^0, \ldots, l_{n-1}^0)$. Für die Sprünge verwenden wir eine Interleaving-Semantik, d. h. die Sprünge in den Komponenten werden nacheinander ausgeführt. Wir lassen jedoch zusätzlich die gleichzeitige Ausführung von Sprüngen in verschiedenen Komponenten zu, solange es keine Schreibkonflikte auf gemeinsamen Uhrenvariablen gibt. Bei den Komponenten, die gerade keinen Sprung ausführen, ändert sich der aktuelle Ort nicht. Ein Übergang kann nur dann stattfinden, wenn die Ausführungsbedingungen aller beteiligten Kanten erfüllt sind. Eine Uhr wird bei dem Übergang zurückgesetzt, wenn sie bei einem der gleichzeitig ausgeführten Sprünge zurückgesetzt wird.

Formal sieht die Komposition wie folgt aus:

- $L = L_0 \times L_1 \times \cdots \times L_{n-1}$,
- $l^0 = (l_0^0, l_1^0, \ldots, l_{n-1}^0)$,
- $\mathrm{inv}((l_0, l_1, \ldots, l_{n-1})) = \bigwedge_{k=0}^{n-1} \mathrm{inv}_k(l_k)$ und
- $\big((l_0, \ldots, l_{n-1}), g, r, (l_0', \ldots, l_{n-1}')\big) \in E$ genau dann, wenn es eine nicht-leere Menge $J \subseteq \{0, \ldots, n-1\}$ und für alle $j \in J$ eine Kante $e_j = (l_j, g_j, r_j, l_j') \in E_j$ gibt, so dass gilt: $g = \bigwedge_{j \in J} g_j$, $r = \bigcup_{j \in J} r_j$. Außerdem gelte für alle $j \in \{1, \ldots, n\} \setminus J : l_j = l_j'$.

Anhand eines Beispiels wollen wir diese Komposition illustrieren.

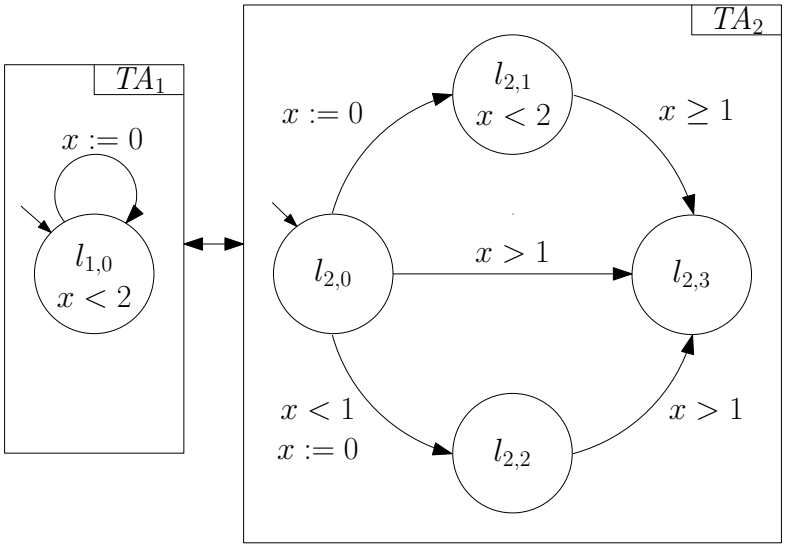

Abbildung 2.2: Netzwerk von Zeitautomaten mit Kommunikation über gemeinsame Uhrenvariablen

Beispiel 2.3
In Abbildung 2.2 ist ein Netzwerk aus zwei Zeitautomaten $TA_1 = (L_1, l_{1,0}, \{x\}, \mathrm{inv}_1, E_1)$ *und* $TA_2 = (L_2, l_{2,0}, \{x\}, \mathrm{inv}_2, E_2)$ *zu sehen, die mit Hilfe der gemeinsamen Uhrenvariable* x *kommunizieren. Die Menge der Orte der Komposition besteht aus den Paaren* $L_1 \times L_2 = \{(l_{1,0}, l_{2,0}), (l_{1,0}, l_{2,1}), (l_{1,0}, l_{2,2}), (l_{1,0}, l_{2,3})\}$*, der Anfangsort ist* $(l_{1,0}, l_{2,0})$*. Die Invariante aller vier Orte ist* $x < 2$*, die sich als Konjunktion von* $\mathrm{inv}_1(l_{1,0}) = (x < 2)$ *mit den Invarianten der Orte aus* TA_2 *ergibt.*

Für die Kanten gibt es drei Möglichkeiten: Entweder führt nur TA_1 *eine Kante aus, oder nur* TA_2 *oder beide Automaten gleichzeitig. Der Zeitautomat, der sich als Komposition der beiden Automaten* TA_1 *und* TA_2 *ergibt, ist in Abbildung 2.3 dargestellt.*

2.1.4 Netzwerke von Zeitautomaten und Kommunikation über Kanäle

In einem zweiten Schritt wollen wir Netzwerke von Zeitautomaten um die Möglichkeit erweitern, mittels *Kanälen* (engl.: channels) zu kommunizieren. Dazu dürfen Kanten der Komponenten mit Kanälen beschriftet werden.

Wie auch im vorherigen Abschnitt 2.1.2 wollen wir die gleichzeitige Ausführung von Sprüngen zulassen, solange dadurch keine Schreibkonflikte auftreten. Da dann mehrere Kanäle gleichzeitig aktiv sein können, benötigen wir formal Mengen von Kanälen auf den Kanten. Verzichtet man auf die gleichzeitige Ausführung, so genügen einzelne Kanäle. Im Allgemeinen werden die Automaten der Komponenten höchstens einen Kanal pro Kante haben, das Gesamtsystem hingegen aufgrund der Kompositionsregeln eine beliebige Anzahl.

Die intuitive Bedeutung der Kanäle ist folgende: Ist eine Kante mit einem Kanal c beschriftet, so müssen alle Automaten, in deren Kanalmenge c enthalten ist, gleichzeitig

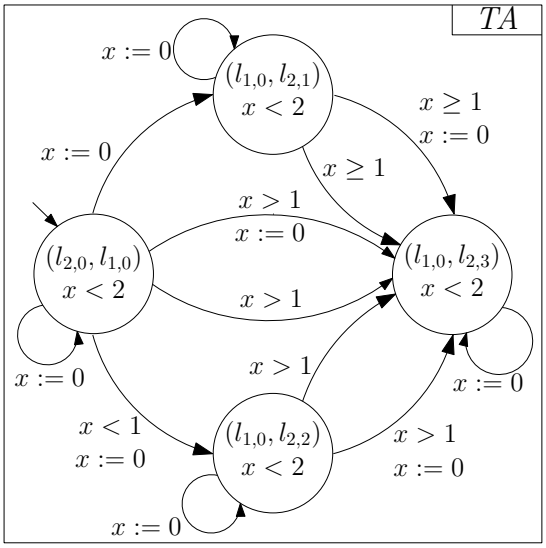

Abbildung 2.3: Komposition des Netzwerks aus Abbildung 2.2.

eine Kante ausführen, die ebenfalls mit c beschriftet ist. Dadurch kann ein Automat des Netzwerks Sprünge in anderen Automaten auslösen und auf diese Weise mit anderen Automaten des Netzwerks kommunizieren.

Wir werden zunächst die Definition 2.1 um Kanäle erweitern und danach zeigen, wie die Kompositionsregeln aus dem vorigen Abschnitt 2.1.2 angepasst werden müssen, um mit den erweiterten Kommunikationsmöglichkeiten im Rahmen des Gesamtsystems umgehen zu können.

Definition 2.3
*Ein **Zeitautomat mit Kanälen** ist ein Tupel* $TA = (L, l^0, Chan, X, \text{inv}, E)$*, wobei*

- L *eine endliche, nicht-leere Menge von Orten,*
- $l^0 \in L$ *der Anfangsort,*
- $Chan$ *eine Menge von Kommunikationskanälen und*
- X *eine Menge von Uhrenvariablen ist.*
- *Die Funktion* $\text{inv} : L \to \Phi(X)$ *ordnet jedem Ort* l *eine Variablenbedingung* $\text{inv}(l)$*, die Invariante von* l*, zu.*
- $E \subseteq L \times \Phi(X) \times 2^{Chan} \times 2^X \times L$ *ist eine Menge von gerichteten Kanten zwischen Orten. Eine Kante* $e \in E$ *ist dabei ein Tupel* $e = (l, g, C, r, l')$ *aus einem Anfangsort* $l \in L$*, einer Ausführungsbedingung (engl.: guard)* $g \in \Phi(X)$*, einer Kanalmenge* $C \subseteq Chan$*, einer Menge* $r \subseteq X$ *von zurückzusetzenden Uhren sowie einem Zielort* $l' \in L$*.*

Die wesentliche Erweiterung gegenüber Definition 2.1 besteht darin, dass dem Automaten eine Menge *Chan* von Kanälen mitgegeben wird und jede Kante mit einer Teilmenge davon beschriftet ist. Folglich unterscheiden sich die Kompositionsregeln für die Konstruktion des Gesamtsystems von denen aus dem Kapitel 2.1.2 hauptsächlich in der Definition der Kanten.

Sei $\mathcal{N} = \{ TA_0, \ldots, TA_{n-1}\}$ ein *Netzwerk von Zeitautomaten mit Kanälen* und $TA_i = (L_i, l_i^0, Chan_i, X_i, \mathrm{inv}_i, E_i)$. Die Menge L der Orte, der Anfangsort l_0, die Menge X der Uhren und die Invarianten inv der Orte von $TA = (L, l_0, Chan, X, \mathrm{inv}, E)$ für das Gesamtsystem werden exakt wie im vorigen Abschnitt berechnet, d. h.

- $L = L_0 \times L_1 \times \cdots \times L_{n-1}$,
- $l^0 = (l_0^0, l_1^0, \ldots, l_{n-1}^0)$,
- $X = \bigcup_{i=0}^{n-1} X_i$ und
- $\mathrm{inv}((l_0, l_1, \ldots, l_{n-1})) = \bigwedge_{i=0}^{n-1} \mathrm{inv}_i(l_i)$.

Die Kanalmenge *Chan* von *TA* enthält alle Kanäle der Komponenten. Also ist

- $Chan = \bigcup_{k=0}^{n-1} Chan_k$.

TA enthält eine Kante $e = \big((l_0, \ldots, l_{n-1}), g, C, r, (l'_0, \ldots, l'_{n-1})\big)$, wenn alle Automaten TA_i, deren Kanalmenge C_i Kanäle aus C enthält, eine Kante ausführen, die genau mit den gemeinsamen Elementen aus C und C_i beschriftet ist. Zusätzlich dürfen in den anderen Automaten Kanten ausgeführt werden, deren Kanalmenge leer ist. Von allen ausgeführten Kanten muss die Ausführungsbedingung erfüllt sein. Es werden bei der Ausführung von e genau die Uhren zurückgesetzt, die von einer der ausgeführten Kanten zurückgesetzt werden. Formal lässt sich dies wie folgt angeben:

- $\big((l_0, \ldots, l_{n-1}), g, C, r, (l'_0, \ldots, l'_{n-1})\big) \in E$ genau dann, wenn gilt:
  - Alle Automaten TA_j mit $Chan_j \cap C \neq \emptyset$ besitzen eine Kante $(l_j, g_j, Chan_j \cap C, r_j, l'_j) \in E_j$.
  - Für die übrigen Automaten TA_j gibt es zwei Möglichkeiten: Entweder es gibt eine Kante $(l_j, g_j, \emptyset, r_j, l'_j) \in E_j$ oder es ist $l_j = l'_j$.

  Mindestens eine Kante muss ausgeführt werden. Die Resetmenge r ergibt sich als Vereinigung der Resetmengen r_j der beteiligten Kanten. Entsprechend ist die Ausführungsbedingung g die Konjunktion der Ausführungsbedingungen g_j der beteiligten Kanten.

Anhand des folgenden Beispiels illustrieren wir die Definition von Netzwerken mit Kanälen und die zugehörigen Kompositionsregeln.

Beispiel 2.4
Abbildung 2.4 zeigt ein einfaches Netzwerk aus zwei Zeitautomaten, die mit Hilfe ihrer gemeinsamen Uhr x und dem Kanal a kommunizieren. Die Schleife $(l_{1,0}, \top, \{a\}, \{x\}, l_{1,0})$

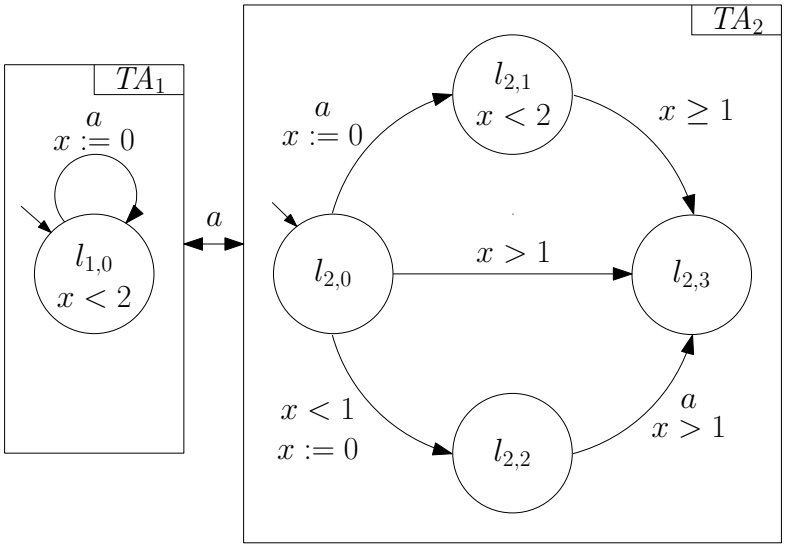

Abbildung 2.4: Netzwerk von Zeitautomaten mit Kommunikation über gemeinsame Uhrenvariablen und Kanäle

des Automaten TA_1 darf nur dann ausgeführt werden, wenn der Automat TA_2 gleichzeitig eine Kante ausführt, die ebenfalls mit a beschriftet ist. Es gibt zwei derartige Kanten $(l_{2,0}, \top, \{a\}, \{x\}, l_{2,1})$ und $(l_{2,2}, x > 1, \{a\}, \emptyset, l_{2,3})$. Ansonsten kann TA_2 nur eine unbeschriftete Kante ausgeführen, während TA_1 wartet. Der aus der Komposition von TA_1 und TA_2 resultierende Zeitautomat ist in Abbildung 2.5 dargestellt. Die Invarianten der Orte ergeben sich wie auch im Beispiel 2.3 aus der Konjunktion der Invariante von $l_{1,0}$ mit denen der Orte von TA_2. Dadurch, dass mit a beschriftete Transitionen nur noch gleichzeitig mit der Schleife ausgeführt werden dürfen, wird bei allen a-Transitionen x auf Null zurückgesetzt. Die Änderungen gegenüber TA_2 aus Abbildung 2.4 haben wir fett hervorgehoben.

2.1.5 Netzwerke von Zeitautomaten und Kommunikation über Kanäle und Integervariablen

In diesem Abschnitt erweitern wir die Zeitautomaten um den dritten Kommunikationsmechanismus, nämlich um beschränkte ganzzahlige Variablen („Integervariablen“). Dazu verallgemeinern wir zunächst die Variablenbedingungen, so dass sie auch Integervariablen enthalten können. Danach erweitern wir die Definition von Zeitautomaten und geben die Kompositionsregeln an, die die Integervariablen berücksichtigen.

Integervariablen haben einen endlichen Wertebereich aus den ganzen Zahlen. Für jede Integervariable v sind drei Werte definiert:

- $\text{lower}(v) \in \mathbb{Z}$ und $\text{upper}(v) \in \mathbb{Z}$ definieren die untere bzw. obere Grenze des Wertebereichs von v. Es gilt also immer

$$\text{lower}(v) \leq v \leq \text{upper}(v),$$

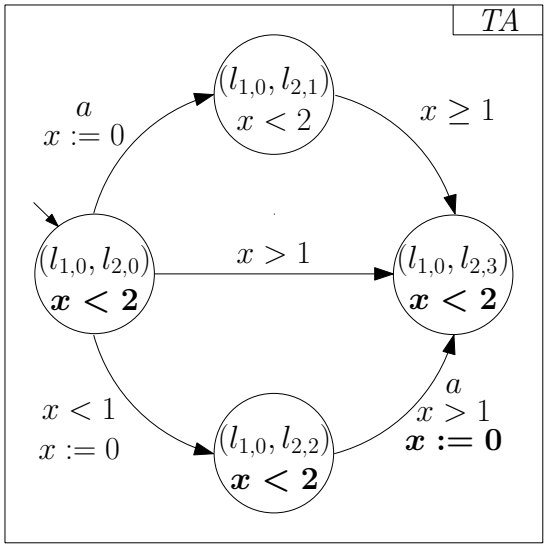

Abbildung 2.5: Komposition der Automaten aus Abbildung 2.4

- $\text{init}(v) \in \mathbb{Z}$ definiert den Anfangswert von v beim Start des Systems.

Im Folgenden bezeichnen wir Integervariablen mit den Buchstaben u, v, w.

Die Variablenbedingungen, die es bisher erlauben, Uhren mit Konstanten und mit anderen Uhren zu vergleichen, erweitern wir so, dass Bedingungen an die Werte der Integervariablen gestellt werden können. Da ihr Wertebereich endlich ist, könnten wir beliebige arithmetische Ausdrücke zulassen wie z. B. $v + 2 \cdot w \leq 4$ oder $u \cdot v = 6$. Für die Uhrenvariablen mit ihrem unendlichen Wertebereich lässt man diese nicht zu, da ansonsten das Erreichbarkeitsproblem unentscheidbar wäre [3]. Typischerweise beschränkt man sich aber auf lineare Gleichungen und Ungleichungen für die Integervariablen.

Sei X eine Menge von Uhrenvariablen und I eine Menge von Integervariablen. Die erweiterte Menge der **Variablenbedingungen** über X und I wird durch folgende Grammatik mit Startsymbol $\mathfrak{B}$ beschrieben:

$$\mathfrak{B} ::= \top \mid x \sim c \mid x - y \sim c \mid \mathfrak{I} \sim c \mid (\mathfrak{B} \wedge \mathfrak{B})$$
$$\mathfrak{I} ::= c \mid c \cdot v \mid \mathfrak{I} + \mathfrak{I}$$

für $x, y \in X$, $v \in I$, $c \in \mathbb{Z}$ und $\sim \in \{<, \leq, \geq, >\}$. Die Menge aller Variablenbedingungen über X und I wird mit $\Phi(X, I)$ bezeichnet, die Menge aller arithmetischen Ausdrücke (generiert durch das Symbol $\mathfrak{I}$) mit $\Psi(I)$.

Eine Variablenbelegung für X und I ist eine Funktion $\alpha : X \cup I \to \mathbb{R}$, so dass gilt:

- Für alle $x \in X$ gilt $\alpha(x) \in \mathbb{R}^{\geq 0}$.
- Für alle $v \in I$ gelten $\alpha(v) \in \mathbb{Z}$ und $\text{lower}(v) \leq \alpha(v) \leq \text{upper}(v)$.

Wir erweitern α auf arithmetische Ausdrücke von Integervariablen, wie sie in der obigen Grammatik durch das Symbol $\mathfrak{I}$ generiert werden:

- Für eine Konstante $c \in \mathbb{Z}$ sei $\alpha(c) = c$.
- Für eine Konstante $c \in \mathbb{Z}$ und eine Integervariable $v \in V$ sei $\alpha(c \cdot v) = c \cdot \alpha(v)$.
- Für Ausdrücke $A_1, A_2 \in \Psi(I)$ sei $\alpha(A_1 + A_2) = \alpha(A_1) + \alpha(A_2)$.

Jetzt können wir rekursiv definieren, wann eine Variablenbelegung α eine Variablenbedingung φ erfüllt (geschrieben $\alpha \vDash \varphi$). Seien dazu $c \in \mathbb{Z}$ eine Konstante, $x, y \in X$ Uhrenvariablen und $\psi \in \Psi(I)$ ein arithmetischer Ausdruck über Integervariablen. Außerdem seien $\varphi_1, \varphi_2 \in \Phi(X, I)$ Variablenbedingungen. Dann gilt insgesamt:

$$\begin{aligned} \alpha &\vDash \top, \\ \alpha \vDash x \sim c &\quad\Leftrightarrow\quad \alpha(x) \sim c, \\ \alpha \vDash x - y \sim c &\quad\Leftrightarrow\quad \alpha(x) - \alpha(y) \sim c, \\ \alpha \vDash (\varphi_1 \wedge \varphi_2) &\quad\Leftrightarrow\quad \alpha \vDash \varphi_1 \text{ und } \alpha \vDash \varphi_2, \\ \alpha \vDash \psi \sim c &\quad\Leftrightarrow\quad \alpha(\psi) \sim c, \end{aligned}$$

wobei nur die letzte Bedingung speziell für die Integervariablen dazukam.

Für eine Konstante $\delta \in \mathbb{R}$ und eine Variablenbelegung α definieren wir die Variablenbelegung $\alpha + \delta$ durch

$$(\alpha + \delta)(x) = \alpha(x) + \delta \qquad \text{und} \qquad (\alpha + \delta)(v) = v$$

für Uhrenvariablen $x \in X$ und Integervariablen $v \in I$.

Nach diesen Vorbereitungen können wir die bisherige formale Definition von Zeitautomaten um Integervariablen erweitern.

Definition 2.4
*Ein **Zeitautomat mit Integervariablen und Kanälen** ist ein Tupel $TA = (L, l^0, Chan, X, I, \text{inv}, E)$, wobei*

- *L eine endliche, nicht-leere Menge von Orten,*
- *$l^0 \in L$ der Anfangsort und*
- *$Chan$ eine Menge von Kanälen ist.*
- *X ist eine Menge von Uhrenvariablen und*
- *I eine Menge von Integervariablen.*
- *Die Funktion $\text{inv} : L \to \Phi(X, \emptyset)$ ordnet jedem Ort l eine Variablenbedingung $\text{inv}(l)$, die Invariante von l, zu.*
- *$E \subseteq L \times \Phi(X, I) \times 2^{Chan} \times 2^X \times \big(I \times \Psi(I)\big) \times L$ ist eine Menge von gerichteten Kanten zwischen Orten. Eine Kante $e \in E$ ist dabei ein Tupel $e = (l, g, C, r, a, l')$ aus einem Anfangsort $l \in L$, einer Ausführungsbedingung $g \in \Phi(X, I)$, einer Kanalmenge $C \subseteq Chan$, einer Menge $r \subseteq X$ von zurückzusetzenden Uhren und Zuweisungen*

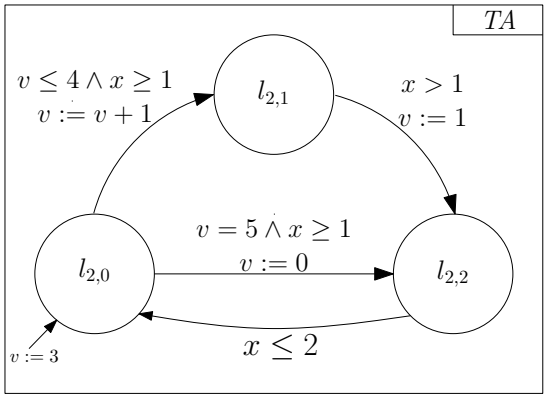

Abbildung 2.6: Zeitautomat mit einer Integervariable $v \in [0, 5] \cap \mathbb{Z}$

$a \subseteq I \times \Psi(I)$ an eine Teilmenge der Integervariablen sowie einem Zielort $l' \in L$. Dabei darf jeder Integervariablen höchstens ein Wert zugewiesen werden, d. h. es gilt für alle $v \in I$ und alle $\varphi_1, \varphi_2 \in \Psi(I) : (v, \phi_1) \in a \wedge (v, \phi_2) \in a \Rightarrow (\phi_1 = \phi_2)$.

Man beachte, dass nach der obigen Definition die Invarianten nur von den Uhrenvariablen abhängen dürfen, nicht aber von den Integervariablen. Die Aufgabe der Invarianten ist zu verhindern, dass der Zeitautomat beliebig lange in einem Ort bleiben darf, ohne einen Schritt zu machen. Da Integervariablen ihren Wert während eines Verzögerungsschritts nicht ändern, genügt es, wenn die Invarianten von Uhrenvariablen abhängen.

Anstelle von „Zeitautomat mit Integervariablen und Kanälen“ schreiben wir im Folgenden einfach „Zeitautomat“, wenn aus dem Kontext klar ist, dass es sich um die obige erweiterte Definition handelt.

Beispiel 2.5
In Abbildung 2.6 ist ein Zeitautomat mit einer Integervariablen v dargestellt. Der Wertebereich von v sei $\{0, 1, \ldots, 5\}$, der Anfangswert 3.

Die Ausführbarkeit von Kanten kann von den Uhrenvariablen und/oder den Integervariablen abhängen. Während Uhren bei der Ausführung einer Kante nur auf Null zurückgesetzt werden dürfen, können Integervariablen auf beliebige Werte innerhalb ihres Wertebereichs gesetzt werden; sogar arithmetische Ausdrücke wie $v := v + 1$ sind möglich.

Wir bezeichnen jetzt als einen *Zustand* eines Zeitautomaten ein Paar (l, α), wobei $l \in L$ ein Ort und α eine Variablenbelegung für die Uhren- und Integervariablen aus X bzw. I ist.

Bei den Transitionen ändert sich im Vergleich zum vorherigen Kapitel wenig: Bei einem Verzögerungsschritt bleibt der Zeitautomat in einem Ort stehen, wobei die Zeit weiterläuft; bei den Sprüngen vergeht keine Zeit, dafür macht der Zeitautomat einen Übergang entlang einer Kante von einem Ort zu einem anderen. Währenddessen können Uhren auf Null zurückgesetzt werden und Integervariablen können neue Werte erhalten. Die angepassten Abläufe eines Zeitautomaten sehen wie folgt aus:

Definition 2.5
Sei $TA = (L, l^0, Chan, X, I, \mathrm{inv}, E)$ ein Zeitautomat. Ein ***Ablauf*** *von TA ist eine Folge $s_0 s_1 s_2 \ldots$ von Zuständen mit $s_i = (l_i, \alpha_i)$, so dass gilt:*

- *$l_0 = l^0$, $\alpha_0(x) = 0$ für alle Uhren $x \in X$ und $\alpha_0(v) = \mathrm{init}(v)$ für alle Integervariablen $v \in I$.*
- *Für alle $i \geq 0$ gilt entweder*
  - *$l_{i+1} = l_i$, und es gibt ein $\delta \geq 0$ mit $\forall x \in X : \alpha_{i+1}(x) = \alpha_i(x) + \delta$ und $\forall v \in I : \alpha_{i+1}(v) = \alpha_i(v)$. Für alle $0 \leq \delta' \leq \delta$ gilt $\alpha_i + \delta' \vDash \mathrm{inv}(l_i)$ (*Verzögerungsschritt*).*
  - *Oder es gibt eine Kante $e = (l, g, C, r, a, l') \in E$ mit $l_i = l$, $l_{i+1} = l'$, $\alpha_i \vDash g$,*

$$\alpha_{i+1}(x) = \begin{cases} 0, & \text{falls } x \in r, \\ \alpha_i(x) & \text{falls } x \in X \setminus r, \end{cases}$$

$$\alpha_{i+1}(v) = \begin{cases} \alpha_i(\phi), & \text{falls } (v, \phi) \in a, \\ \alpha_i(v) & \text{sonst} \end{cases}$$

*und $\alpha_{i+1} \vDash \mathrm{inv}(l')$ (*Sprung*).*

Nun müssen wir noch die Definition der Kompositionsregeln so anpassen, dass sie die Integervariablen berücksichtigen. Im Wesentlichen betrifft dies wieder die Definition der Kanten.

Sei $\mathcal{N} = \{TA_0, \ldots, TA_{n-1}\}$ ein Netzwerk von Zeitautomaten $TA_i = (L_i, l_i^0, Chan_i, X_i, I_i, \mathrm{inv}_i, E_i)$. Für dessen Komposition $TA = (L, l^0, Chan, X, I, \mathrm{inv}, E)$ gilt:

- $L = L_0 \times L_1 \times \cdots \times L_{n-1}$,
- $l^0 = (l_0^0, l_1^0, \ldots, l_{n-1}^0)$,
- $Chan = \bigcup_{k=0}^{n-1} Chan_k$,
- $I = \bigcup_{k=0}^{n-1} I_k$,
- $\mathrm{inv}\big((l_0, l_1, \ldots, l_{n-1})\big) = \bigwedge_{k=0}^{n-1} \mathrm{inv}_k(l_k)$ und
- $\big((l_0, \ldots, l_{n-1}), g, C, r, (l'_0, \ldots, l'_{n-1})\big) \in E$ genau dann, wenn gilt:
  - Alle Automaten TA_j mit $Chan_j \cap C \neq \emptyset$ besitzen eine Kante $(l_j, g_j, Chan_j \cap C, r_j, l'_j) \in E_j$.
  - Für die übrigen Automaten TA_j gibt es zwei Möglichkeiten: Entweder es gibt eine Kante $(l_j, g_j, \emptyset, r_j, l'_j) \in E_j$ oder es gilt $l_j = l'_j$.

  Mindestens eine Kante muss ausgeführt werden. Die Resetmenge r ergibt sich als Vereinigung der Resetmengen r_j der beteiligten Kanten und die Zuweisungen an die Integervariablen als die Vereinigung der Zuweisungsmengen der beteiligten Kanten. Wichtig ist dabei, dass keiner Integervariable auf mehreren Kanten unterschiedliche Werte zugewiesen werden. Und zuletzt ergibt sich die Ausführungsbedingung g als die Konjunktion der Ausführungsbedingungen g_j der beteiligten Kanten.

2.2 Bounded Model Checking

Bounded Model Checking (BMC) [6, 7] ist ein SAT-basiertes Verfahren [8], um Invarianteneigenschaften für diverse Strukturen, wie z. B. Schaltkreise oder auch Zeitautomaten, zu falsifizieren. Invarianteneigenschaften sind Eigenschaften, die jeder erreichbare Systemzustand erfüllen muss.

Beispiel 2.6
Nehmen wir ein System, das für n Prozesse den wechselseitigen Ausschluss für einen kritischen Bereich garantieren soll. Eine Invarianteneigenschaft dafür wäre beispielsweise, dass sich nie mehr als ein Prozess im kritischen Zustand befindet.

Um eine solche Eigenschaft zu widerlegen, wird die Existenz eines Pfades fester Länge k, der zu einem Zustand führt, welcher die Invariante verletzt, als ein Erfüllbarkeitsproblem formuliert. Dieses Erfüllbarkeitsproblem wird mit Hilfe eines SAT-Solvers gelöst. Liefert der Solver zurück, dass das Problem erfüllbar ist, entspricht die erfüllende Belegung der Variablen einem Gegenbeispiel, d. h. einem Pfad, der im Anfangszustand beginnt und in einem Zustand endet, in dem die Invariante nicht gilt. Mit Hilfe dieses Gegenbeispiels kann ein Systementwickler den Fehler reproduzieren, lokalisieren und beheben.

Sei $T(s^i, s^{i+1})$ die Transitionsrelation, die genau dann erfüllt ist, wenn s^{i+1} ein Nachfolgezustand von s^i ist. Desweiteren sei $I(s^0)$ eine Formel, die erfüllt ist, wenn s^0 der Anfangszustand des Systems ist. Die zu falsifizierende Invarianteneigenschaft wird ebenfalls als Formel $P(s^k)$ kodiert, so dass $P(s^k)$ erfüllt ist, wenn im Zustand s^k die Invariante gilt.

Formel 2.1 beschreibt Pfade einer vorgegebenen Länge k, die im Anfangszustand s^0 beginnen und in einem Zustand s^k, der die Invariante verletzt, enden.

$$BMC(k) = I(s^0) \wedge \bigwedge_{i=0}^{k-1} T(s^i, s^{i+1}) \wedge \neg P(s^k) \tag{2.1}$$

Um die Invarianteneigenschaft zu widerlegen, beginnt man mit $k = 0$. Hat man $BMC(0)$ konstruiert, prüft man ihre Erfüllbarkeit mit Hilfe eines SAT-Solvers. Falls der Solver eine erfüllende Belegung findet, entspricht diese einem Gegenbeispiel für die Invarianteneigenschaft. Falls die Formel unerfüllbar ist, erhöht man die Abwicklungstiefe k um 1 und beginnt von Neuem.

Allerdings ist diese Methode unvollständig. Findet man keine erfüllende Belegung, so kann man nicht darauf schließen, dass das System korrekt ist. Der Vorteil von BMC gegenüber der vollständigen Modellprüfung ist, dass bei letzterer der Speicherplatzbedarf enorm groß werden kann, nämlich exponentiell in der Größe des Zeitautomaten bzw. Schaltkreises. Demgegenüber ist die Länge der Formel, die beim BMC erzeugt wird, linear in der Systemgröße. Dadurch können oft sehr viel größere Systeme analysiert werden als durch die vollständigen Model-Checking-Verfahren.

2.3 SMT-Formeln

Will man BMC auf Zeitautomaten anwenden, reichen dafür boolesche Formeln nicht aus. Stattdessen benötigt man sogenannte SMT-Formeln (engl.: SAT modulo theories). Dafür erlaubt man, boolesche Literale durch Atome einer anderen Logik zu ersetzen – im Fall von Zeitautomaten durch lineare Gleichungen und Ungleichungen.

Sei X eine Menge reellwertiger Variablen, I eine Menge ganzzahliger Variablen und B eine Menge boolescher Variablen. Für eine Variable $v \in X \cup I \cup B$ bezeichne $\mathrm{dom}(v)$ den Wertebereich von v, d. h. $\mathrm{dom}(v) = \{\top, \bot\}$ für $v \in B$, $\mathrm{dom}(v) = \mathbb{Z}$ für $v \in I$ und $\mathrm{dom}(v) = \mathbb{R}$ für $v \in X$.

Die Menge der *SMT-Formeln für die gemischt-ganzzahlige lineare Arithmetik* (im Folgenden als SMT-Formeln bezeichnet) ist gegeben durch folgende Grammatik:

$$\begin{aligned} \mathit{Term} &::= x \mid i \mid c \mid c \cdot \mathit{Term} \mid (\mathit{Term} + \mathit{Term}) \\ \mathit{Atom} &::= b \mid \mathit{Term} \sim \mathit{Term} \\ \mathit{Formel} &::= \top \mid \bot \mid \mathit{Atom} \mid \neg \mathit{Formel} \mid (\mathit{Formel} \vee \mathit{Formel}) \mid (\mathit{Formel} \wedge \mathit{Formel}) \end{aligned} \tag{2.2}$$

für boolesche Variablen $b \in B$, ganzzahlige Variablen $i \in I$, reelle Variablen $x \in X$, Konstanten $c \in \mathbb{R}$ und Vergleichsoperatoren $\sim \in \{<, \leq, \geq, >\}$.

Wie für die Uhrenvariablen und Integervariablen im Zusammenhang mit Zeitautomaten definieren wir auch hier Variablenbelegungen.

Eine Variablenbelegung $\alpha : (I \cup X \cup B) \to \mathbb{R} \cup \{\top, \bot\}$ ist eine Funktion, so dass für alle $v \in (I \cup X \cup B)$ gilt: $\alpha(v) \in \mathrm{dom}(v)$. Die Variablenbelegung erweitern wir auf Terme, Atome und Formeln wie folgt. Seien t, t_1 und t_2 Terme, φ, φ_1 und φ_2 Formeln, dann gilt:

- $\alpha(c) = c$, $\alpha(c \cdot t) = c \cdot \alpha(t)$ und $\alpha(t_1 + t_2) = \alpha(t_1) + \alpha(t_2)$.
- $\alpha(t_1 \sim t_2) = \top$, falls $\alpha(t_1) \sim \alpha(t_2)$. Ansonsten ist $\alpha(t_1 \sim t_2) = \bot$.
- $\alpha(\top) = \top$, $\alpha(\bot) = \bot$; $\alpha(\neg\varphi) = \top$, falls $\alpha(\varphi) = \bot$. Ansonsten ist $\alpha(\neg\varphi) = \bot$. $\alpha(\varphi_1 \vee \varphi_2) = \top$, falls $\alpha(\varphi_1) = \top$ oder $\alpha(\varphi_2) = \top$; ansonsten ist $\alpha(\varphi_1 \vee \varphi_2) = \bot$. Und schließlich ist $\alpha(\varphi_1 \wedge \varphi_2) = \top$, falls $\alpha(\varphi_1) = \top$ und $\alpha(\varphi_2) = \top$; ansonsten ist $\alpha(\varphi_1 \wedge \varphi_2) = \bot$.

Wir schreiben $\alpha \vDash \varphi$ für eine Variablenbelegung α und eine SMT-Formel φ, falls $\alpha(\varphi) = \top$. Eine SMT-Formel φ heißt *erfüllbar*, falls es eine Variablenbelegung α gibt mit $\alpha \vDash \varphi$. Ansonsten ist φ *unerfüllbar*.

Die so definierten SMT-Formeln sind eine Erweiterung der Variablenbedingungen, die wir für die Zeitautomaten benötigen. Das heißt, jede Variablenbedingung ist eine SMT-Formel.

Beispiel 2.7
Sei $I = \{i\}$, $X = \{x, y\}$ und $B = \{a\}$.

$$(\neg a \vee x + y = 2) \wedge (a \vee 2x - 3y \leq 0) \wedge (x + 2i > 3)$$

ist ein Beispiel für eine SMT-Formel über den drei Variablenmengen. Sie ist erfüllbar, denn durch

Variable	i	x	y	a
Belegung	2	1	1	$\top$

ist eine erfüllende Variablenbelegung gegeben.

Quantifizierte SMT-Formeln sind eine Erweiterung der oben definierten (quantorenfreien) SMT-Formeln um Quantoren. Ist φ eine SMT-Formel über den Variablenmengen X, I und B, so ist

$$Q_1 v_1 Q_2 v_2 \ldots, Q_n v_n : \varphi \tag{2.3}$$

mit $Q_i \in \{\exists, \forall\}$ und $X \cup I \cup B = \{v_1, \ldots, v_n\}$ eine quantifizierte SMT-Formel. Die Variablen v_i müssen paarweise verschieden sein. Der erste Teil $Q_1 v_1 Q_2 v_2 \ldots Q_n v_n$ wird als Quantorenpräfix der Formel bezeichnet. Variablen mit $\forall$ als Quantor heißen *universell*, solche mit $\exists$ *existentiell* quantifiziert.

Für eine Formel φ, eine Variable v und einen Wert $c \in \mathrm{dom}(v)$ bezeichne $\varphi[c/v]$ diejenige Formel, die man erhält, indem man in φ alle Vorkommen der Variablen v durch die Konstante c ersetzt.

Eine quantifizierte SMT-Formel $Q_1 v_1 Q_2 v_2 \ldots Q_n x_n : \varphi$ ist *gültig*, wenn

- φ frei von Variablen und erfüllbar ist, oder
- $Q_1 = \forall$ und für alle $c \in \mathrm{dom}(v_1)$ die Formel $Q_2 v_2 \ldots Q_n v_n : \varphi[c/v_1]$ gültig ist, oder
- $Q_1 = \exists$ und es ein $c \in \mathrm{dom}(v_1)$ gibt, so dass die Formel $Q_2 v_2 \ldots, Q_n v_n : \varphi[c/v_1]$ gültig ist.

Ansonsten ist die Formel ungültig.

Beispiel 2.8
Die Formel

$$\forall a \forall i \exists x \exists y : \big((\neg a \vee x + y = 2) \wedge (a \vee 2x - 3y \leq 0) \wedge (x + 2i > 3)\big)$$

ist eine quantifizierte SMT-Formel. Sie ist gültig, denn im Fall $a = \top$ müssen wir $\forall i \exists x \exists y : (x + y = 2 \wedge x + 2i > 3)$ erfüllen und im Fall $a = \bot$ die Formel $\forall i \exists x \exists y : (2x - 3y \leq 0 \wedge x + 2i > 3)$. Setzen wir $x = 3 - 2i + 1$, können wir im ersten Fall $y = 2 - x = 2 - 3 + 2i + 1 = 2i$ wählen, um die Formel zu erfüllen und im zweiten Fall $y \geq \frac{8}{3} - \frac{4}{3}i$, um das zweite Konjunktionsglied zu erfüllen.

Sowohl die Erfüllbarkeit von quantorenfreien SMT-Formeln als auch die Gültigkeit von quantifizierten SMT-Formeln ist entscheidbar. Für die Entscheidungsalgorithmen verweisen wir auf die entsprechende Literatur, z. B. [10] für quantorenfreie Formeln und [5] für quantifizierte Formeln.

Im nächsten Kapitel werden wir quantorenfreie SMT-Formeln verwenden, um BMC auf vollständige Netzwerke von Zeitautomaten anzuwenden. Im übernächsten Kapitel konstruieren wir für unvollständige Netzwerke sowohl quantorenfreie als auch quantifizierte SMT-Formeln für das BMC.

3 Bounded Model Checking

Wir werden in diesem Kapitel zeigen, wie Bounded Model Checking auf vollständige Netzwerke von Zeitautomaten angewendet werden kann, um die Gültigkeit von Invarianten zu widerlegen. Zunächst werden wir uns auf BMC für einzelne Zeitautomaten ohne Kommunikationskanäle und ohne Integervariablen beschränken. Danach erweitern wir BMC auf Netzwerke von Zeitautomaten, die mittels Kommunikationskanälen bzw. zusätzlich mittels Integervariablen kommunizieren, ohne dass davor die Komposition berechnet werden muss.

3.1 BMC für Zeitautomaten

Sei $TA = (L, l^0, X, \text{inv}, E)$ ein Zeitautomat, wie er in Definition 2.1 definiert wurde (also ohne Kanäle und ohne Integervariablen).

Codierung von Orten und Zuständen Wir nehmen an, die Orte $l \in L$ seien von 0 bis $|L| - 1$ durchnummeriert. Die ihnen zugeordnete Zahl bezeichnen wir mit $\text{id}(l)$. Für jeden Ort benötigen wir eine Formel, die genau dann erfüllt ist, wenn die erfüllende Belegung gerade dem betrachteten Ort entspricht. Dazu führen wir $m = \lceil \log_2(|L|) \rceil$ boolesche Variablen $at_0, \ldots, at_{m-1}$ ein. Eine Belegung dieser Variablen entspricht genau dann einem Ort l, wenn sie der Binärcodierung von $\text{id}(l)$ entspricht.

Beispiel 3.1
Nehmen wir an, die Nummer von l sei $id(l) = 5$ und L enthalte insgesamt 14 Orte. Dann benötigen wir $m = \lceil \log_2 14 \rceil = 4$ Variablen at_3, at_2, at_1, at_0 zur Codierung der Orte. Die Binärcodierung von 5 mit vier Bits ist 0101. Eine Variablenbelegung α entspricht dann dem Ort l, wenn $\alpha(at_3) = \bot$, $\alpha(at_2) = \top$, $\alpha(at_1) = \bot$ und $\alpha(at_0) = \top$ gilt.

Als Formel, die genau dann erfüllt ist, wenn die vorkommenden Variablen den Zustand l codieren, verwenden wir eine Konjunktion von Variablen und negierten Variablen. Die Variable at_i kommt negiert vor, wenn das i-te Bit von $\text{id}(l)$ gleich Null ist, ansonsten nicht negiert.

Beispiel 3.2
Betrachten wir wieder den Ort l mit $\text{id}(l) = 5$ aus dem vorigen Beispiel. Er wird durch die Formel

$$\neg at_3 \wedge at_2 \wedge \neg at_1 \wedge at_0$$

codiert.

Um die Notation der nachfolgenden Formeln übersichtlich zu halten, verwenden wir dafür die Schreibweise $at = l$.

Alle Variablen, die den Zustand des Zeitautomaten beschreiben, fassen wir in einem Vektor $s = (at_{m-1}, \ldots, at_0, x_{k-1}, \ldots, x_0)$ zusammen. Die Belegung dieser Variablen beschreibt also genau den aktuellen Zustand des Automaten.

Dies können wir nutzen, um die Bestandteile der BMC-Formel, die wir im Abschnitt 2.2 angegeben haben, zu konstruieren. Dabei handelt es sich um Formeln, die den Anfangszustand und die Transitionensrelation des Automaten sowie verletzte Invarianteneigenschaft beschreiben.

Anfangszustand Bei der Codierung des Anfangszustands ist zu beachten, dass die Formel genau dann erfüllt sein muss, wenn sich der Automat im Anfangsort l^0 befindet und alle Uhren den Wert Null haben. Dies erreichen wir durch folgende Formel:

$$I(s) := (at = l^0) \wedge \bigwedge_{x \in X} (x = 0). \tag{3.1}$$

Wie wir bereits in der Definition 2.2 eines Ablaufs eines Zeitautomaten gesehen haben, können Zeitautomaten zwei Arten von Schritten ausführen: Verzögerungsschritte und Sprünge. Zunächst erzeugen wir getrennt für beide Arten Formeln, die wir anschließend zur Transitionsformel zusammenfügen. Dabei stehen die Variablen $s, x, at, \ldots$ für den aktuellen Zustand vor der Transition und die Variablen $s', x', at', \ldots$ für den Folgezustand nach der Ausführung des Übergangs.

Verzögerungsschritte Während eines Verzögerungsschrittes vergeht Zeit, aber er ändert den aktuellen Ort eines Zeitautomaten nicht. Ein Verzögerungsschritt kann nur solange Zeit vergehen lassen, wie die Invariante des Ortes gilt. Wir müssen jedoch nicht zu jedem Zeitpunkt die Invariante überprüfen, sondern können es ausnutzen, dass die Variablenbedingungen konvexe Mengen beschreiben, d. h. wenn zwei Punkte in der Lösungsmenge liegen, dann auch alle Punkte dazwischen. Außerdem prüfen wir die Invariante explizit nach jedem Schritt, d. h. die Formel für die Verzögerungsschritte stellt sicher, dass am Ende des Verzögerungsschrittes die Invariante gilt, und die Formel für die Sprünge verlangt, dass beim Betreten des Nachfolgeortes die Invariante gilt. Mit $\mathrm{inv}'(l)$ bezeichnen wir die Invariante des Ortes l bei der alle Variablen $x \in X$ durch ihr Pendant nach der Transition ersetzt sind, d. h. durch x'.

Weiterhin müssen die Werte aller Uhrenvariablen um denselben nicht-negativen Betrag erhöht werden. Wir führen dazu eine reell-wertige Variable $\delta \geq 0$ ein, die die Zeit misst, die während des Verzögerungsschrittes vergeht. Den Nachfolgewert x' einer Uhr $x \in X$ am Ende des Verzögerungsschrittes berechnet man aus dem Wert zu Beginn des Verzögerungsschrittes durch $x' = x + \delta$.

Damit erhalten wir folgende Formel:

$$D(s, s') := (at = at') \wedge \bigwedge_{l \in L} \big((at' = l) \;\Rightarrow\; \mathrm{inv}'(l)\big) \wedge (\delta \geq 0) \wedge \bigwedge_{x \in X} (x' = x + \delta). \tag{3.2}$$

Sprünge Bei jedem Sprung wird eine Kante $e \in E$ ausgeführt. Die Formel ist somit eine Disjunktion über alle Kanten. Für jede Kante $e = (l, g, r, l')$ muss Folgendes gelten: Der Ort im aktuellen Zeitschritt muss der Ausgangsort l der Kante sein und der Ort im folgenden Zeitschritt muss der Zielort l' der Kante sein. Außerdem darf die Kante nur dann ausgeführt werden, wenn ihre Ausführungsbedingung g erfüllt ist und die Invariante des Nachfolgeortes nach dem Sprung gilt. Alle Uhrenvariablen, die in r enthalten sind, werden auf Null gesetzt; alle anderen behalten ihren Wert. Dadurch ergibt sich folgende Formel:

$$J(s, s') := \bigvee_{\substack{e \in E \\ e=(l,g,r,l')}} \Big((at = l) \wedge (at' = l') \wedge g \wedge \mathrm{inv}'(l') \wedge \bigwedge_{x \in r} (x' = 0) \wedge \bigwedge_{x \in X \setminus r} (x' = x) \Big). \tag{3.3}$$

Transitionsformel Aus den Formeln für Verzögerungsschritte und Sprünge können wir jetzt die Transitionsformel aufstellen. Dafür gibt es zwei Möglichkeiten: (1) Sprünge und Verzögerungsschritte werden immer abwechselnd ausgeführt, oder (2) in jedem Schritt ist sowohl ein Sprung als auch ein Verzögerungsschritt möglich. Die Korrektheit der ersten Variante folgt aus der Überlegung, dass zwei aufeinanderfolgende Verzögerungsschritte zu einem zusammengefasst werden können. Weiterhin können zwei aufeinanderfolgende Sprünge durch einen Verzögerungsschritt der Dauer $\delta = 0$ getrennt werden. Deshalb darf man annehmen, dass sich Verzögerungschritte und Sprünge abwechseln [1].

(1) Nicht-alternierende Formel:

$$T(s, s') := D(s, s') \vee J(s, s'). \tag{3.4}$$

(2) Alternierende Formel:

$$T(s, s') := \begin{cases} D(s, s') & \text{für gerade Abwicklungstiefen,} \\ J(s, s') & \text{für ungerade Abwicklungstiefen.} \end{cases} \tag{3.5}$$

Invarianteneigenschaft Als letzten Bestandteil benötigen wir noch die Formel für die Invarianteneigenschaft P. Wir lassen als Invarianteneigenschaften Beschreibungen von Mengen von Zuständen zu, d. h. boolesche Kombinationen von Orten (z. B., $at = l$) und Uhrenbedingungen.

Beispiel 3.3
Die Invariante „Immer wenn wir im Ort l_3 sind, muss der Wert von x größer als 5 sein“ wird durch die Formel

$$P := \big((at = l_3) \;\Rightarrow\; (x > 5) \big)$$

codiert.

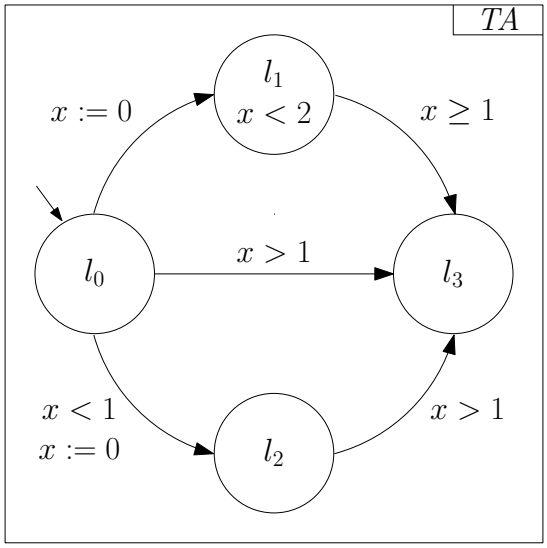

Abbildung 3.1: Beispiel für einen einfachen Zeitautomaten

Aus diesen Bestandteilen kann die BMC-Formel

$$BMC(k) := I(s_0) \wedge \bigwedge_{i=0}^{k-1} T(s_i, s_{i+1}) \wedge \neg P(s_k) \tag{3.6}$$

zusammengesetzt werden. Sie ist genau dann erfüllbar, wenn es einen Ablauf des Zeitautomaten der Länge k gibt, dessen letzter Zustand die Invarianteneigenschaft P verletzt, d. h. der $\neg P$ erfüllt.

Beispiel 3.4
Für den Zeitautomaten aus Abbildung 3.1 wollen wir mit BMC einen Ablauf finden, der die Invariante $P := \big((at = l_3) \Rightarrow (x > 1)\big)$ *verletzt. Ein solcher Ablauf lässt sich leicht finden: Vom Ort* l_0 *aus gehen wir nach* l_1*, warten dort eine Zeiteinheit und gehen weiter zum Ort* l_3*.*

Die Formeln, die wir dafür erstellen, lauten:

$$\begin{aligned}
I(s) &:= (at = l_0 \wedge x = 0),\\
D(s, s') &:= (at = at') \wedge \big((at = l_1) \Rightarrow (x < 2)\big) \wedge (\delta \geq 0) \wedge (x' = x + \delta),\\
J(s, s') &:= (at = l_0 \wedge at' = l_1 \wedge x' = 0)\\
&\quad \vee (at = l_0 \wedge at' = l_2 \wedge x < 1 \wedge x' = 0)\\
&\quad \vee (at = l_0 \wedge at' = l_3 \wedge x > 1 \wedge x' = x)\\
&\quad \vee (at = l_1 \wedge at' = l_3 \wedge x \geq 1)\\
&\quad \vee (at = l_2 \wedge at' = l_3 \wedge x > 1),\\
\neg P(s) &:= (at = l_3 \wedge x \leq 1).
\end{aligned}$$

Zunächst verwenden wir die alternierende Variante der Transitionsformel. Anschließend gehen wir auf die nicht-alternierende Variante ein.

Zuerst erstellen wir $BMC(0) = I(s^0) \wedge \neg P(s^0)$ *und prüfen diese Formel auf Erfüllbarkeit. Sie ist unerfüllbar. Dann gehen wir zur nächsten Abwicklungstiefe über und prüfen* $BMC(1) = I(s^0) \wedge D(s^0, s^1) \wedge \neg P(s^1)$ *auf Erfüllbarkeit. Auch diese Formel ist unerfüllbar. Folglich erhöhen wir wieder die Abwicklungstiefe und erstellen* $BMC(2) = I(s^0) \wedge D(s^0, s^1) \wedge J(s^1, s^2) \wedge \neg P(s^2)$*. Auch diese Formel, ist unerfüllbar, da wir mit einem Sprung nicht zum Ort* l_3 *kommen können. Die Formel* $BMC(3) = I(s^0) \wedge D(s^0, s^1) \wedge J(s^1, s^2) \wedge D(s^2, s^3) \wedge \neg P(s^3)$ *für nächste Abwicklungstiefe ist aus dem gleichen Grund unerfüllbar. Die Formel* $BMC(4) = I(s^0) \wedge D(s^0, s^1) \wedge J(s^1, s^2) \wedge D(s^2, s^3) \wedge J(s^3, s^4) \wedge \neg P(s^4)$ *hingegen besitzt eine erfüllende Belegung* α_1*, nämlich*

Variable	at^0	x^0	at^1	x^1	at^2	x^2	at^3	x^3	at^4	x^4
Belegung	l_0	0	l_0	0	l_1	0	l_1	1	l_3	1

Damit ist bewiesen, dass das System fehlerhaft ist.

Nun wollen wir die BMC-Formeln mit nicht-alternierenden Transitionsformeln erzeugen. Bei $BMC(0) = I(s^0) \wedge \neg P(s^0)$ *ergibt sich noch kein Unterschied. Die Formel für die nächste Abwicklungstiefe,* $BMC(1) = I(s^0) \wedge \big(D(s^0, s^1) \vee J(s^0, s^1)\big) \wedge \neg P(s^1)$ *hingegen enthält die Wahlmöglichkeit zwischen einem Verzögerungsschritt und einem Sprung. Sie ist unerfüllbar; deshalb gegen wir zur nächsten Tiefe über und prüfen die Formel* $BMC(2) = I(s^0) \wedge \big(D(s^0, s^1) \vee J(s^0, s^1)\big) \wedge \big(D(s^1, s^2) \vee J(s^1, s^2)\big) \wedge \neg P(s^1)$*. Auch sie ist unerfüllbar. Man kann zwar mit zwei Sprüngen in den Ort* l_3 *gelangen, allerdings muss dazwischen mindestens ein Verzögerungsschritt ausgeführt werden, damit die Ausführungsbedingung* $x \geq 1$ *erfüllt werden kann. Wir gehen deshalb zur nächsten Tiefe über; die Formel* $BMC(3) = I(s^0) \wedge \big(D(s^0, s^1) \vee J(s^0, s^1)\big) \wedge \big(D(s^1, s^2) \vee J(s^1, s^2)\big) \wedge \big(D(s^2, s^3) \vee J(s^2, s^3)\big) \wedge \neg P(s^3)$ *ist erfüllbar. Eine erfüllende Belegung* α_2 *ist gegeben durch*

Variable	at^0	x^0	at^1	x^1	at^2	x^2	at^3	x^3
Belegung	l_0	0	l_1	0	l_1	1	l_3	1

Anhand dieses Beispiels konnte man sehen, dass Fehler mit der nicht-alternierenden Variante der BMC-Formel unter Umständen in kleineren Abwicklungstiefen gefunden werden können als mit der alternierenden Variante, da zwischen zwei Sprüngen kein Verzögerungsschritt der Dauer Null eingefügt werden muss. Allerdings ist die alternierende Formel in der Regel kürzer als die nicht-alternierende, da in jedem Schritt entweder nur $D(s, s')$ oder nur $J(s, s')$ enthalten ist. Bei unseren Experimenten in Kapitel 5 werden wir untersuchen, welche der beiden Varianten effizienter ist.

3.2 BMC für Netzwerke von Zeitautomaten (Kommunikation über Kanäle)

Anstelle von einzelnen Zeitautomaten betrachten wir nun Netzwerke von Zeitautomaten, die zunächst nur über gemeinsame Uhrenvariablen und Kanäle kommunizieren. Um

eine BMC-Formel zu erhalten, die nur Uhrenvariablen als Kommunikationsmechanismus berücksichtigt, reicht es aus, die Menge der Kanäle durch die leere Menge zu ersetzen. Später gehen wir dann darauf ein, wie gemeinsame Integervariablen berücksichtigt werden können.

Sei im Folgenden ein Netzwerk $\mathcal{N} = \{TA_0, \ldots, TA_{n-1}\}$ von Zeitautomaten gegeben mit $TA_i = (L_i, l_i^0, Chan_i, X_i, \mathrm{inv}_i, E_i)$ für $i = 0, \ldots, n-1$. Wir setzen $X := \bigcup_{i=0}^{n-1} X_i$ und $Chan := \bigcup_{i=0}^{n-1} Chan_i$. Wir führen für jeden Automaten TA_i des Netzwerks einen eigenen Vektor von booleschen Variablen $at_i = (at_{i,\lceil \log_2 |L_i| \rceil - 1}, \ldots, at_{i,0})$ ein. Der Vektor s mit den Zustandsvariablen wird erweitert, so dass er alle $at_{i,j}$-Variablen und alle Uhrenvariablen enthält. Wieder bezeichnen wir bei der Codierung von Transitionen die Zustandsvariablen vor der Transition mit $s, x, at, \ldots$ und die entsprechenden Variablen danach mit $s', x', at', \ldots$

Die Codierung des Anfangszustands und der Verzögerungsschritte können direkt auf Netzwerke erweitert werden. Man erhält folgende Formeln:

$$I(s) := \bigwedge_{i=0}^{n-1} \Big((at_i = l_i^0) \wedge \mathrm{inv}_i(l_i^0) \Big) \wedge \bigwedge_{x \in X} (x = 0), \tag{3.7}$$

$$D(s, s') := \bigwedge_{i=0}^{n-1} \Big((at_i = at_i') \wedge \bigwedge_{l \in L_i} \Big((at_i = l) \;\Rightarrow\; \mathrm{inv}_i'(l) \Big) \Big) \wedge (\delta \geq 0) \wedge \bigwedge_{x \in X} (x' = x + \delta). \tag{3.8}$$

Bei den Sprüngen müssen wir die Synchronisation der einzelnen Automaten mit Hilfe der Kommunikationskanäle berücksichtigen. Dafür führen wir für jeden Kanal $c \in Chan$ eine boolesche Variable ein, die wir der Einfachheit halber auch mit c bezeichnen. Sie gibt an, ob der entsprechende Kanal aktiviert ist oder nicht.

Wenn eine Kante, die mit Kanälen $C \neq \emptyset$ beschriftet ist, ausgeführt wird, müssen alle Automaten TA_i mit $C \cap Chan_i \neq \emptyset$ zur gleichen Zeit jeweils eine Kante, die mit $C \cap Chan_i$ beschriftet ist, ausführen. Alle anderen Kanäle aus $Chan_i$ müssen inaktiv sein. Wird eine Kante von TA_i ausgeführt, die mit $\emptyset$ beschriftet ist, darf kein Kanal aus $Chan_i$ aktiv sein, jedoch möglicherweise Kanäle anderer Automaten.

Das Zurücksetzen der Uhren muss gesondert behandelt werden. Zunächst nehmen wir an, dass die Mengen der Uhren, die die verschiedenen Automaten jeweils zurücksetzen dürfen, paarweise disjunkt sind. Danach betrachten wir den Fall, dass mehrere Automaten des Netzwerks dieselbe Uhr zurücksetzen dürfen. Mehrfacher Lesezugriff auf dieselbe Uhrenvariable ist unproblematisch. Nehmen wir an, dass die Menge X_i für jeden Automaten TA_i $(i = 0, \ldots, n-1)$ genau die Uhren enthält, die TA_i zurücksetzen darf.

Es besteht die Möglichkeit, dass bei einem Sprung nicht alle Automaten des Netzwerks gleichzeitig eine Kante ausführen, sondern nur ein Teil der Automaten. Um zu modellieren, dass der Automat TA_i bei einem Sprung keine Kante ausführt, sondern wartet, verwenden wir die Formel $\mathrm{fix}(TA_i)$:

$$\mathrm{fix}(TA_i) := (at_i' = at_i) \wedge \bigwedge_{x \in X_i} (x' = x) \wedge \bigwedge_{c \in Chan_i} \neg c \wedge \bigwedge_{l \in L_i} \big((at_i = l) \;\Rightarrow\; \mathrm{inv}'(l) \big). \tag{3.9}$$

Diese Formel besagt, dass sich der aktuelle Ort von TA_i und die Werte der Uhren, die TA_i zurücksetzen darf, nicht ändern. Außerdem darf kein Kanal von TA_i aktiv sein. Da andere Automaten zur gleichen Zeit Uhren zurücksetzen dürfen, die von TA_i gelesen werden, müssen wir gewährleisten, dass nach dem Sprung die Invariante des aktuellen Ortes von TA_i immer noch erfüllt ist.

Als weitere Komponente der Formel für Sprünge führen wir die Codierung $\Theta(e, TA_i)$ einer einzelnen Kante $e = (l, g, C, r, l')$ von TA_i ein. Der Unterschied der Codierung einer Kante im Vergleich zur Formel (3.3) besteht nur in der Berücksichtigung der Kanäle. Für alle $c \in C$ muss gelten, dass c aktiv ist und alle anderen $c \in \mathit{Chan}_i \setminus C$ dagegen inaktiv sind. Damit erhalten wir folgende Formel:

$$\begin{aligned}\Theta(e, TA_i) := (at_i = l) \wedge (at_i' = l') \wedge g \wedge \mathrm{inv}'(l') \\ \wedge \bigwedge_{x \in r} (x' = 0) \wedge \bigwedge_{x \in X_i \setminus r} (x' = x) \wedge \bigwedge_{c \in C} c \wedge \bigwedge_{c \in \mathit{Chan}_i \setminus C} \neg c.\end{aligned} \tag{3.10}$$

Die Formel für die Sprünge können wir aus der Beobachtung ableiten, dass jeder Automat TA_i entweder untätig bleibt (dann ist $\mathrm{fix}(TA_i)$ erfüllt) oder eine Kante ausführt (dann muss $\Theta(e, TA_i)$ für ein $e \in E_i$ gelten):

$$J(s, s') := \bigwedge_{i=0}^{n-1} \Big(\mathrm{fix}(TA_i) \vee \bigvee_{e \in E_i} \Theta(e, TA_i) \Big). \tag{3.11}$$

Lassen wir zu, dass mehrere Zeitautomaten dieselbe Uhrenvariable zurücksetzen dürfen, dann können wir diese Codierung der Sprünge nicht mehr verwenden. Angenommen sowohl TA_i als auch TA_j $(i \neq j)$ besitzen jeweils eine Kante, auf der die Uhr x zurückgesetzt wird. Nehmen wir außerdem an, TA_i führt eine Kante e_i aus, auf der x zurückgesetzt wird, und TA_j gleichzeitig eine Kante e_j, auf der x nicht zurückgesetzt wird. Dann würde die Codierung von e_i die Formel $x' = 0$ enthalten, die von e_j aber $x' = x$. Solange $x \neq 0$ ist, führt dies zu einem Widerspruch. Laut den Kompositionsregeln für das Gesamtsystem (vgl. Abschnitt 2.1.4) ist das gleichzeitige Ausführen von e_1 und e_2 aber erlaubt. Es werden dann bei einem Sprung alle Uhren zurückgesetzt, die von irgendeinem Automaten zurückgesetzt werden.

Um dieses Problem zu beseitigen, dürfen wir nur dann verlangen, dass $x' = x$ gilt, wenn *kein* Automat x zurücksetzt. Dazu führen wir für jeden Automaten TA_i, der eine Kante $e \in E_i$ enthält, in deren Rücksetzmenge x enthalten ist, eine boolesche Variable $w_{x,i}$ ein. Sie erhält den Wert $\top$, wenn auf der aktuell ausgeführten Kante von TA_i die Uhr x zurückgesetzt wird und den Wert $\bot$ sonst. Deshalb nennen wir $w_{x,i}$ den *Schreibindikator* von x in TA_i. Damit können wir die Sprünge wie folgt codieren.

In der Formel $\mathrm{fix}(TA_i)$ ersetzen wir den Ausdruck $x' = x$ durch den Schreibindikator $\neg w_{x,i}$, um uns zu merken, dass in TA_i die Uhr x nicht zurückgesetzt wird.

$$\mathrm{fix}(TA_i) := (at_i' = at_i) \wedge \bigwedge_{x \in X_i} \neg w_{x,i} \wedge \bigwedge_{c \in \mathit{Chan}_i} \neg c \wedge \bigwedge_{l \in L_i} \big((at_i = l) \Rightarrow \mathrm{inv}'(l) \big). \tag{3.12}$$

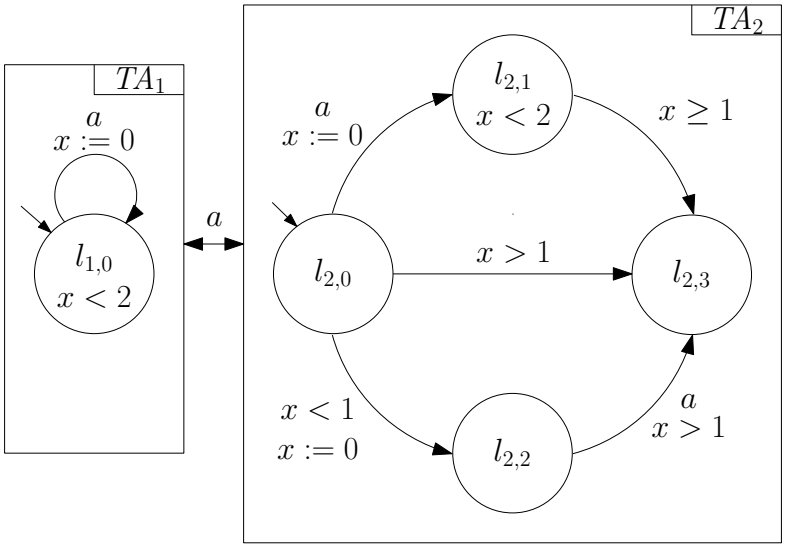

Abbildung 3.2: Netzwerk von Zeitautomaten mit Kommunikation über Kanäle

Bei der Codierung der Kanten fügen wir, wenn eine Uhr x zurückgesetzt wird, den positiven Schreibindikator $w_{x,i}$ ein. Wird x nicht zurückgesetzt, ersetzen wir den Ausdruck $x' = x$ durch den negierten Schreibindikator $\neg w_{x,i}$:

$$\begin{aligned}\Theta(e,\, TA_i) := (at_i = l) \wedge (at_i' = l') \wedge g \wedge \mathrm{inv}'(l') \\ \wedge \bigwedge_{x \in r} (x' = 0 \wedge w_{x,i}) \wedge \bigwedge_{x \in X_i \setminus r} \neg w_{x,i} \wedge \bigwedge_{c \in C} c \wedge \bigwedge_{c \in Chan_i \setminus C} \neg c.\end{aligned} \tag{3.13}$$

Nun müssen wir berücksichtigen, dass eine Uhrenvariable ihren Wert behält, wenn alle ihre Schreibindikatoren mit $\perp$ belegt sind:

$$\mathrm{noassign}(s, s')) := \bigwedge_{x \in X} \Big(\Big(\bigwedge_{\substack{i \in \{0,\ldots,n-1\} \\ x \in X_i}} \neg w_{x,i} \Big) \Rightarrow \Big(x' = x \Big) \Big). \tag{3.14}$$

Damit können wir die Formel für die Sprünge konstruieren:

$$J(s, s') := \bigwedge_{i=0}^{n-1} \Big(\mathrm{fix}(TA_i) \vee \bigvee_{e \in E_i} \Theta(e,\, TA_i) \Big) \wedge \mathrm{noassign}(s, s'). \tag{3.15}$$

Die Formeln zur Beschreibung der Transitionen – sowohl die alternierende Version (Formel (3.5)) als auch die nicht-alternierende (3.4) – bleiben unverändert, wie auch die BMC-Formel (3.6).

Beispiel 3.5
Wir erstellen die Bestandteile der BMC-Formel für das Netzwerk von Zeitautomaten aus Abbildung 3.2. Wir betrachten die Eigenschaft, dass im Ort l_3 der Wert von x immer größer als Eins sein muss, d. h. $P := \big((at = l_3) \Rightarrow (x > 1)\big)$.

Für dieses Netzwerk benötigen wir Schreibindikatoren für die Uhrenvariable x. Wenn nämlich TA_1 einen Sprung ausführt und gleichzeitig TA_2 beispielsweise die Kante von $l_{2,1}$ nach $l_{2,3}$ ausführt, dann setzt TA_1 die Uhrenvariable x zurück, TA_2 hingegen nicht. Ohne Schreibindikatoren wäre diese Kombination nicht möglich.

$$
\begin{aligned}
I(s) &:= (at_1 = l_{1,0}) \wedge (at_2 = l_{2,0}) \wedge (x = 0),\\
D(s,s') &:= (at_1 = at_1') \wedge \big((at_1 = l_{1,0}) \Rightarrow (x < 2)\big)\\
&\quad \wedge (at_2 = at_2') \wedge \big((at_2 = l_{2,1}) \Rightarrow (x < 2)\big)\\
&\quad \wedge (\delta \geq 0) \wedge (x' = x + \delta),\\
\mathrm{fix}(TA_1) &:= (at_1 = at_1') \wedge \big(at_1' = l_{1,0}) \Rightarrow (x' < 2)\big) \wedge \neg w_{x,1}\\
\mathrm{fix}(TA_2) &:= (at_2 = at_2') \wedge \big(at_2' = l_{2,1}) \Rightarrow (x' < 2)\big) \wedge \neg w_{x,2}\\
J(s,s') &:= \Big(\mathrm{fix}(TA_1) \vee \big((at_1 = l_{1,0}) \wedge (at_1' = l_{1,0}) \wedge (x' < 2) \wedge (x' = 0 \wedge w_{x,1}) \wedge a\big)\Big)\\
&\quad \wedge \Big(\mathrm{fix}(TA_2)\\
&\qquad \vee \big((at_2 = l_{2,0}) \wedge (at' = l_{2,1}) \wedge (x' < 2) \wedge (x' = 0 \wedge w_{x,2}) \wedge a\big)\\
&\qquad \vee \big((at_2 = l_{2,0}) \wedge (at' = l_{2,2}) \wedge (x < 1) \wedge (x' = 0 \wedge w_{x,2}) \wedge \neg a\big)\\
&\qquad \vee \big((at_2 = l_{2,0}) \wedge (at' = l_{2,3}) \wedge (x > 1) \wedge \neg w_{x,2} \wedge \neg a\big)\\
&\qquad \vee \big((at_2 = l_{2,1}) \wedge (at' = l_{2,3}) \wedge (x \geq 1) \wedge \neg w_{x,2} \wedge \neg a\big)\\
&\qquad \vee \big((at_2 = l_{2,2}) \wedge (at' = l_{2,3}) \wedge (x > 1) \wedge \neg w_{x,2} \wedge a\big)\\
&\quad \Big) \wedge \big((\neg w_{x,1} \wedge \neg w_{x,2}) \Rightarrow (x' = x)\big),\\
\neg P(s) &:= \big((at_2 = l_{2,3}) \wedge (x \leq 1)\big).
\end{aligned}
$$

Daraus kann wie zuvor die BMC-Formel für die verschiedenen Abwicklungstiefen erzeugt werden. Bei der alternierende Version wird eine erfüllende Belegung bei Abwicklungstiefe 4 gefunden. Die folgende Tabelle zeigt eine möglich erfüllende Belegung (ein „−“ bedeutet, dass diese Variable in der Abwicklungstiefe nicht vorkommt).

Tiefe	at_1	$w_{x,1}$	at_2	$w_{x,2}$	x	a
0	$l_{1,0}$	−	$l_{2,0}$	−	0	−
1	$l_{1,0}$	$\top$	$l_{2,0}$	$\top$	0	$\top$
2	$l_{1,0}$	−	$l_{2,1}$	−	0	−
3	$l_{1,0}$	$\bot$	$l_{2,1}$	$\bot$	1	$\bot$
4	$l_{1,0}$	−	$l_{2,3}$	−	1	−

Bei der nicht-alternierenden Version wird der Fehler bereits in Tiefe 3 gefunden. Die entsprechende Variablenbelegung lautet:

Tiefe	at_1	$w_{x,1}$	at_2	$w_{x,2}$	x	a
0	$l_{1,0}$	$\top$	$l_{2,0}$	$\top$	0	$\top$
1	$l_{1,0}$	−	$l_{2,1}$	−	0	−
2	$l_{1,0}$	$\bot$	$l_{2,1}$	$\bot$	1	$\bot$
3	$l_{1,0}$	−	$l_{2,3}$	−	1	−

3.3 BMC für Netzwerke von Zeitautomaten (Kommunikation über Kanäle und Integervariablen)

Zuletzt zeigen wir, wie BMC-Formeln für Netzwerke von Zeitautomaten konstruiert werden können, die nicht nur über Kanäle, sondern auch über gemeinsame Integervariablen kommunizieren. Um eine BMC-Formel zu erhalten, die nur Integervariablen als Kommunikationsmechanismus berücksichtigt, reicht es aus, die Menge der Kanäle durch die leere Menge zu ersetzen.

Sei dazu ein Netzwerk $\mathcal{N} = \{TA_0, \ldots, TA_{n-1}\}$ von Zeitautomaten gegeben mit $TA_i = (L_i, l_i^0, Chan_i, X_i, I_i, \mathrm{inv}_i, E_i)$. Wir setzen $X := \bigcup_{i=0}^{n-1} X_i$ und $I := \bigcup_{i=0}^{n-1} I_i$ sowie $Chan := \bigcup_{i=0}^{n-1} Chan_i$.

Codierung des Anfangszustands Bei der Erzeugung der Formel für den Anfangszustand muss jeder Zeitautomat des Netzwerks sich im Anfangsort befinden, alle Uhren müssen mit Null und alle Integervariablen mit ihrem Anfangswert initialisiert werden. Das wird durch die folgende Formel beschrieben:

$$I(s) := \bigwedge_{i=0}^{n-1} \big((at_i = l_i^0) \wedge \mathrm{inv}_i(l_i^0)\big) \wedge \bigwedge_{x \in X} (x = 0) \wedge \bigwedge_{v \in I} \big(v = \mathrm{init}(v)\big) \tag{3.16}$$

Codierung von Verzögerungsschritten Gegenüber der Formel (3.8) für Netzwerke ohne Integervariablen ändert sich nur, dass sich die Werte der Integervariablen bei einem Verzögerungsschritt nicht ändern dürfen. Dies führt zu folgender Formel:

$$\begin{aligned} D(s, s') := \bigwedge_{i=0}^{n-1} \Bigg((at_i = at_i') \wedge \bigwedge_{l \in L_i} \Big((at_i = l) \;\Rightarrow\; \mathrm{inv}_i'(l) \Big) \Bigg) \\ \wedge (\delta \geq 0) \wedge \bigwedge_{x \in X} (x' = x + \delta) \wedge \bigwedge_{v \in I} (v' = v). \end{aligned} \tag{3.17}$$

Codierung von Sprüngen Die Berücksichtigung der Integervariablen bei der Codierung von Sprüngen kann ähnlich wie die der Uhrenvariablen erfolgen. Da Integervariablen – genau wie Uhrenvariablen – ihren Wert bei einem Sprung behalten, wenn ihnen von keiner ausgeführten Kante ein neuer Wert zugewiesen wird, benötigen wir auch für die Integervariablen Schreibindikatoren, falls eine Integervariable von mehr als einem Automaten geschrieben werden darf. Die boolesche Variable $w_{v,i}$ für $v \in I$ und $0 \leq i \leq n-1$ speichert, ob auf der Kante, die in TA_i aktuell ausgeführt wird, eine Zuweisung an die Integervariable v erfolgt.

Dann lassen sich die Bestandteile der Sprung-Formel jeweils wie folgt formulieren:

$$\begin{aligned} \mathrm{fix}(TA_i) := (at_i' = at_i) \wedge \bigwedge_{x \in X_i} \neg w_{x,i} \wedge \bigwedge_{v \in I_i} \neg w_{v,i} \\ \wedge \bigwedge_{c \in Chan_i} \neg c \wedge \bigwedge_{l \in L_i} \big((at_i = l) \;\Rightarrow\; \mathrm{inv}'(l) \big), \end{aligned} \tag{3.18}$$

Der Übersichtlichkeit halber bezeichnen wir die Menge der Integervariablen, denen in a kein neuer Wert zugewiesen wird, also die Menge $\{v \in I \mid \nexists \varphi \in \Psi(I) : (v, \varphi) \in a\}$ mit $I \setminus a$. Dann erhalten wir

$$\begin{aligned}\Theta(e, TA_i) := (at_i = l) \wedge (at_i' = l') \wedge g \wedge \mathrm{inv}'(l') \\ \wedge \bigwedge_{x \in r} (x' = 0 \wedge w_{x,i}) \wedge \bigwedge_{x \in X_i \setminus r} \neg w_{x,i} \\ \wedge \bigwedge_{(v,\varphi) \in a} (v' = \varphi(I) \wedge w_{v,i}) \wedge \bigwedge_{v \in I_i \setminus a} \neg w_{v,i} \\ \wedge \bigwedge_{c \in C} c \wedge \bigwedge_{c \in Chan_i \setminus C} \neg c,\end{aligned} \tag{3.19}$$

$$\begin{aligned}\mathrm{noassign}(s, s') := \bigwedge_{x \in X} \Bigg(\Big(\bigwedge_{\substack{i \in \{0,\ldots,n-1\} \\ x \in X_i}} \neg w_{x,i} \Big) \Rightarrow \Big(x' = x \Big) \Bigg) \\ \wedge \bigwedge_{v \in I} \Bigg(\Big(\bigwedge_{\substack{i \in \{0,\ldots,n-1\} \\ v \in I_i}} \neg w_{v,i} \Big) \Rightarrow \Big(v' = v \Big) \Bigg).\end{aligned} \tag{3.20}$$

An der Formel für die Sprünge selbst ändert sich nichts:

$$J(s, s') := \bigwedge_{i=0}^{n-1} \Bigg(\bigvee_{e \in E_i} \Phi(e, TA_i) \vee \mathrm{fix}(TA_i) \Bigg) \wedge \mathrm{noassign}(s, s'). \tag{3.21}$$

Wertebereich der Integervariablen Wir müssen nun noch sicherstellen, dass zu jedem Zeitpunkt der Wertebereich der Integervariablen berücksichtigt wird. Dazu definieren wir die Formel

$$\mathrm{range}(s) := \bigwedge_{v \in I} \big(v \geq \mathrm{lower}(v) \wedge v \leq \mathrm{upper}(v) \big). \tag{3.22}$$

Transitionsformel Da wir annehmen, dass der Anfangswert jeder Integervariablen in ihrem Wertebereich liegt, genügt es, den Wertebereich *nach* jedem Schritt zu prüfen. Die Formel für den Wertebereich der Integervariablen fügen wir deshalb der Transitionsformel hinzu. Damit sehen die alternierende und die nicht-alternierende Version wie folgt aus:

(1) Nicht-alternierende Formel:

$$T(s, s') := \big(D(s, s') \vee J(s, s') \big) \wedge \mathrm{range}(s'). \tag{3.23}$$

(2) Alternierende Formel:

$$T(s, s') := \begin{cases} D(s, s') \wedge \mathrm{range}(s') & \text{für gerade Abwicklungstiefen,} \\ J(s, s') \wedge \mathrm{range}(s') & \text{für ungerade Abwicklungstiefen.} \end{cases} \tag{3.24}$$

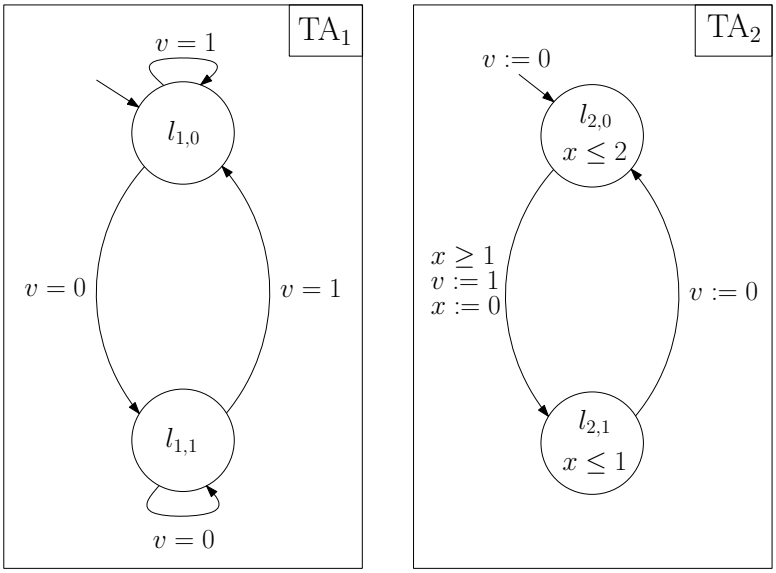

Abbildung 3.3: Netzwerk von Zeitautomaten, die über eine gemeinsame Integervariable kommunizieren

Beispiel 3.6
Wir wollen für das Netzwerk von Zeitautomaten, die über die gemeinsame Integervariable v kommunizieren, eine BMC-Formel aufstellen. Es gelte $\mathrm{lower}(v) = 0$, $\mathrm{upper}(v) = 1$, $\mathrm{init}(v) = 0$. *Der Einfachheit halber verzichten wir auf Schreibindikatoren, da TA_1 keine der Variablen schreibt. Als Invarianteneigenschaft untersuchen wir, ob in $l_{1,1}$ immer $v = 0$ gilt, d. h. $P := \big((at_1 = l_{1,1}) \Rightarrow (v = 0)\big)$.*

$$
\begin{aligned}
I(s) &:= (at_1 = l_{1,0}) \wedge (at_2 = l_{2,0}) \wedge (x = 0) \wedge (v = 0),\\
D(s, s') &:= (at_1 = at_1') \wedge (at_2 = at_2') \wedge \big((at_2 = l_{2,0}) \Rightarrow (x' \leq 2)\big)\\
&\quad \wedge \big((at_2' = l_{2,1}) \Rightarrow (x \leq 1)\big) \wedge (\delta \geq 0) \wedge (x' = x + \delta) \wedge (v' = v),\\
\mathrm{fix}(TA_1) &:= (at_1 = at_1')\\
\mathrm{fix}(TA_2) &:= (at_2 = at_2') \wedge (x' = x) \wedge (v' = v)\\
&\quad \wedge \big((at_2' = l_{2,0}) \Rightarrow (x \leq 2)\big) \wedge \big((at_2' = l_{2,1}) \Rightarrow (x \leq 1)\big),
\end{aligned}
$$

$$
\begin{aligned}
J(s,s') := \Big(&\mathrm{fix}(TA_1) \\
&\quad \vee \big((at_1 = l_{1,0}) \wedge (at_1' = l_{1,0}) \wedge (v = 1) \wedge (v' = v)\big) \\
&\quad \vee \big((at_1 = l_{1,0}) \wedge (at_1' = l_{1,1}) \wedge (v = 0) \wedge (v' = v)\big) \\
&\quad \vee \big((at_1 = l_{1,1}) \wedge (at_1' = l_{1,0}) \wedge (v = 1) \wedge (v' = v)\big) \\
&\quad \vee \big((at_1 = l_{1,1}) \wedge (at_1' = l_{1,1}) \wedge (v = 0) \wedge (v' = v)\big) \\
&\Big) \wedge \Big(\mathrm{fix}(TA_2) \\
&\quad \vee \big((at_2 = l_{2,0}) \wedge (at_2' = l_{2,1}) \wedge (x \geq 1) \wedge (v' = 1) \wedge (x' = 0)\big) \\
&\quad \vee \big((at_2 = l_{2,1}) \wedge (at_2' = l_{2,0}) \wedge (v' = 0)\big) \\
&\Big) \\
\mathrm{range}(s) :=& (v \geq 0) \wedge (v \leq 1), \\
\neg P(s) :=& \big((at_1 = l_{1,1}) \wedge (v = 1)\big).
\end{aligned}
$$

Folgender Ablauf führt zum unsicheren Zustand:

$$
\begin{aligned}
(l_{1,0}, l_{2,0}, x = 0, v = 0) &\rightarrow (l_{1,1}, l_{2,0}, x = 0, v = 0) \\
&\rightarrow (l_{1,1}, l_{2,0}, x = 1, v = 0) \\
&\rightarrow (l_{1,1}, l_{2,1}, x = 0, v = 1).
\end{aligned}
$$

Mit der nicht-alternierenden Formel für die Transitionen, die durch die Disjunktion von $D(s,s')$ und $J(s,s')$ gebildet wird, wird dieser Ablauf in Abwicklungstiefe 3 gefunden, mit der alternierenden Variante in Tiefe 4.

Nachdem wir in diesem Kapitel gezeigt haben, wie sich BMC auf Netzwerke von Zeitautomaten mit unterschiedlichen Kommunikationsmechanismen anwenden lässt, werden wir im nächsten Kapitel Lösungen für unvollständige Netzwerke vorstellen.

4 Verifikation unvollständiger Netzwerke von Zeitautomaten

Wir haben im letzten Kapitel gezeigt, wie BMC verwendet werden kann, um Abläufe in vollständigen Netzwerken von Zeitautomaten zu finden, die zu einem Zustand des Netzwerks führen, in dem eine vorgegebene Invarianteneigenschaft verletzt ist. In diesem Abschnitt werden wir vorstellen, wie *unvollständige Netzwerke* von Zeitautomaten mittels BMC untersucht werden können. Unvollständig bedeutet, dass der innere Aufbau von einzelnen Zeitautomaten des Netzwerks nicht bekannt ist (sogenannte *Blackboxes*), sondern nur ihre Schnittstelle nach außen, d. h. über welche Kanäle sie sich mit anderen (bekannten) Automaten synchronisieren und/oder welche Integervariablen sie schreiben.

Wir werden BMC so anpassen, dass damit überprüft werden kann, ob die bereits vollständig vorhandenen Teile des Netzwerks fehlerhaft sind, d. h. ob es für jede mögliche Implementierung der Blackboxes einen Ablauf des Systems gibt, der zu einem unsicheren Zustand führt, in dem die Invarianteneigenschaft verletzt ist. In diesem Fall gibt es keine Implementierung der Blackboxes, so dass das Gesamtsystem korrekt ist. Die Eigenschaft wird in diesem Fall als *nicht realisierbar* bezeichnet.

Für unvollständige Netzwerke von Zeitautomaten werden wir zwei unterschiedliche Techniken vorstellen. Die erste verwendet – wie der BMC-Ansatz aus dem vorigen Kapitel – *SMT-Formeln ohne Quantoren*, für die eine erfüllende Belegung gesucht wird. Diesen Ansatz untersuchen wir in Abschnitt 4.2 sowohl für Netzwerke, die mittels Kanälen kommunizieren, als auch für solche, die zusätzlich gemeinsame Integervariablen verwenden.

Für Netzwerke, die (neben Kanälen) mittels Integervariablen kommunizieren, gibt es jedoch Fälle, in denen dieser Ansatz kein Ergebnis liefert, obwohl für jede Implementierung der Blackboxes ein Ablauf des Netzwerks existiert, der zu einem Zustand führt, in dem die Invariante verletzt ist. Mehr Lösungen erhält man durch den zweiten Ansatz, der *quantifizierte SMT-Formeln* verwendet. Damit können mehr Fehler entdeckt werden, allerdings ist die Überprüfung dieser Formeln wesentlich teurer, was wir im Kapitel 5 anhand von experimentellen Ergebnissen zeigen werden. Im Abschnitt 4.3 formulieren wir das BMC-Problem als quantifizierte SMT-Formeln.

4.1 Unvollständige Netzwerke

Zunächst definieren wir genauer, was wir unter einem unvollständigen Netzwerk von Zeitautomaten verstehen.

Definition 4.1
Ein ***unvollständiges Netzwerk*** $\mathcal{N}$ *besteht aus einer Menge* $\{TA_0, \ldots, TA_{n-1}\}$ *von Zeit-*

automaten $TA_i = (L_i, l_i^0, Chan_i, X_i, E_i)$ *bzw.* $TA_i = (L_i, l_i^0, Chan_i, X_i, I_i, E_i)$*, zusammen mit einer Menge* $Chan_{BB}$ *von Kanälen und ggf. einer Menge* I_{BB} *von Integervariablen.*

Die Automaten $TA_0, \ldots, TA_{n-1}$ sind diejenigen Automaten des Netzwerks, deren Aufbau bekannt ist. Von den Blackbox-Automaten kennen wir nur ihre Schnittstelle zu den vorliegenden Automaten. Wir kennen weder ihre Orte und Transitionen noch, welche Uhrenvariablen, Integervariablen und Kanäle sie intern verwenden.

Deshalb sind die einzigen bekannten Informationen über die Blackboxes die Menge $Chan_{BB}$ derjenigen Kanäle, mit denen die Blackbox-Automaten mit den Automaten $TA_0, \ldots, TA_{n-1}$ durch Synchronisation kommunizieren, und die Menge I_{BB} der Integervariablen, denen von den Blackbox-Automaten bei einem Sprung neue Werte zugewiesen werden dürfen, um Informationen an die übrigen Automaten zu übermitteln.

Wir setzen $Chan := \bigcup_{i=0}^{n-1} Chan_i$ und $I := \bigcup_{i=0}^{n-1} I_i$. Für die Erzeugung der BMC-Formeln sind nur die Blackbox-Kanäle aus $Chan_{BB} \cap Chan$ und ggf. die Integervariablen aus $I_{BB} \cap I$ relevant. Kanäle und Integervariablen, die nur innerhalb der Blackboxes verwendet werden, können wir ignorieren. Deshalb nehmen wir o. B. d. A. an, dass $Chan_{BB} \subseteq Chan$ und $I_{BB} \subseteq I$ gilt.

Man könnte auch zulassen, dass die Blackbox-Automaten Uhren zurücksetzen dürfen, die von den Automaten $TA_0, \ldots, TA_{n-1}$ gelesen werden. Dies führt aber insbesondere beim Ansatz mit quantifizierten SMT-Formeln dazu, dass reelle Variablen universell quantifiziert werden müssen. Der Solver Lira ist zwar in der Lage, solche Formeln zu lösen, aber die Laufzeit und der Speicherplatzbedarf sind dann so groß, dass für praktisch relevante Systeme kein Ergebnis mehr berechnet werden kann. Deshalb verzichten wir in dieser Arbeit darauf, dass Blackbox-Automaten Uhren zurücksetzen dürfen, die außerhalb der Blackboxes gelesen werden. Bezüglich dem Lesen von Uhren- und Integer-Variablen brauchen wir keine Einschränkungen zu machen.

4.2 SMT-Codierung

Für die Erzeugung der BMC-Formel als SMT-Formel ohne Quantoren verwenden wir folgende Beobachtung: Ist ein Zustand, der die vorgegebene Invariante verletzt, ausschließlich über Transitionen erreichbar, deren Ausführung nicht von der Implementierung der Blackboxes abhängt, dann widerlegt dieser Ablauf die Gültigkeit der Invariante für alle möglichen Blackbox-Implementierungen.

Wir nennen Kanten, deren Ausführbarkeit von der Implementierung der Blackboxes abhängt, *unsichere Kanten*. Da wir einen Zustand, der die Invariante verletzt, unabhängig von den Blackboxes erreichen wollen (also ausschließlich über sichere Kanten), nehmen wir an, dass unsichere Kanten nicht ausgeführt werden dürfen. Indem wir in Formel (3.15) die unsicheren Kanten weglassen, erhalten wir die folgende Formel für die Sprünge:

$$J(s, s') := \bigwedge_{i=0}^{n-1} \left(\mathrm{fix}(TA_i) \vee \bigvee_{\substack{e \in E_i \\ e \text{ sichere Kante}}} \Theta(e, TA_i) \right) \wedge \mathrm{noassign}(s, s'). \tag{4.1}$$

Welche Kanten unsicher sind, hängt von den Kommunikationsmöglichkeiten der Blackboxes ab. Dementsprechend müssen wir die Teilformeln $\mathrm{fix}(TA_i)$, $\Theta(e, TA_i)$ und $\mathrm{noassign}(s, s')$ anpassen. Wir betrachten die Kommunikationsmodelle mit Kanälen und zusätzlich mit Integervariablen. Netzwerke, die nur über gemeinsame Uhrenvariablen kommunizieren, spielen in der Praxis eine geringe Rolle. Man erhält sie als Spezialfall der Kommunikation über Kanäle, indem man in den Formeln $Chan_i = Chan_{BB} = \emptyset$ wählt.

4.2.1 Kommunikation ausschließlich über Kanäle

Die Automaten $TA_0, \ldots, TA_{n-1}$ dürfen nur dann eine Kante ausführen, in deren Kanalmenge ein Kanal $c \in Chan_{BB}$ vorkommt, wenn gleichzeitig alle Blackbox-Automaten, in deren Kanalmenge c enthalten ist, eine entsprechend beschriftete Kante ausführen. Da wir den inneren Aufbau der Blackboxes nicht kennen, wissen wir nicht, ob eine solche Kante im aktuellen Zustand ausgeführt werden kann. Also sind alle Kanten $e = (l, g, C, r, a, l')$ mit $C \cap Chan_{BB} \neq \emptyset$ unsicher.

Da die Werte der Uhrenvariablen nicht von den Blackboxes abhängen können – wir haben angenommen, dass die Blackboxes keine Uhrenvariable zurücksetzen dürfen, die außerhalb der Blackboxes gelesen wird –, hängt die Gültigkeit von Invarianten und Ausführungsbedingungen nicht von den Blackboxes ab. Folglich sind die Kanalbeschriftungen der einzige Grund, warum Kanten unsicher sein können. Bei der Formelerzeugung müssen wir also nur alle Kanten ignorieren, deren Kanalmenge nicht disjunkt zur Menge $Chan_{BB}$ der Blackbox-Kanäle ist.

Die Formeln für den Anfangszustand (Formel (3.7)) und für die Verzögerungsschritte (Formel (3.8)) können wir unverändert übernehmen, da sie unabhängig von Kanälen sind.

In den Formeln $\mathrm{fix}(TA_i)$ (Formel 3.12) und $\Theta(e, TA_i)$ (Formel 3.13) müssen wir alle Blackbox-Kanäle weglassen; der Rest bleibt unverändert:

$$\mathrm{fix}(TA_i) := (at_i' = at_i) \wedge \bigwedge_{x \in X_i} \neg w_{x,i} \wedge \bigwedge_{c \in Chan_i \setminus Chan_{BB}} \neg c \wedge \bigwedge_{l \in L_i} \big(at_i = l \;\Rightarrow\; \mathrm{inv}'(l)\big), \tag{4.2}$$

$$\begin{aligned}\Theta(e, TA_i) := {} & (at_i = l) \wedge (at_i' = l') \wedge g \wedge \mathrm{inv}'(l') \\ & \wedge \bigwedge_{x \in r} (x' = 0 \wedge w_{x,i}) \wedge \bigwedge_{x \in X_i \setminus r} \neg w_{x,i} \wedge \bigwedge_{c \in C \setminus Chan_{BB}} c \wedge \bigwedge_{c \in Chan_i \setminus (C \cup Chan_{BB})} \neg c.\end{aligned} \tag{4.3}$$

Beispiel 4.1
In Abbildung 4.1 ist ein unvollständiges Netzwerk zu sehen. Der vorliegende Automat TA kommuniziert mit den Automaten in der Blackbox mit Hilfe des Kanals a. Deshalb sind alle Kanten, die mit a beschriftet sind, unsicher. Wir haben sie in der Abbildung gestrichelt eingezeichnet. Diese Kanten werden bei der Erzeugung der Formel für die Sprünge vernachlässigt. Da wir nur einen sichtbaren Automaten haben und annehmen, dass die

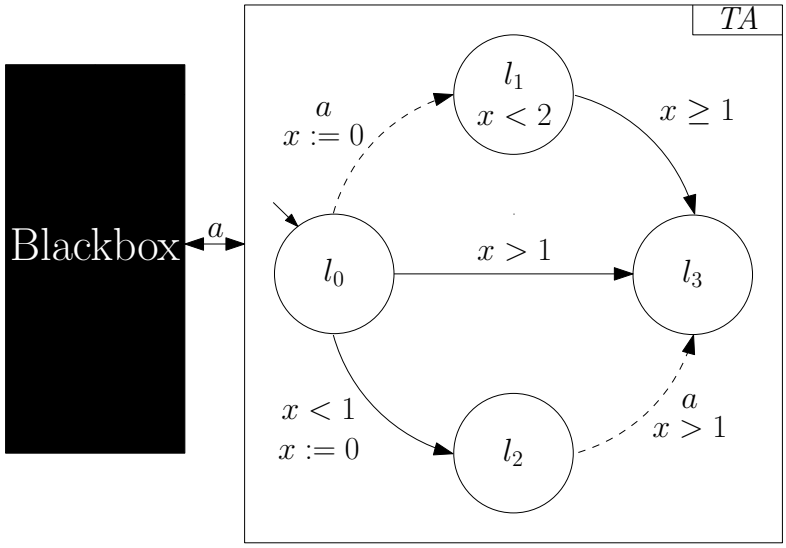

Abbildung 4.1: Unvollständiges Netzwerk von Zeitautomaten

Blackbox die Uhr x nicht zurücksetzen darf, benötigen wir hier keine Schreibindikatoren.

$$
\begin{aligned}
I(s) &:= (at = l_0) \wedge (x = 0) \\
D(s) &:= (at = at') \wedge \big((at' = l_1) \Rightarrow (x' < 2)\big) \wedge (\delta \geq 0) \wedge (x' = x + \delta) \\
\mathrm{fix}(TA) &:= (at = at') \wedge (x' = x) \wedge \big((at' = l_1) \Rightarrow (x' < 2)\big) \\
J(s, s') &:= \mathrm{fix}(TA) \vee \big((at = l_0) \wedge (at' = l_2) \wedge (x < 1) \wedge (x' = 0)\big) \\
&\quad \vee \big((at = l_0) \wedge (at' = l_3) \wedge (x > 1) \wedge (x' = x)\big) \\
&\quad \vee \big((at = l_1) \wedge (at' = l_3) \wedge (x \geq 1)\big) \\
\neg P(s) &:= \big((at = l_3) \wedge (x \leq 1) \wedge (x' = x)\big).
\end{aligned}
$$

Da der einzige Ablauf zu einem unsicheren Zustand über die unsichere Kante von l_0 zu l_1 führt, kann in diesem Fall kein Fehler, der unabhängig von der Blackbox auftritt, gefunden werden. Alle Abwicklungstiefen der BMC-Formel sind unerfüllbar.

4.2.2 Kommunikation über Kanäle und Integervariablen

Nehmen wir an, wir haben ein unvollständiges Netzwerk $\mathcal{N} = \{TA_0, \ldots, TA_{n-1}\}$ mit $TA_i = (L, l^0, Chan_i, X_i, I_i, E_i)$ vorliegen mit den Mengen $Chan_{BB}$ von Blackbox-Kanälen und I_{BB} von Blackbox-Integervariablen. Die Variablen in I_{BB} sind diejenigen, die bei einem Sprung von einem der Blackbox-Automaten geschrieben werden können.

Die Vorgehensweise zur Erzeugung der BMC-Formeln für unvollständige Netzwerke, die zusätzlich zu den Kanälen über gemeinsame Integervariablen kommunizieren, ist analog zum obigen Fall. Erst müssen wir uns überlegen, welche Kanten unsicher sind und ignoriert werden müssen.

Sichere und unsichere Kanten Sei $e = (l, g, C, r, a, l') \in E_i$ eine Kante von TA_i. Sie ist unsicher, wenn eine der folgenden Bedingungen gilt:

- $C \cap Chan_{BB} \neq \emptyset$, d. h. C enthält einen Kanal, der auch in einer Blackbox vorkommt. Die Kante e darf nur dann genommen werden, wenn alle Automaten deren Kanalmenge nicht disjunkt zu C ist, gleichzeitig eine Kante ausführen, deren Kanalmenge die gemeinsamen Kanäle von C und der Kanalmenge ihres Automaten enthält. Insbesondere muss ein Blackbox-Automat eine solche Kante ausführen. Da wir nicht wissen, ob das im aktuellen Zustand möglich ist, ist e unsicher.

- Der Guard g von e hängt von einer Integervariablen $v \in I_{BB}$ ab, die in einer Blackbox geschrieben werden darf.
  Wir wissen nicht, wann der Variablen v welcher Wert aus ihrem Wertebereich $\big[\text{lower}(v), \text{upper}(v)\big]$ zugewiesen wird. Ob der Guard erfüllt ist, hängt somit von der Implementierung der Blackboxes ab. Folglich ist e unsicher.

Formelgenerierung In der Formel für den Anfangszustand ignorieren wir die Initialisierung der Blackbox-Integervariablen. Da die Werte der Blackbox-Variablen nirgendwo mehr gelesen werden, nachdem wir alle unsicheren Kanten entfernt haben, können wir auch alle Zuweisungen an diese Variablen ignorieren, so dass die Variablen aus I_{BB} nicht mehr in der Formel vorkommen.

Ansonsten kann die Formel für den Anfangszustand unverändert übernommen werden.

$$I(s) := \bigwedge_{i=0}^{n-1} (at_i = l_i^0) \wedge \bigwedge_{x \in X} (x = 0) \wedge \bigwedge_{v \in I \setminus I_{BB}} \big(v = \text{init}(v)\big). \tag{4.4}$$

Während die bekannten Automaten $TA_0, \ldots, TA_{n-1}$ einen Verzögerungsschritt ausführen, kann in den Blackbox-Automaten ein Sprung ausgeführt werden, der den Variablen aus I_{BB} einen neuen Wert zuweist. Deshalb können wir in der Formel für Verzögerungsschritte für alle Variablen $v \in I_{BB}$ die Zuweisung $v' = v$ weglassen, da diese Variablen nach dem Verzögerungsschritt nicht mehr dieselben Werte haben müssen, wie davor. Außerdem gilt auch hier, dass die Variablen aus I_{BB} nicht mehr gelesen werden, so dass wir ihnen auch keinen Wert zuweisen müssen.

$$\begin{aligned} D(s, s') := \bigwedge_{i=0}^{n-1} \Bigg((at_i = at_i') \wedge \bigwedge_{l \in L_i} \Big(at_i = l \;\Rightarrow\; \text{inv}_i'(l) \Big) \Bigg) \\ \wedge\, \delta \geq 0 \wedge \bigwedge_{x \in X} (x' = x + \delta) \wedge \bigwedge_{v \in I \setminus I_{BB}} (v' = v). \end{aligned} \tag{4.5}$$

In der Formel $\text{fix}(TA_i)$ benötigen wir weder Integervariablen aus I_{BB} noch Kanäle aus $Chan_{BB}$. Für Integervariablen $v \in I_{BB}$, die in Blackboxes geschrieben werden, benötigen wir keinen Schreibindikator. Wir müssen nämlich ständig damit rechnen, dass der Variable von einer Blackbox ein neuer Wert zugewiesen wird. Deshalb dürfen wir nicht verlangen,

dass $v = v'$ gilt.

$$\begin{aligned}\mathrm{fix}(TA_i) := (at_i' = at_i) \wedge \bigwedge_{x \in X_i} \neg w_{x,i} \wedge \bigwedge_{v \in I_i \setminus I_{BB}} \neg w_{v,i} \\ \wedge \bigwedge_{c \in Chan_i \setminus Chan_{BB}} \neg c \wedge \bigwedge_{l \in L_i} \big(at_i = l \;\Rightarrow\; \mathrm{inv}'(l)\big)\end{aligned} \tag{4.6}$$

Sei $e = (l, g, C, r, a, l') \in E_i$ eine sichere Kante von TA_i. Mit $I \setminus (I_{BB} \cup a)$ bezeichnen wir die Menge der Integervariablen, die nicht in einer Blackbox geschrieben werden und denen beim Laufen der Kante e kein neuer Wert zugewiesen wird, also

$$I \setminus (I_{BB} \cup a) = \{v \in I \mid v \notin I_{BB} \wedge \neg\exists \varphi \in \Psi(I) : (v, \varphi) \in a\}.$$

Dann sieht die Codierung von e folgendermaßen aus:

$$\begin{aligned}\Theta(e, TA_i) := (at_i = l) \wedge (at_i' = l') \wedge g \wedge \mathrm{inv}'(l') \\ \wedge \bigwedge_{x \in r} (x' = 0 \wedge w_{x,i}) \wedge \bigwedge_{x \in X_i \setminus r} \neg w_{x,i} \\ \wedge \bigwedge_{(v,\varphi) \in a \wedge v \notin I_{BB}} (v' = \varphi(I) \wedge w_{v,i}) \wedge \bigwedge_{v \in I_i \setminus (I_{BB} \cup a)} \neg w_{v,i} \\ \wedge \bigwedge_{c \in C} c \wedge \bigwedge_{c \in Chan_i \setminus (C \cup Chan_{BB})} \neg c.\end{aligned} \tag{4.7}$$

Im Unterschied zur Formel (3.19) werden überall die Integervariablen aus I_{BB} und die Kanäle aus $Chan_{BB}$ ignoriert. Dasselbe gilt für die Formel noassign(s, s'):

$$\begin{aligned}\mathrm{noassign}(s, s') := \bigwedge_{x \in X} \Bigg(\Big(\bigwedge_{\substack{i \in \{0,\ldots,n-1\} \\ x \in X_i}} \neg w_{x,i} \Big) \Rightarrow \Big(x' = x \Big) \Bigg) \\ \wedge \bigwedge_{v \in I \setminus I_{BB}} \Bigg(\Big(\bigwedge_{\substack{i \in \{0,\ldots,n-1\} \\ v \in I_i}} \neg w_{v,i} \Big) \Rightarrow \Big(v' = v \Big) \Bigg).\end{aligned} \tag{4.8}$$

Aus diesen Bausteinen können wir die Formel $J(s, s')$ wie in Formel (4.1) zusammensetzen und mit deren Hilfe die BMC-Formel (3.6) für beliebige Abwicklungstiefen erstellen.

4.3 SMT-Codierung mit Quantoren

Es gibt unvollständige Netzwerke von Zeitautomaten mit Integervariablen, in denen für alle möglichen Ersetzungen der Blackbox eine Invarianteneigenschaft verletzt ist, für die der Ansatz mit SMT-Formeln ohne Quantoren aber trotzdem kein Gegenbeispiel findet. Um dies zu verdeutlichen, betrachten wir folgendes Beispiel:

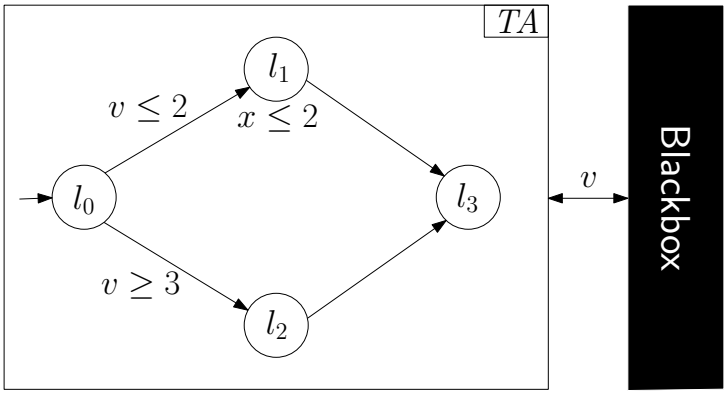

Abbildung 4.2: Gegenbeispiel für die Unerreichbarkeit von l_3

Beispiel 4.2
Betrachten wir den Zeitautomaten aus Abbildung 4.2 zusammen mit der Menge $I_{BB} = \{v\}$ und dem Wertebereich $0 \leq v \leq 5$. Wir wollen prüfen, ob die Invarianteneigenschaft $P := \neg(at = l_3)$ unabhängig von der Implementierung der Blackbox verletzt ist.

Offensichtlich ist das der Fall. Falls $v \leq 2$ erfüllt ist, dann führt der folgende Ablauf zum Zielort l_3:

$$(l_0, x = 0) \rightarrow (l_1, x = 0) \rightarrow (l_3, x = 0).$$

Andernfalls ist $v \geq 3$. Dann können wir den Weg über den Ort l_2 nehmen:

$$(l_0, x = 0) \rightarrow (l_2, x = 0) \rightarrow (l_3, x = 0).$$

Verwenden wir den Ansatz mit quantorenfreien SMT-Formeln, dann wären die beiden ausgehenden Kanten von l_0 unsicher, da ihre Ausführungsbedingung von v abhängt und v von der Blackbox geschrieben werden darf. Der Ort l_0 hat damit keine ausgehende sichere Kante mehr. Damit ist es auch nicht möglich, über sichere Kanten in den Zielort zu kommen. Somit sind alle Abwicklungstiefen bei BMC mit quantorenfreien SMT-Formeln unerfüllbar.

Wie das Beispiel gezeigt hat, gibt es unvollständige Netzwerke, in denen unabhängig von der Implementierung der Blackboxes eine Invarianteneigenschaft verletzt ist, für die aber der SMT-Ansatz trotzdem keinen Fehler nachweisen kann. Abhilfe schaffen quantifizierte SMT-Formeln. Damit können wir nachweisen, dass es für jeden Wert der Integervariablen, die in den Blackboxes geschrieben werden dürfen, einen Pfad zu einem Zustand gibt, der die Invarianteneigenschaft verletzt.

Beispiel 4.3
Betrachten wir nochmal den Zeitautomaten aus Beispiel 4.2 und die quantifizierte SMT-Formel, bei der wir, um die Formel übersichtlich zu halten, die Verzögerungsschritte

weggelassen haben:

$$
\begin{aligned}
&\exists at^0 \exists x^0 \forall i^0 \exists at^1 \exists x^1 \forall i^1 \exists at^2 \exists i^2 \forall i^2 : \\
&\quad (at^0 = l_0 \wedge x^0 = 0) \wedge \\
&\quad \big((at^0 = l_0 \wedge i^0 \leq 2 \wedge x^1 = x^0 \wedge x^1 \leq 2 \wedge at^1 = l_1) \vee \\
&\qquad (at^0 = l_0 \wedge i^0 \geq 3 \wedge x^1 = x^0 \wedge at^1 = l_2) \vee \\
&\qquad (at^0 = l_1 \wedge x^1 = x^0 \wedge at^1 = l_3) \vee \\
&\qquad (at^0 = l_2 \wedge x^1 = x^0 \wedge at^1 = l_3)\big) \\
&\quad \wedge \big((at^1 = l_0 \wedge i^1 \leq 2 \wedge x^2 = x^1 \wedge x^2 \leq 2 \wedge at^2 = l_1) \vee \\
&\qquad (at^1 = l_0 \wedge i^1 \geq 3 \wedge x^2 = x^1 \wedge at^2 = l_2) \vee \\
&\qquad (at^1 = l_1 \wedge x^2 = x^1 \wedge at^2 = l_3) \vee \\
&\qquad (at^1 = l_2 \wedge x^2 = x^1 \wedge at^2 = l_3)\big) \\
&\quad \wedge (at^2 = l_3).
\end{aligned}
$$

Die Formel ohne das Quantorenpräfix beschreibt zunächst mit $(at^0 = l_0 \wedge x^0 = 0)$ *den Anfangszustand und danach zwei Abwicklungen der Transitionsrelation. Zuletzt folgt die negierte Invariante, nämlich* $at^2 = l_3$.

Das Quantorenpräfix, in dem die Instanzen i^0, i^1 *und* i^2 *der Integervariablen* i*, die von der Blackbox geschrieben werden darf, universell quantifiziert sind, sorgt dafür, dass eine erfüllende Belegung für alle möglichen Werte der Integervariablen gefunden werden muss. Dabei müssen die Variablen im Präfix nach Abwicklungstiefe sortiert vorkommen, da der Werte einer Variable nicht von späteren Abwicklungstiefen abhängen darf. Die Integervariable* v *wird nach den Variablen* at *und* x *quantifiziert, da der Wert von* v *zwar von* at *und* x *abhängen darf, nicht jedoch umgekehrt der aktuellen Ort von der Wahl des Wertes für* v.

Diese Formel ist gültig, denn die Klausel $(at^0 = l_0 \wedge x^0 = 0)$ *hängt nicht von den Integervariablen ab. Beim ersten Sprung erfüllen wir entweder* $(at^0 = l_0 \wedge i^0 \leq 2 \wedge x^1 = x^0 \wedge x^1 \leq 2 \wedge at^1 = l_1)$*, falls* $i^0 \leq 2$ *ist, und kommen in den Ort* l_1*, oder* $(at^0 = l_0 \wedge i^0 \geq 3 \wedge x^1 = x^0 \wedge at^1 = l_2)$*, falls* $i^0 > 2$ *ist. Damit kommen wir nach* l_2*. In Abhängigkeit davon wählen wir beim nächsten Sprung entweder* $(at^1 = l_1 \wedge x^2 = x^1 \wedge at^2 = l_3)$ *oder* $(at^1 = l_2 \wedge x^2 = x^1 \wedge at^2 = l_3)$*. Beide sind unabhängig von den universell quantifizierten Integervariablen und führen uns in den Ort* l_3*. Damit ist auch die negierte Invariante erfüllt.*

Folglich gibt es für jede mögliche Implementierung der Blackbox einen Pfad, auf dem die Invariante verletzt wird.

Wir werden im Folgenden zeigen, wie man quantifizierte SMT-Formel für BMC erstellen kann, mit denen mehr Fehler in Systemen gefunden werden können als mit quantorenfreien SMT-Formeln.

Wir nehmen dazu an, dass wir ein unvollständiges Netzwerk $\mathcal{N} = \{ TA_0, \ldots, TA_{n-1} \}$ von Zeitautomaten $TA_i = (L_i, l_i^0, Chan_i, X_i, I_i, \mathrm{inv}_I, E_i)$ vorliegen haben. Die Menge $Chan_{BB}$

enthalte wieder alle Kanäle, die von den Blackboxes verwendet werden, und I_{BB} die Menge aller Integervariablen, die in den Blackboxes geschrieben werden dürfen. Wir setzen $X := \bigcup_{i=0}^{n-1} X_i$, $I := \bigcup_{i=0}^{n-1} I_i$ und $\mathit{Chan} := \bigcup_{i=0}^{n-1} \mathit{Chan}_i$.

Um die Notation des Quantorenpräfixes zu vereinfachen, führen wir folgende Notationen ein: Ist $S = \{s_0, \ldots, s_{m-1}\}$ eine Menge von Variablen, so schreiben wir $\exists S$ für $\exists s_0 \exists s_1 \ldots \exists s_{m-1}$ und $\forall S$ für $\forall s_0 \forall s_1 \ldots \forall s_{m-1}$. Zu einer Menge S von Variablen bezeichne S^k die Instanzen der enthaltenen Variablen für die Abwicklungstiefe k.

$$W := \{w_{x,i} \,|\, x \in X_i\} \cup \{w_{v,i} \,|\, v \in (I_i \setminus I_{BB})\}$$

sei die Menge der Schreibindikatoren für Uhren- und Integervariablen.

Im Quantorenpräfix müssen wir nacheinander alle Variablen mit einem Quantor versehen. Alle Blackbox-Integervariablen werden universell quantifiziert, da wir Fehler finden wollen, die unabhängig vom Wert der Blackbox-Integervariablen sind. Alle anderen Variablen werden existentiell quantifiziert. Außerdem sortieren wir die Variablen nach Abwicklungstiefe und beginnen mit der niedrigsten. Innerhalb einer Abwicklungstiefe werden zuerst alle Variablen, die den aktuellen Zustand des Systems codieren, existentiell quantifiziert, danach die Blackbox-Integervariablen universell. Da die ausgehenden Kanten in Abhängigkeit der Werte der Blackbox-Integervariablen gewählt werden dürfen, werden die Variablen, die die Kanäle codieren, nach den Blackbox-Integervariablen quantifiziert; ebenso die Schreibindikatoren, deren Werte von den ausgeführten Kanten abhängen.

Wie bei den Formeln im vorigen Kapitel müssen wir auch hier die Wertebereiche der Integervariablen berücksichtigen. Für die existentiell quantifizierten Variablen kann dies genauso wie dort geschehen, indem bei der Transitionsformel gefordert wird, dass der Wert der Integervariablen zwischen der unteren und oberen Schranke liegt (siehe unten); bei den universell quantifizierten erfolgt die Berücksichtigung des Wertebereichs auf andere Art. Ist $v \in I_{BB}$ universell quantifiziert, so müssen wir dafür sorgen, dass die Formel für alle Werte außerhalb des Wertebereichs von v erfüllt ist; ansonsten wäre die BMC-Formel automatisch ungültig. Dies erreichen wir, indem wir vor den Quantifizierung die Disjunktion von $\big(v < \mathrm{lower}(v) \vee v > \mathrm{upper}(v)\big)$ mit der übrigen Formel bilden.

$$\mathrm{range}_{BB}(s) = \bigvee_{v \in (I^0_{BB} \cup I^1_{BB} \cup \cdots \cup I^k_{BB})} \big(v < \mathrm{lower}(v) \vee v > \mathrm{upper}(v)\big). \tag{4.9}$$

Damit erhält die Formel folgende Struktur:

$$\begin{aligned} \mathit{BMC}(k) = {} & \exists \mathit{at}^0 \, \exists X^0 \, \forall I^0_{BB} \, \exists (I \setminus I_{BB})^0 \, \exists (\mathit{Chan} \setminus \mathit{Chan}_{BB})^0 \, \exists \delta^0 \, \exists W^0 \\ & \exists \mathit{at}^1 \, \exists X^1 \, \forall I^1_{BB} \, \exists (I \setminus I_{BB})^1 \, \exists (\mathit{Chan} \setminus \mathit{Chan}_{BB})^1 \, \exists \delta^1 \, \exists W^1 \\ & \ldots \\ & \exists \mathit{at}^k \, \exists X^k \, \forall I^k_{BB} \, \exists (I \setminus I_{BB})^k \, \exists (\mathit{Chan} \setminus \mathit{Chan}_{BB})^k \, \exists \delta^k \, \exists W^k : \\ & \qquad \mathrm{range}_{BB}(s^0, \ldots, s^k) \vee \Big(I(s^0) \wedge \bigwedge_{i=0}^{k-1} T(s^i, s^{i+1}) \wedge \neg P(s^k) \Big). \end{aligned} \tag{4.10}$$

Für die Erstellung der eigentlichen Formel sind gegenüber der quantorenfreien Formulierung der BMC-Formel vor allem bei den Sprüngen einige Änderungen notwendig. Denn unser Ziel war es dort, *einen* Ablauf zu finden, der für alle möglichen Werte der Blackbox-Integervariablen zu einem unsicheren Zustand führt. Deswegen mussten wir für die quantorenfreien Formeln auf alle Kanten verzichten, deren Ausführung durch die Blackboxes verhindert werden konnte.

Da wir jetzt die Blackbox-Integervariablen universell quantifizieren, müssen wir nicht einen einzelnen Pfad zu einem unsicheren Zustand finden, der unabhängig von den Werten der Blackbox-Integervariablen ist. Stattdessen genügt es, für jeden möglichen Wert der Blackbox-Integervariablen einen Ablauf zu finden, der von den Werten der Integervariablen abhängen darf. Deshalb müssen wir nicht mehr auf Kanten verzichten, deren Ausführungsbedingung von einer Blackbox-Integervariable abhängt.

Anders sieht es bei den Kanälen aus. Wir könnten die Blackbox-Kanäle aus $Chan_{BB}$ ebenfalls universell quantifizieren. Dies bringt aber gegenüber dem Entfernen der Kanten, die mit einem Blackbox-Kanal beschriftet sind, keine Vorteile: Nehmen wir an, c sei eine boolesche Variable für einen Kanal, die wir universell quantifizieren. Wir müssen die Formel also sowohl für $c = \top$ als auch für $c = \bot$ erfüllen. Im Falle $c = \bot$ dürfen wir alle Kanten, die mit dem Kanal c beschriftet sind, nicht verwenden, um einen unsicheren Zustand zu erreichen. Finden wir für $c = \bot$ einen Pfad zu einem unsicheren Zustand, kann dieser jedoch unabhängig von der Blackbox gegangen werden, und wir benötigen keinen eigenen Pfad für den Fall $c = \top$. Umgekehrt gilt dies nicht: Finden wir einen Pfad mit $c = \top$, so ist dieser abhängig von der Blackbox, und wir müssen trotzdem noch einen Pfad für $c = \bot$ finden.

Somit sind bei der Erzeugung quantifizierter SMT-Formeln nur diejenigen Kanten als unsicher anzusehen, die mit einem Kanal aus $Chan_{BB}$ beschriftet sind.

Außerdem sind wie auch schon bei der Erstellung von quantorenfreien Formeln folgende Punkte zu beachten:

- Blackbox-Integervariablen benötigen keinen Schreibindikator, da wir zu jedem Zeitpunkt annehmen müssen, dass eine Blackbox der Variablen einen neuen Wert zuweist.
- Alle Zuweisungen an Blackbox-Integervariablen können vernachlässigt werden, da in den Blackboxes diesen Variablen jederzeit ein anderer Wert zugewiesen werden kann.

Darüber hinaus müssen wir noch dem Wertebereich aller Integer-Variablen, die existentiell quantifiziert sind, berücksichtigen; für die universell quantifizierten Variablen haben wir dies bereits oben getan, vgl. Formel (4.10). Dazu stellen wir wie bereits bei vollständigen Netzwerken die Formel range(s) auf, die allerdings hier nur die Integervariablen enthält, die nicht von Blackboxes geschrieben werden.

$$\text{range}(s) := \bigwedge_{v \in I \setminus I_{BB}} \big(v \geq \text{lower}(v) \wedge v \leq \text{upper}(v)\big) \tag{4.11}$$

Sie wird in der Transitionsformel $T(s, s')$ verwendet.

Damit ergeben sich insgesamt die folgenden Formeln, die wir der Vollständigkeit halber angeben, auch wenn sie ohne Änderung von der quantorenfreien Formulierung der BMC-Formel übernommen werden können:

Anfangszustand:

$$I(s) := \bigwedge_{i=0}^{n-1} (at_i = l_i^0) \wedge \bigwedge_{x \in X} (x = 0) \wedge \bigwedge_{v \in I \setminus I_{BB}} \big(v = \text{init}(v)\big), \tag{4.12}$$

Verzögerungsschritte:

$$\begin{aligned} D(s, s') := \bigwedge_{i=0}^{n-1} \Bigg((at_i = at_i') \wedge \bigwedge_{l \in L_i} \Big(at_i = l \;\Rightarrow\; \text{inv}_i'(l) \Big) \Bigg) \\ \wedge\, \delta \geq 0 \wedge \bigwedge_{x \in X} (x' = x + \delta) \wedge \bigwedge_{v \in I \setminus I_{BB}} (v' = v), \end{aligned} \tag{4.13}$$

Sprünge:

$$J(s, s') := \bigwedge_{i=0}^{n-1} \Bigg(\text{fix}(TA_i) \vee \bigvee_{\substack{e \in E_i \\ e \text{ sicher}}} \Theta(e, TA_i) \Bigg) \wedge \text{noassign}(s, s'),$$

mit folgenden Teilformeln:

$$\begin{aligned} \text{fix}(TA_i) := (at_i' = at_i) \wedge \bigwedge_{x \in X_i} \neg w_{x,i} \wedge \bigwedge_{v \in I_i \setminus I_{BB}} \neg w_{v,i} \\ \wedge \bigwedge_{c \in Chan_i \setminus Chan_{BB}} \neg c \wedge \bigwedge_{l \in L_i} \big(at_i = l \;\Rightarrow\; \text{inv}'(l) \big), \end{aligned} \tag{4.14}$$

$$\begin{aligned} \Theta(e, TA_i) := (at_i = l) \wedge (at_i' = l') \wedge g \wedge \text{inv}'(l') \\ \wedge \bigwedge_{x \in r} (x' = 0 \wedge w_{x,i}) \wedge \bigwedge_{x \in X_i \setminus r} \neg w_{x,i} \\ \wedge \bigwedge_{(v,\varphi) \in a \wedge v \notin I_{BB}} (v' = \varphi(I) \wedge w_{v,i}) \wedge \bigwedge_{v \in I_i \setminus (I_{BB} \cup a)} \neg w_{v,i} \\ \wedge \bigwedge_{c \in C} c \wedge \bigwedge_{c \in Chan_i \setminus (C \cup Chan_{BB})} \neg c, \end{aligned} \tag{4.15}$$

$$\begin{aligned} \text{noassign}(s, s') := \bigwedge_{x \in X} \Bigg(\Big(\bigwedge_{\substack{i \in \{0,\ldots,n-1\} \\ x \in X_i}} \neg w_{x,i} \Big) \Rightarrow \Big(x' = x \Big) \Bigg) \\ \wedge \bigwedge_{v \in I \setminus I_{BB}} \Bigg(\Big(\bigwedge_{\substack{i \in \{0,\ldots,n-1\} \\ v \in I_i}} \neg w_{v,i} \Big) \Rightarrow \Big(v' = v \Big) \Bigg). \end{aligned} \tag{4.16}$$

Transitionsrelation, alternierende Variante:

$$T(s, s') := \begin{cases} D(s, s') \wedge \text{range}(s') & \text{für gerade Abwicklungstiefen,} \\ J(s, s') \wedge \text{range}(s') & \text{für ungerade Abwicklungstiefen,} \end{cases} \tag{4.17}$$

Transitionsrelation, nicht-alternierende Variante:

$$T(s, s') := \big(D(s, s') \vee J(s, s')\big) \wedge \text{range}(s'). \tag{4.18}$$

Im nächsten Kapitel werden wir die Ansätze zur Fehlersuche in unvollständigen Netzwerken von Zeitautomaten, die wir in diesem und im vorherigen Kapitel entwickelt haben, experimentell anhand einer Fallstudie evaluieren.

5 Experimente

Wir haben die in den vorherigen Kapiteln beschriebenen Techniken in einem Tool namens `t-bounce` implementiert. Zum Einlesen der Beispielmodelle verwenden wir den Parser des Modellprüfers UPPAAL [13], als Solver für die SMT-Formeln stehen YICES [10] und MATHSAT [9] zur Verfügung, zum Lösen von quantifizierten Formeln YICES und LIRA. Zu beachten ist, dass die Bibliotheksschnittstelle von YICES keine Quantoren unterstützt. Deshalb übergeben wir die Formeln in diesem Fall per Dateiaustausch an YICES. Der Eindeutigkeit halber bezeichnen wir den YICES-Solver in diesem Fall als YICESFILE-Solver. Auch bei LIRA[5] wird die zu prüfende Formel per Dateiaustausch übergeben.

Wir haben alle Experimente auf einem AMD Opteron DualCore Prozessor mit 2.4 GHz Taktfrequenz und 4 GB Hauptspeicher unter Kubuntu Linux Version 10.04 durchgeführt. Die Implementierung erfolgte in der Sprache C++. Als Compiler kam gcc in der Version 4.4.3 zum Einsatz. Für alle Benchmarks haben wir ein Zeitlimit von 10 Stunden und ein Speicherlimit von 3 GB gesetzt.

5.1 Benchmarks

Für die durchgeführten Experimente haben wir das *Fischer-Protokoll* [12] benutzt, bei dem mehrere Prozesse auf eine gemeinsame Ressource zugreifen möchten. Dabei soll der wechselseitige Ausschluss sichergestellt werden, so dass zu jedem Zeitpunkt höchstens ein Prozess auf die Ressource zugreifen kann.

Wir modellieren jeden Prozess durch einen eigenen Zeitautomaten, die alle zusammen ein Netzwerk bilden. Wir modellieren das Fischer-Protokoll auf zwei verschiedene Arten: Zum einen erlauben wir nur Kommunikation über Kanäle (vgl. Kapitel 2.1.4; im Folgenden *FChan* genannt). Um den wechselseitigen Ausschluss zu gewährleisten, fügen wir in das Netzwerk einen zusätzlichen Zeitautomaten ein, den wir *Synchronisationsautomat* nennen. Zum anderen untersuchen wir Netzwerke von Zeitautomaten, die nur über Integervariablen kommunizieren (im Folgenden *FInt* genannt). Dann benötigen wir keinen zusätzlichen Synchronisationsautomaten.[1]

In beiden Fällen haben alle Zeitautomaten für die einzelnen Prozesse die gleiche Anzahl von Orten, die gleiche Anzahl und Positionierung von Kanten mit den selben Ausführungsbedingungen, die gleichen Invarianten und Uhrenvariablen. Jede Menge der Uhrenvariablen eines Automaten ist zu den Mengen der Uhrenvariablen der anderen Zeitautomaten disjunkt. Zusätzlich haben die Automaten bei FChan die gleiche Anzahl

[1] Dies kann als Spezialfall von Netzwerken von Zeitautomaten, die über Kanäle *und* Integervariablen kommunizieren (vgl. Kapitel 2.1.4), angesehen werden, bei dem die Menge der Kanäle *Chan* leer ist.

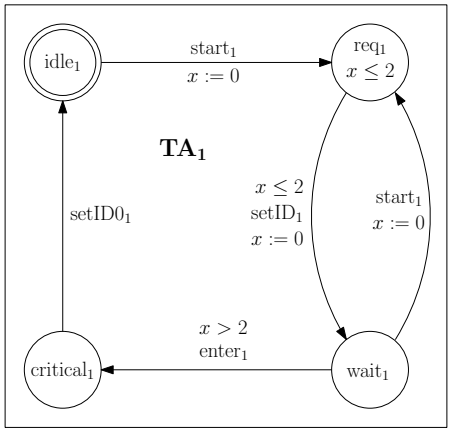

Abbildung 5.1: Fischer-Protokoll, ein einzelner Automat

und Positionierung von Kanälen, bzw. bei FInt eine gemeinsame Integervariable, die bei FChan durch den Synchronisationsautomaten simuliert wird.

Betrachten wir zunächst den Zeitautomaten eines einzelnen Prozesses in Abbildung 5.1. Der Automat hat vier Zustände: *idle* – der Startzustand –, *req*, *wait* und *critical*. Im Zustand *req* fordert der Prozess den Zugriff auf den kritischem Bereich an und wartet auf eine positive Rückmeldung in *wait*. Schließlich beschreibt *critical* den Zustand, in dem der Prozess im kritischen Bereich ist. Der Automat besitzt eine Uhr x sowie die Kanäle $start_1$, $setID0_1$, $setID_1$ und $enter_1$.

Im nächsten Schritt vereinigen wir drei derartige Automaten (und zusätzlich einen Synchronisationsautomaten) zu einem Netzwerk (siehe Abbildung 5.2). Einer der drei Netzwerkautomaten, die einen Prozess beschreiben, ist in der Blackbox, so dass wir nur die Namen seiner Kanäle kennen, nicht aber, ob sie zu einem bestimmten Zeitpunkt aktiv sind. Nach der Initialisierung befindet sich jeder Automaten in seinem jeweiligen Startzustand *idle* und alle Uhren haben den Wert Null.

Die zu prüfende Eigenschaft für dieses Netzwerk ist, dass zu jedem Zeitpunkt höchstens $p - 1$ Zeitautomaten sich im kritischen Bereich befinden. Da der Aufbau der Automaten identisch ist, können wir o. B. d. A. die Eigenschaft so formulieren, dass sich die ersten p Prozesse nicht gleichzeitig im kritischen Bereich aufhalten. Diese Automaten bezeichnen wir im Folgenden als *Property-Automaten*. Es wurde in jedem der Automaten, die einen Prozess beschreiben, je ein Fehler an der gleichen Stelle eingebaut (siehe graue Einkreisungen in Abb. 5.2), der dazu führt, dass diese Eigenschaft verletzt werden kann. Es gilt herauszufinden, ob und wie schnell dieser Modellfehler mit unterschiedlichen Ansätzen gefunden werden kann. Als Ausgabe erhalten wir ein Gegenbeispiel in Form einer Codierung eines Pfades, der ausgehend vom Initialzustand des Netzwerkes zu einem Zustand führt, der die Eigenschaft verletzt.

Für unsere Experimente haben wir anhand des Fischer-Protokolls die Netzwerke mit n Netzwerkautomaten ($n \in \{2, 3, 4, 5, 10, 15, 20\}$) erzeugt. Auch die maximale Anzahl der Property-Automaten p haben wir variiert ($p \in \{2, 3, 4\}$). Für jede Kombination von n und p haben wir unterschiedliche Experimente durchgeführt, wobei $n \geq p$ immer gelten

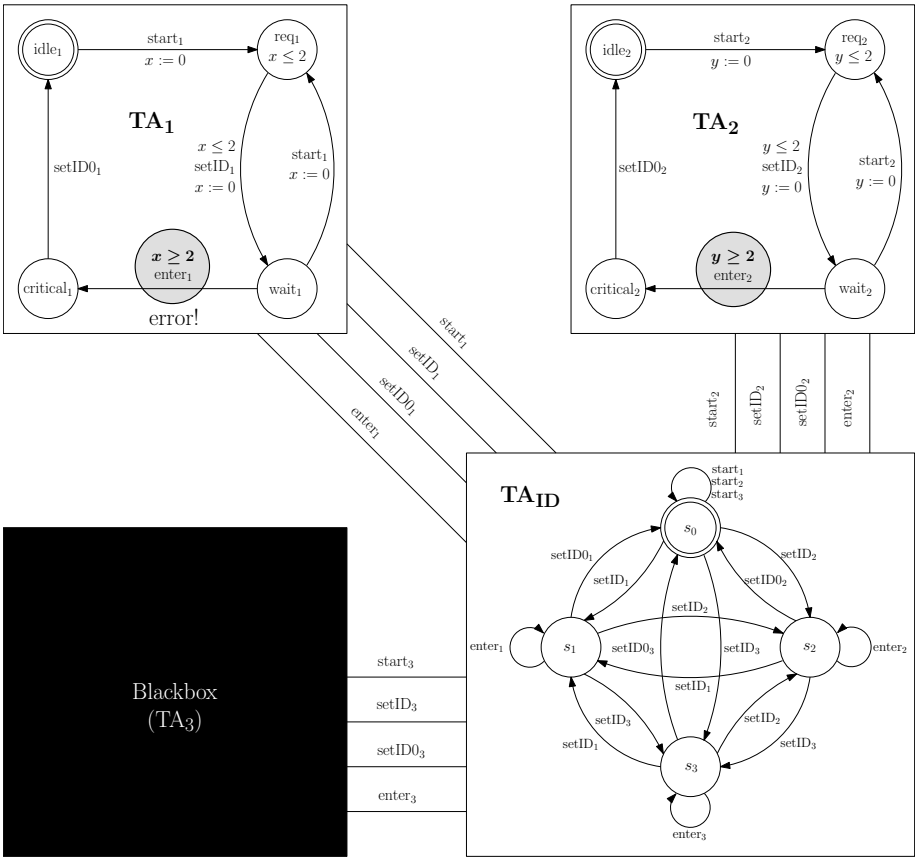

Abbildung 5.2: Fischer-Protokoll, ein Netzwerk

muss. In den Experimenten mit Blackboxes werden immer die Automaten, die keine Property-Automaten sind, in jeweils eine Blackbox gesetzt (also $n - p$ Blackboxes).

Im ersten Teil der Experimente mit FChan haben wir Laufzeit und Speicherverbrauch bei vier verschiedenen Ansätzen gegeneinander verglichen – alternierend bzw. nicht-alternierend bei inkrementellem bzw. nicht-inkrementellem Modus.

Im inkrementellen Modus merken wir uns nach jeder Abrolltiefe der BMC-Formel das Zwischenergebnis und verwenden es wieder bei den folgenden Abrolltiefen. Beim nicht-inkrementellen Modus wird die BMC-Formel für jede Abrolltiefe neu aufgestellt. Der alternierende bzw. nicht-alternierende Modus lässt in jedem Zeitschritt abwechselnd ausschließlich Sprünge oder ausschließlich Verzögerungsschritte zu.

All diese Varianten werden mit Yices und MathSat, sowohl mit als auch ohne Blackboxes, gemessen.

Im zweiten Teil unserer Experimente haben wir die gleiche Ansätze nochmal betrachtet, diesmal mit FInt. In diesem Fall verzichteten wir auf den Einsatz von Blackboxes, weil es nicht mehr möglich wäre, ein Gegenbeispiel zu finden. Alle Kanten, an denen eine Ausführungsbedingung steht, die über Integervariablen spricht, müssen als „unsicher“ markiert werden. Da sich alle Prozesse eine Integervariable teilen – also explizit auch

		inkr., alt.			nicht inkr., alt.			inkr., nicht alt.			nicht inkr., nicht alt.		
n	p	Tiefe	Zeit	Speicher	Tiefe	Zeit	Speicher	Tiefe	Zeit	Speicher	Tiefe	Zeit	Speicher
2	2	12	0.02	10.67	12	0.10	10.67	8	0.03	10.87	8	0.11	10.89
3	2	12	0.03	11.12	12	0.15	11.13	8	0.06	11.47	8	0.17	11.50
	3	26	1.42	13.13	26	2.28	13.15	17	4.06	15.54	17	3.89	14.68
4	2	12	0.05	11.78	12	0.26	11.78	8	0.09	12.32	8	0.26	12.33
	3	26	3.84	15.16	26	5.72	15.09	17	15.54	19.26	17	12.53	18.54
	4	40	375.46	27.04	40	404.63	24.14	26	1117.50	39.68	26	903.27	34.41
5	2	12	0.07	12.51	12	0.35	12.49	8	0.11	13.33	8	0.40	13.25
	3	26	12.18	18.63	26	11.73	17.20	17	29.44	25.47	17	42.96	24.21
	4	40	3265.03	52.82	40	3012.44	39.63	26	7986.97	78.86	26	6818.79	67.56
10	2	12	0.26	19.97	12	1.47	20.14	8	0.44	23.72	8	1.57	23.53
	3	26	59.96	41.35	26	76.18	39.34	17	93.90	52.76	17	131.13	54.36
	4	40	11482.00	99.97	40	21533.30	106.50	– TIMEOUT –			– TIMEOUT –		
15	2	12	0.69	36.20	12	4.22	36.53	8	1.28	47.77	8	4.66	47.86
	3	26	109.25	80.85	26	198.44	83.81	17	208.33	113.44	17	273.43	113.09
	4	40	18779.00	180.59	40	18886.00	162.75	– TIMEOUT –			– TIMEOUT –		
20	2	12	1.52	64.61	12	9.07	64.75	8	2.42	89.39	8	9.59	89.80
	3	26	333.86	153.51	26	434.18	152.09	17	397.82	217.26	17	466.99	218.46
	4	40	34030.30	316.91	– TIMEOUT –			– TIMEOUT –			– TIMEOUT –		

Tabelle 5.1: Vergleich von Kombination aus inkrementellem/nicht-inkrementellem und alternierendem/nicht-alternierendem Modus *ohne Blackboxes* beim Fischer-Protokoll mit Synchronisation über Kanäle und YICES als Solver.

die, die in der Blackbox stehen – werden alle Pfade, die zu einem Gegenbeispiel führen könnten, unsicher.

Als letztes wollen wir zwei weitere SMT-Solver-Varianten mit Quantifizierung verwenden: YICESFILE und LIRA. In diesem Fall konnten wir nur alternierende und nicht-alternierende Alternativen im nicht-inkrementellen Modus vergleichen, da die inkrementelle Vorgehensweise von diesen Solver-Varianten nicht unterstützt wird.

5.2 Ergebnisse

Die nachfolgenden Tabellen präsentieren die Ergebnisse der Evaluierung anhand von Benchmarks auf der Basis des Fischer-Protokolls wie zuvor beschrieben.

Beim Aufbau der Tabellen haben wir folgende Abkürzungen benutzt: *inkr.* steht für die Benutzung des inkrementellen Modus, *nicht inkr.* für den nicht-inkrementellen Modus. Analog steht *alt.* für die alternierende und *nicht alt.* für die nicht-alternierenden Variante der BMC-Formeln. Die ersten Spalten geben die Anzahl n der Zeitautomaten im Netzwerk und die Anzahl p der Property-Automaten an. Die Abrolltiefe, in der das Gegenbeispiel gefunden wurde bzw. die Lösungsroutine abgebrochen wurde, ist in der Spalte „Tiefe“angegeben. Die verbrauchte Laufzeit wurden in Sekunden und der verbrauchte Speicher in Megabyte gemessen. Bei der Überschreitung der festgelegten Zeitbegrenzung von 10 Stunden haben wir „– TIMEOUT –“ als Ausgabe eingetragen.

Die Tabellen 5.1 und 5.2 geben die Ergebnisse für die Variante FChan mit YICES, die Tabellen 5.3 und 5.4 mit MATHSAT an, jeweils mit und ohne Benutzung von Blackboxes.

Die Tabellen 5.5 und 5.6 enthalten die Ergebnisse für FInt mit YICES bzw. MATHSAT ohne Benutzung von Blackboxes.

		inkr., alt.			nicht inkr., alt.			inkr., nicht alt.			nicht inkr., nicht alt.		
n	p	Tiefe	Zeit	Speicher	Tiefe	Zeit	Speicher	Tiefe	Zeit	Speicher	Tiefe	Zeit	Speicher
2	2	12	0.02	10.66	12	0.08	10.66	8	0.04	10.87	8	0.10	10.86
3	2	12	0.01	10.79	12	0.12	10.80	8	0.03	11.01	8	0.12	11.01
	3	26	1.55	13.14	26	2.10	13.01	17	3.27	15.24	17	3.53	14.61
4	2	12	0.03	11.05	12	0.16	11.00	8	0.04	11.28	8	0.13	11.28
	3	26	1.63	13.61	26	2.38	13.56	17	2.97	15.44	17	3.91	15.18
	4	40	415.75	28.18	40	381.34	25.32	26	821.81	36.83	26	804.19	34.58
5	2	12	0.03	11.19	12	0.17	11.18	8	0.04	11.50	8	0.15	11.53
	3	26	1.63	14.14	26	2.34	13.76	17	3.27	16.17	17	4.16	16.27
	4	40	430.90	28.93	40	465.24	25.57	26	1246.12	41.03	26	1057.10	35.35
10	2	12	0.05	12.45	12	0.33	12.43	8	0.07	13.16	8	0.31	13.17
	3	26	2.03	18.19	26	4.16	18.05	17	3.87	21.03	17	5.41	20.75
	4	40	537.24	37.29	40	490.88	33.64	26	1118.13	49.51	26	1070.27	47.42
15	2	12	0.09	14.22	12	0.54	14.22	8	0.11	15.51	8	0.55	15.60
	3	26	2.65	23.82	26	5.58	22.84	17	3.78	27.90	17	7.66	28.12
	4	40	520.79	47.14	40	586.89	43.34	26	1176.19	66.08	26	1102.66	58.23
20	2	12	0.14	16.67	12	0.87	16.68	8	0.18	18.55	8	0.92	18.56
	3	26	2.72	30.11	26	8.17	29.78	17	4.90	37.68	17	10.04	37.07
	4	40	657.48	63.87	40	565.73	57.38	26	1109.55	83.06	26	1097.13	76.62

Tabelle 5.2: Vergleich von Kombination aus inkrementellem/nicht-inkrementellem und alternierendem/nicht-alternierendem Modus *mit Blackboxes* beim Fischer-Protokoll mit Synchronisation über Kanäle und YICES als Solver.

Tabelle 5.7 präsentiert die Ergebnisse von YICESFILE mit FInt und Blackboxes. Zusätzlich geben wir den Rückgabewert von YICESFILE an. Wir werden darauf im folgenden Abschnitt genauer eingehen. Wir haben auch gleiche Tests unter Verwendung LIRA als Solver durchgeführt, allerdings scheiterte LIRA schon bei kleinsten Instanzen (z. B. $n = 2,\ p = 2$) am zur Verfügung stehenden Speicher. Der Grund dafür ist, dass LIRA den kompletten Zustandsraum explizit aufbaut, was zu einem *enormen* Speicherverbrauch führt. Deswegen verzichten wir auf die Darstellung der Ergebnisse von LIRA.

5.3 Auswertung der Ergebnisse

Die wichtigste Aufgabenstellung bei den Experimenten war herauszufinden, ob das Einsetzen von Blackboxes bei allen getesteten Modi- und Solverkombinationen eine Verringerung der Laufzeit und des Speicherverbrauchs bringt.

Weiter analysieren wir die Ergebnisse der Experimente mit YICES (siehe Abbildungen 5.1, 5.2 und 5.5) und MATHSAT (siehe Abbildungen 5.3, 5.4 und 5.6):

Betrachten wir zunächst nur die Ergebnisse von YICES und MATHSAT. Für FChan (sowohl mit als auch ohne Blackboxes) und für FInt konnten folgende Beobachtungen gemacht werden:

Das inkrementelle Vorgehen verwendet Informationen, die der Solver in vorherigen Abwicklungstiefen gelernt hat, und somit erspart sich der Solver das nochmalige Explorieren von Teilen dieser Zustandsräume. Dies kommt sowohl beim nicht-alternierenden als auch beim alternierenden Vorgehen zum Tragen und führt im Allgemeinen zu geringerem Ressourcenbedarf.

Beim alternierenden Formelaufbau wird in jedem Schritt genau eine Aktion codiert,

		inkr., alt.			nicht inkr., alt.			inkr., nicht alt.			nicht inkr., nicht alt.		
n	p	Tiefe	Zeit	Speicher	Tiefe	Zeit	Speicher	Tiefe	Zeit	Speicher	Tiefe	Zeit	Speicher
2	2	12	0.04	11.25	12	0.19	11.70	8	0.05	11.54	8	0.21	11.83
3	2	12	0.07	12.01	12	0.31	12.54	8	0.10	12.54	8	0.32	13.11
	3	26	2.71	16.55	26	10.18	18.20	17	10.61	20.43	17	14.20	21.20
4	2	12	0.10	13.01	12	0.47	13.97	8	0.13	13.78	8	0.48	14.59
	3	26	3.79	18.83	26	15.89	22.72	17	16.54	24.17	17	30.24	30.65
	4	40	2511.47	213.52	40	1500.22	148.23	– TIMEOUT –			– TIMEOUT –		
5	2	12	0.12	14.12	12	0.70	15.46	8	0.20	15.45	8	0.74	16.67
	3	26	5.42	21.62	26	26.49	28.71	17	21.08	29.95	17	41.57	40.30
	4	40	3767.65	255.63	40	14707.90	971.33	– TIMEOUT –			– TIMEOUT –		
10	2	12	0.47	26.06	12	2.80	31.48	8	0.82	31.81	8	3.05	36.64
	3	26	12.12	49.85	26	77.83	80.34	17	38.25	67.23	17	125.77	91.61
	4	40	7648.51	459.49	40	21254.90	757.25	– TIMEOUT –			– TIMEOUT –		
15	2	12	1.40	54.00	12	8.78	67.90	8	2.20	66.48	8	8.29	79.05
	3	26	27.64	112.08	26	171.48	177.93	17	92.91	151.83	17	215.67	208.87
	4	40	19867.30	1057.59	– TIMEOUT –			– TIMEOUT –			– TIMEOUT –		
20	2	12	3.21	100.90	12	19.15	130.21	8	4.76	126.08	8	19.27	154.21
	3	26	63.92	215.24	26	303.08	362.00	17	151.60	285.26	17	390.93	417.91
	4	– TIMEOUT –			– TIMEOUT –			– TIMEOUT –			– TIMEOUT –		

Tabelle 5.3: Vergleich von Kombination aus inkrementellem/nicht-inkrementellem und alternierendem/nicht-alternierendem Modus *ohne Blackboxes* beim Fischer-Protokoll mit Synchronisation über Kanäle und MathSat als Solver.

d. h. abwechselnd entweder nur ein Sprung oder ein Verzögerungsschritt. Bei nicht-alternierendem Vorgehen können in jedem Schritt sowohl Sprünge als auch Verzögerungsschritte stattfinden, deswegen müssen in jedem Schritt beide Möglichkeiten codiert werden. Dies führt zu längeren Formel als bei der alternierenden Vorgehensweise. Allerdings kann bei Verwendung der nicht-alternierenden Formel oft in geringeren Abwicklungstiefen ein Gegenbeispiel gefunden werden, wenn zwei aufeinanderfolgende Sprünge nicht wie bei der alternierenden Variante durch einen Verzögerungsschritt der Dauer Null getrennt werden müssen. Bei den von uns untersuchten Modellen ist überwiegt der Effekt durch die Verkleinerung der Formel den durch die geringe Länge des Gegenbeispiels, so dass der alternierende Modus in der Regel effizienter ist als der nicht-alternierende. Diese Beobachtung ist unabhängig davon, ob der Solver inkrementell arbeitet oder nicht.

Die Erwartung, dass eine Kombination aus alternierenden und inkrementellen Vorgehensweisen im Allgemeinen den geringsten Speicherbedarf bzw. die geringste Laufzeit vorweist, hat sich durch die experimentellen Ergebnisse bestätigt.

Im Vergleich zwischen Yices und MathSat schneidet Yices unabhängig vom gewählten Modus deutlich besser ab. Nicht nur die Laufzeiten von Yices sind in der Regel kürzer, auch der Speicherplatzbedarf von Yices fällt geringer aus. Der höhere Ressourcenverbrauch von MathSat führte in vielen Fällen dazu, dass aufgrund der Zeitbeschränkung MathSat im Unterschied zu Yices kein Ergebnis liefern konnte.

Die Einführung von Blackboxes für diejenigen Teile des Modells, die keinen Einfluss auf die Realisierbarkeit der untersuchten Eigenschaft haben (sollten), hat in jedem Fall den Ressourcenverbrauch verringert, und zwar umso mehr, je größer der Teil des Modells war, der in Blackboxes gesetzt werden konnte. Bei n Prozessen und p Property-Automaten sind dies $n - p$ Automaten in Blackboxes und $p + 1$ Automaten, die im Modell sichtbar

		inkr., alt.			nicht inkr., alt.			inkr., nicht alt.			nicht inkr., nicht alt.		
n	p	Tiefe	Zeit	Speicher	Tiefe	Zeit	Speicher	Tiefe	Zeit	Speicher	Tiefe	Zeit	Speicher
2	2	12	0.03	11.21	12	0.19	11.56	8	0.05	11.50	8	0.18	11.79
3	2	12	0.04	11.41	12	0.22	11.84	8	0.06	11.73	8	0.23	12.25
	3	26	2.71	15.54	26	9.93	18.06	17	10.12	21.39	17	14.13	21.47
4	2	12	0.06	11.65	12	0.23	12.20	8	0.07	12.19	8	0.26	12.64
	3	26	2.43	16.24	26	9.94	19.15	17	9.06	21.07	17	13.16	22.30
	4	40	2405.11	198.02	40	1791.16	168.01	– TIMEOUT –			– TIMEOUT –		
5	2	12	0.06	11.97	12	0.26	12.51	8	0.08	12.54	8	0.31	13.09
	3	26	2.82	18.49	26	10.25	20.78	17	8.55	21.42	17	15.15	23.80
	4	40	1836.57	190.98	40	2718.72	248.37	– TIMEOUT –			– TIMEOUT –		
10	2	12	0.11	13.96	12	0.55	15.14	8	0.16	15.15	8	0.60	16.23
	3	26	3.35	23.20	26	16.88	31.12	17	10.77	29.72	17	18.92	34.90
	4	40	3585.75	290.11	40	3986.68	359.01	– TIMEOUT –			– TIMEOUT –		
15	2	12	0.19	16.94	12	1.01	19.06	8	0.25	18.73	8	1.04	21.00
	3	26	3.88	31.79	26	19.95	46.70	17	11.68	39.18	17	21.92	49.30
	4	40	3718.62	271.09	40	3197.48	329.14	– TIMEOUT –			– TIMEOUT –		
20	2	12	0.29	20.84	12	1.58	24.12	8	0.39	23.86	8	1.67	26.81
	3	26	5.07	41.67	26	29.62	65.58	17	13.76	54.54	17	31.13	73.57
	4	40	2637.11	237.85	40	3035.13	359.31	– TIMEOUT –			– TIMEOUT –		

Tabelle 5.4: Vergleich von Kombination aus inkrementellem/nicht-inkrementellem und alternierendem/nicht-alternierendem Modus *mit Blackboxes* beim Fischer-Protokoll mit Synchronisation über Kanäle und MathSat als Solver.

bleiben – abgesehen von den Property-Automaten wird auch der Synchronisationsautomat benötigt. Entsprechend sind die Zeiten mit und ohne Blackboxes vergleichbar, wenn $n = p$ ist.

In der Kombination von Blackboxes mit inkrementellem/nicht-inkrementellem Lösen und alternierender/nicht-alternierender BMC-Formel kann mit der alternierende Variante der Formel und inkrementellem Lösen am schnellsten ein Gegenbeispiel gefunden werden.

Betrachten wir abschließend die Ergebnisse der Experimente mit Yicesfile (siehe Tabelle 5.7).

Es ist zu beobachten, dass nur die Fälle „$n = p$“ gelöst werden konnten. Hier sind alle Automaten des Netzwerks Property-Automaten und folglich werden keine Blackboxes gesetzt. Deshalb enthalten die Formeln keine universell quantifizierten Variablen. In allen anderen Fällen, in denen Blackboxes gesetzt wurden, konnte Yicesfile-Solver ab einer bestimmten Tiefe nicht mehr entscheiden, ob die BMC-Formel erfüllbar ist („SAT“) oder nicht („UNSAT“), sondern Yicesfile terminierte in der angegebenen Tiefe mit dem Ergebnis „UNKNOWN“. Das liegt daran, dass das von Yicesfile eingesetzte Entscheidungsverfahren für quantifizierte SMT-Formeln unvollständig ist und nicht in jedem Fall ein Ergebnis liefern kann.

Daraus lässt sich die Schlussfolgerung ziehen, dass wir ein mächtigeres Verfahren benötigen als dasjenige, das z. B. in Yices verwendet wird. Gleichzeitig sollte es aber in der Lage sein, effizienter über beschränkte Integervariablen zu quantifizieren als z. B. Lira.

		inkr., alt.			nicht inkr., alt.			inkr., nicht alt.			nicht inkr., nicht alt.		
n	p	Tiefe	Zeit	Speicher	Tiefe	Zeit	Speicher	Tiefe	Zeit	Speicher	Tiefe	Zeit	Speicher
2	2	8	0.01	10.34	8	0.02	10.34	6	0.02	10.47	6	0.05	10.48
3	2	8	0.01	10.48	8	0.05	10.49	6	0.02	10.67	6	0.05	10.64
	3	16	0.15	11.01	16	0.30	10.88	12	0.60	12.08	12	0.83	11.94
4	2	8	0.02	10.60	8	0.07	10.62	6	0.02	10.81	6	0.07	10.84
	3	16	0.25	11.34	16	0.47	11.36	12	1.12	12.86	12	2.17	13.21
	4	24	12.98	13.73	24	17.38	13.67	18	157.97	21.45	18	95.91	19.15
5	2	8	0.02	10.75	8	0.08	10.74	6	0.04	11.02	6	0.09	11.07
	3	16	0.41	11.86	16	0.61	11.68	12	2.52	14.12	12	2.70	14.10
	4	24	36.98	15.92	24	42.17	15.35	18	471.62	27.49	18	367.32	24.86
10	2	8	0.05	11.43	8	0.16	11.49	6	0.07	11.93	6	0.19	12.02
	3	16	0.87	13.55	16	1.76	14.08	12	13.05	18.98	12	13.74	18.99
	4	24	211.43	23.68	24	226.55	22.09	18	1962.45	42.33	18	1427.78	38.81
15	2	8	0.09	12.16	8	0.24	12.14	6	0.12	12.95	6	0.28	12.94
	3	16	1.96	15.55	16	3.01	16.13	12	32.96	24.27	12	21.86	21.48
	4	24	268.07	26.32	24	310.22	25.13	18	3035.82	58.24	18	3464.78	54.59
20	2	8	0.13	12.91	8	0.37	12.92	6	0.16	14.07	6	0.38	14.11
	3	16	2.94	17.44	16	4.25	17.83	12	60.34	30.05	12	64.78	31.38
	4	24	542.90	33.46	24	808.85	32.42	18	6814.26	82.33	18	5605.58	72.14

Tabelle 5.5: Vergleich von Kombination aus inkrementellem/nicht-inkrementellem und alternierendem/nicht-alternierendem Modus *ohne Blackboxes* beim Fischer-Protokoll mit Kommunikation über gemeinsame Integervariablen mit YICES als Solver.

		inkr., alt.			nicht inkr., alt.			inkr., nicht alt.			nicht inkr., nicht alt.		
n	p	Tiefe	Zeit	Speicher	Tiefe	Zeit	Speicher	Tiefe	Zeit	Speicher	Tiefe	Zeit	Speicher
2	2	8	0.03	10.80	8	0.08	10.90	6	0.05	11.01	6	0.08	11.09
3	2	8	0.03	11.09	8	0.13	11.22	6	0.07	11.38	6	0.16	11.48
	3	16	0.49	12.35	16	1.86	12.93	12	1.06	13.51	12	2.92	14.16
4	2	8	0.06	11.33	8	0.15	11.57	6	0.08	11.74	6	0.20	11.91
	3	16	0.56	13.16	16	2.29	13.89	12	2.02	14.62	12	3.80	15.20
	4	24	19.17	17.63	24	66.93	18.58	18	78.65	36.47	18	78.91	30.13
5	2	8	0.10	11.63	8	0.21	11.83	6	0.13	12.15	6	0.25	12.42
	3	16	0.98	13.83	16	4.62	15.23	12	2.10	15.42	12	5.00	16.49
	4	24	30.18	20.77	24	119.66	24.10	18	454.44	98.05	18	468.96	88.87
10	2	8	0.21	13.23	8	0.52	13.80	6	0.39	14.36	6	0.55	14.64
	3	16	2.58	17.32	16	7.90	19.49	12	6.39	20.75	12	12.11	22.94
	4	24	94.93	30.28	24	294.73	42.08	18	1066.16	166.24	18	1194.89	152.79
15	2	8	0.43	14.85	8	0.74	15.46	6	0.99	16.22	6	0.88	16.82
	3	16	3.88	20.90	16	12.79	23.77	12	11.33	26.27	12	18.63	28.22
	4	24	180.16	43.97	24	571.14	63.06	18	1561.80	214.46	18	2544.90	256.40
20	2	8	1.11	16.44	8	1.98	17.47	6	1.38	18.47	6	1.61	19.19
	3	16	8.61	25.69	16	21.50	29.10	12	20.72	32.59	12	29.61	35.33
	4	24	443.28	72.86	24	1316.43	73.76	18	1880.37	271.17	18	7291.71	500.60

Tabelle 5.6: Vergleich von Kombination aus inkrementellem/nicht-inkrementellem und alternierendem/nicht-alternierendem Modus *ohne Blackboxes* beim Fischer-Protokoll mit Kommunikation über gemeinsame Integervariablen mit MATHSAT als Solver.

		alternierend				nicht alternierend			
n	p	Ergebnis	Tiefe	Zeit	Speicher	Ergebnis	Tiefe	Zeit	Speicher
2	2	SAT	8	1.04	9.78	SAT	6	0.64	9.87
3	2	UNKNOWN	16	166.26	9.94	UNKNOWN	9	461.91	9.97
	3	SAT	16	11.97	10.06	SAT	12	8.00	10.34
4	2	UNKNOWN	14	113.44	9.92	UNKNOWN	8	378.61	9.95
	3	UNKNOWN	14	183.71	10.02	UNKNOWN	8	490.03	10.11
	4	SAT	24	75.27	10.50	SAT	18	184.62	11.01
5	2	UNKNOWN	12	73.60	9.90	UNKNOWN	7	121.89	9.93
	3	UNKNOWN	12	122.34	9.98	UNKNOWN	7	314.25	10.06
	4	UNKNOWN	12	164.68	10.11	UNKNOWN	7	250.12	10.19
10	2	UNKNOWN	12	73.60	9.96	UNKNOWN	7	123.63	9.99
	3	UNKNOWN	12	131.01	10.04	UNKNOWN	7	230.61	10.12
	4	UNKNOWN	10	108.00	10.09	UNKNOWN	7	378.45	10.25
15	2	UNKNOWN	12	73.20	10.01	UNKNOWN	7	122.42	10.04
	3	UNKNOWN	12	131.98	10.09	UNKNOWN	7	232.65	10.18
	4	UNKNOWN	10	107.83	10.14	UNKNOWN	7	377.29	10.31
20	2	UNKNOWN	12	73.39	10.08	UNKNOWN	7	126.63	10.11
	3	UNKNOWN	12	131.32	10.16	UNKNOWN	7	232.48	10.24
	4	UNKNOWN	10	108.89	10.21	UNKNOWN	7	385.49	10.37
25	2	UNKNOWN	12	73.36	10.13	UNKNOWN	7	125.53	10.16
	3	UNKNOWN	12	130.48	10.21	UNKNOWN	7	228.53	10.30
	4	UNKNOWN	10	107.71	10.27	UNKNOWN	7	379.23	10.43

Tabelle 5.7: Vergleich von alternierendem und nicht-alternierendem Modus *mit Black-boxes* beim Fischer-Protokoll mit Kommunikation über gemeinsame Integervariablen mit YICESFILE als Solver.

6 Zusammenfassung und Ausblick

In dieser Arbeit haben wir gezeigt, wie mit Hilfe von Bounded Model Checking Fehler in vollständigen und unvollständigen Netzwerken von Zeitautomaten gefunden werden können.

Im ersten Teil der Arbeit haben wir zunächst für vollständige Netzwerke von Zeitautomaten Techniken entwickelt, um nachzuweisen, ob ein System fehlerhaft ist. Dabei haben wir drei verschiedene Kommunikationsmechanismen berücksichtigt: *gemeinsame Uhrenvariablen*, *Synchronisation über Kanäle* und *gemeinsame beschränkte Integervariablen*. Die entwickelten Methoden können die Gültigkeit von Invarianteneigenschaften, d. h. Eigenschaften, die jeder erreichbare Zustand erfüllen muss, damit das System korrekt ist, widerlegen. Dafür haben wir Pfade einer festen Länge, die zu einem Zustand führen, der die Invarianteneigenschaft verletzt, als SMT-Formeln codiert. Deren Erfüllbarkeit kann von entsprechenden Solvern wie z. B. Yices oder MathSat überprüft werden. Aus einer erfüllenden Belegung kann der entsprechende Pfad, der zu dem Fehler führt, extrahiert und zur Reproduktion, Lokalisierung und Behebung des Fehlers verwendet werden.

Der zweite Teil der Arbeit beschäftigt sich mit unvollständigen Netzwerken von Zeitautomaten. Unvollständig heißt, dass nur von einem Teil der Zeitautomaten eines Netzwerks die Funktionsweise bekannt ist; von den fehlenden Automaten, den *Blackboxes*, kennen wir nur die Schnittstelle, über die sie mit den anderen Automaten kommunizieren, also die Kanäle zur Synchronisation mit den bekannten Automaten und die Integervariablen, die von den Blackboxes geschrieben und den anderen Automaten gelesen werden. Ansonsten ist das mögliche Verhalten der Blackboxes nicht eingeschränkt.

Unser Ziel war, auch für solche unvollständigen Netzwerke Invarianteneigenschaften zu widerlegen. Das heißt, wir wollten nachweisen, dass es für alle möglichen Implementierungen der Blackboxes Pfade gibt, die zu einem Zustand führen, der die Invariante verletzt. Dann kann die Eigenschaft des Systems nicht mehr durch geeignete Implementierung der unbekannten Teile realisiert werden; die vorliegenden Teile sind fehlerhaft.

Dafür haben wir zwei verschiedene Methoden entwickelt. Die erste beruht auf der Formulierung als SMT-Problem. Im Vergleich zu BMC für vollständige Netzwerke können bestimmte Kanten nicht unabhängig von der Implementierung der Blackboxes ausgeführt werden. Bei der SMT-Formulierung müssen wir versuchen, einen Zustand zu erreichen, in dem die Invariante verletzt ist, ohne solche unsicheren Kanten zu verwenden.

Wenn die Zeitautomaten mit Hilfe von Integervariablen kommunizieren, kann es allerdings vorkommen, dass diese Methode keinen Fehler findet, obwohl unabhängig von den Blackboxes einer vorliegt. Das kann passieren, wenn für alle möglichen Implementierungen jeweils ein Pfad existiert, jedoch nicht immer derselbe.

Aus diesem Grund entwickelten wir die zweite Methode, die auf quantifizierten SMT-Formeln basiert. Damit erreicht man für Netzwerke mit Integervariablen eine höhere

Genauigkeit bei der Fehlersuche.

Um die Methoden experimentell evaluieren zu können, haben wir sie in einem Werkzeug namens `t-bounce` implementiert. Als Fallstudie haben wir zwei verschiedene Varianten des Fischer-Protokolls verwendet. Dabei wollen n Prozesse auf eine gemeinsame Ressource zugreifen, die als ein kritischer Bereich anzusehen ist. In das Protokoll haben wir Blackboxes und einen Fehler eingebaut, indem wir in allen Automaten (außer dem für die Synchronisation bei der Version mit Kommunikation über Kanäle) eine strikte Ungleichung durch eine nicht-strikte ersetzt haben.

Für die Variante mit ausschließlicher Kommunikation über Kanäle konnten wir mit dem SMT-Ansatz zeigen, dass der Fehler unabhängig von den Blackboxes auftritt. Der Vergleich verschiedener Varianten und Solvereinstellungen hat gezeigt, dass die alternierende Formel, bei der sich Verzögerungsschritte und Sprünge abwechseln, zusammen mit dem inkrementellen Lösen der Formel am effizientesten ist.

Außerdem haben unsere Experimente gezeigt, dass geeignet gewählte Blackboxes die Laufzeit für die Fehlersuche deutlich verringern können. Mit Blackboxes waren wir in der Lage, in wesentlich größeren Instanzen des Fischer-Protokolls Fehler zu finden als ohne Blackboxes. Im letzteren Fall scheiterte BMC an der zu großen Laufzeit.

Die Experimente mit dem Ansatz mit quantifizierten SMT-Formeln waren wegen der Beschränkungen der verfügbaren Solver weniger erfolgreich. Yices' Verfahren zur Lösung solcher Formeln sind unvollständig. In allen nicht-trivialen Fällen bekamen wir nur „UNKNOWN“ als Ergebnis. Lira ist zwar vollständig, allerdings ist sein Speicherverbrauch so groß, dass der zur Verfügung stehende Speicher bereits bei den kleinsten Instanzen verbraucht war, bevor ein Fehler gefunden werden konnte.

Diese Schwäche der Solver zeigt, dass an dieser Stelle großer Forschungsbedarf steht. Nachdem mit booleschen SAT-Solvern bereits sehr große Probleme effizient gelöst werden konnten, wurden die Solver für quantifizierte boolesche Formeln (QBF) [8] verbessert, so dass auch damit heutzutage reale Probleme gelöst werden können. Für quantorenfreie SMT-Formeln stehen zwar effiziente Solver wie Yices oder MathSat zur Verfügung; für quantifizierte SMT-Formeln jedoch nichts Vergleichbares. Diese wären aber notwendig, um unseren zweiten Ansatz praktisch anzuwenden. Deshalb ist eine wichtige Aufgabe für die künftige Forschung, effiziente Solver für quantifizierte SMT-Formeln zu entwickeln.

Anhang

Aufruf von `t-bounce`

Der Aufruf des Werkzeugs erfolgt mit folgender Syntax:

`t-bounce` ⟨Optionen⟩

Als Optionen stehen dabei zur Auswahl:

`-help` Zeigt Informationen über die Aufrufsyntax und die verfügbaren Optionen von `t-bounce` an. Danach terminiert `t-bounce`.

`-ta` ⟨`datei1`⟩ ⟨`datei2`⟩ Spezifiziert die Dateien mit der Beschreibung des Automatennetzwerks (1. Parameter) und den zu prüfenden Eigenschaften (2. Parameter).

`-prop all` | ⟨`zahl`⟩ Gibt an, welche Eigenschaft aus der Eigenschaftsdatei überprüft werden soll. Die Eigenschaften sind durchnummeriert (die erste Eigenschaft hat die Nummer 0). Wird anstatt einer Zahl „all“ angegeben, so werden nacheinander alle Eigenschaften überprüft.

`-l` ⟨`zahl`⟩ Spezifiziert die minimale Abwicklungstiefe, die beim Bounded Model Checking betrachtet werden soll.

`-u` ⟨`zahl`⟩ Spezifiziert die maximale Abwicklungstiefe, die untersucht werden soll.

`-bb` ⟨`name`⟩ Behandelt den Automaten mit dem angegebenen Namen als Blackbox. Sollen mehrere Automaten als Blackboxes behandelt werden, muss diese Option für jeden dieser Automaten angegeben werden.

`-autobb` Setzt alle Automaten in Blackboxes, von denen die zu prüfende Eigenschaft nicht abhängt und die nicht mit Hilfe des Parameters `-wb` davon ausgeschlossen werden.

`-wb` ⟨`name`⟩ Sorgt dafür, dass der Automat mit dem angegebenen Namen nicht in eine Blackbox gesetzt wird, wenn mit Hilfe des Parameters `-autobb` automatisch Blackboxes gesetzt werden. Falls mehrere Automaten nicht in Blackboxes gesetzt werden sollen, kann dies geschehen, indem der Parameter mehrfach angegeben wird.

`-solver yices` | `yicesfile` | `mathsat` | `lira` Wählt den zu verwendenden Solver aus. Der Wert „`yices`“ steht dabei für den Solver Yices, der mittels der Bibliotheksfunktionen aufgerufen wird; bei „`yicesfile`“ steht für denselben Solver, allerdings wird in diesem Fall die Formel per Datei übergeben. Der Grund für diese Unterscheidung ist, dass die Bibliotheksschnittstelle keine Quantoren unterstützt. Standardmäßig wird „`yices`“ verwendet.

`-qbf` Verwendet die SMT-Codierung mit Quantoren anstelle der normalen SMT-Codierung ohne Quantoren für das Bounded Model Checking, sofern der verwendete Solver Quantoren unterstützt. Andernfalls terminiert `t-bounce` mit einer Fehlermeldung.

`-inc` Aktiviert das inkrementelle Lösen der BMC-Formeln, sofern der verwendete Solver dies unterstützt. Standardmäßig ist diese Option aktiviert.

`-noinc` Deaktiviert das inkrementelle Lösen.

`-alt` Verwendet für BMC die Codierung, bei der sich Verzögerungsschritte und Sprünge abwechseln. Diese Option ist standardmäßig aktiviert.

`-noalt` Deaktiviert die alternierende Codierung, d. h. in jedem Zeitschritt ist dann sowohl ein Verzögerungsschritt als auch ein Sprung möglich.

Abgesehen von `-bb` und `-wb` darf jede Option höchstens einmal angegeben werden. Der Schalter `-autobb` darf nicht zusammen mit `-bb` verwendet werden. Wird `-wb` verwendet, muss auch `-autobb` gesetzt sein. Da nur die Bibliotheksschnittstelle von Yices und MathSat inkrementelles Lösen unterstützen, nicht aber Lira und Yices per Dateiaustausch, ist der Parameter `-inc` nur bei ersteren sinnvoll. Im SMT-Modus mit Quantoren unterstützt bisher keiner der Solver den inkrementellen Modus. Dann wird eine Warnung ausgegeben und im nicht-inkrementellen Modus fortgefahren. Das Lösen von SMT-Formeln mit Quantoren wird nur von Lira und Yices per Dateiaustausch unterstützt, jedoch nicht von MathSat und der Bibliotheksschnittstelle von Yices. Somit führt die Verwendung des Schalters `-qbf` zusammen mit Yices und MathSat zu einer Fehlermeldung. Da die Parameter `-inc` und `-noinc` bzw. `-alt` und `-noalt` gegensätzliche Wirkung haben, dürfen sie nicht gleichzeitig angegeben werden.

Literaturverzeichnis

[1] Ábrahám, Erika, Bernd Becker, Felix Klaedtke und Martin Steffen: *Optimizing Bounded Model Checking for Linear Hybrid Systems*. In: Cousot, Radhia (Herausgeber): *6th Int'l Conf. on Verification, Model Checking, and Abstract Interpretation (VMCAI)*, Band 3385 der Reihe *Lecture Notes in Computer Science*, Seiten 396–412, Paris, France, Januar 2005. Springer-Verlag.

[2] Alur, Rajeev, Costas Courcoubetis und David L. Dill: *Model-Checking in Dense Real-time*. Information and Computation, 104(1):2–34, 1993.

[3] Alur, Rajeev und David L. Dill: *A Theory of Timed Automata*. Theoretical Computer Science, 126:183–235, 1994.

[4] Alur, Rajeev und Thomas A. Henzinger: *Real-Time Logics: Complexity and Expressiveness*. Information and Computation, 104(1):35–77, 1993.

[5] Becker, Bernd, Christian Dax, Jochen Eisinger und Felix Klaedtke: *LIRA: Handling Constraints of Linear Arithmetics over the Integers and the Reals*. In: Damm, Werner und Holger Hermanns (Herausgeber): *19th Int'l Conf. on Computer-Aided Verification (CAV)*, Band 4590 der Reihe *Lecture Notes in Computer Science*, Seiten 307–310, Berlin, Germany, 2007. Springer-Verlag.

[6] Biere, Armin, Alessandro Cimatti, Edmund M. Clarke, Masahiro Fujita und Yunshan Zhu: *Symbolic Model Checking Using SAT Procedures instead of BDDs*. In: *Design Automation Conference (DAC)*, Seiten 317–320. ACM Press, 1999.

[7] Biere, Armin, Alessandro Cimatti, Edmund M. Clarke und Yunshan Zhu: *Symbolic Model Checking without BDDs*. In: Cleaveland, Rance (Herausgeber): *5th Int'l Conf. on Tools and Algorithms for Construction and Analysis of Systems (TACAS)*, Band 1579 der Reihe *Lecture Notes in Computer Science*, Seiten 193–207, Amsterdam, The Netherlands, März 1999. Springer-Verlag.

[8] Biere, Armin, Marijn J. H. Heule, Hans van Maaren und Toby Walsh (Herausgeber): *Handbook of Satisfiability*, Band 185 der Reihe *Frontiers in Artificial Intelligence and Applications*. IOS Press, Februar 2009.

[9] Bruttomesso, Roberto, Alessandro Cimatti, Anders Franzén, Alberto Griggio und Roberto Sebastiani: *The MathSAT 4 SMT Solver*. In: Gupta, Aarti und Sharad Malik (Herausgeber): *20th Int'l Conf. on Computer-Aided Verification (CAV)*, Band 5123 der Reihe *Lecture Notes in Computer Science*, Seiten 299–303, Princeton, NJ, USA, Juli 2008. Springer-Verlag.

[10] Dutertre, Bruno und Leonardo Mendonça de Moura: *A Fast Linear-Arithmetic Solver for DPLL(T).* In: Ball, Thomas und Robert B. Jones (Herausgeber): *16th Int'l Conf. on Computer-Aided Verification (CAV)*, Band 4144 der Reihe *Lecture Notes in Computer Science*, Seiten 81–94, Seattle, WA, USA, August 2006. Springer-Verlag.

[11] Herbstritt, Marc, Bernd Becker und Christoph Scholl: *Advanced SAT-Techniques for Bounded Model Checking of Blackbox Designs.* In: Abadir, Magdy S., Li-C. Wang und Jayanta Bhadra (Herausgeber): *7th Int'l Workshop on Microprocessor Test and Verification (MTV)*, Seiten 37–44, Austin, Texas, USA, Dezember 2006. IEEE Computer Society.

[12] Lamport, Leslie: *A Fast Mutual Exclusion Algorithm.* ACM Trans. Comput. Syst., 5(1):1–11, 1987.

[13] Larsen, Kim Guldstrand, Paul Pettersson und Wang Yi: *UPPAAL in a Nutshell.* Int'l Journal on Software Tools for Technology Transfer, 1(1–2):134–152, 1997.

[14] Miller, Christian, Stefan Kupferschmid, Matthew D. T. Lewis und Bernd Becker: *Encoding Techniques, Craig Interpolants and Bounded Model Checking for Incomplete Designs.* In: Strichman, Ofer und Stefan Szeider (Herausgeber): *13th Int'l Conf. Theory and Applications of Satisfiability Testing (SAT)*, Band 6175 der Reihe *Lecture Notes in Computer Science*, Seiten 194–208, Edinburgh, United Kingdom, Juli 2010. Springer-Verlag.

[15] Niebert, Peter, Moez Mahfoudh, Eugene Asarin, Marius Bozga, Oded Maler und Navendu Jain: *Verification of Timed Automata via Satisfiability Checking.* In: Damm, Werner und Ernst-Rüdiger Olderog (Herausgeber): *7th Int'l Symp. on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT)*, Band 2469 der Reihe *Lecture Notes in Computer Science*, Seiten 225–244, Oldenburg, Germany, September 2002. Springer-Verlag.

[16] Nieuwenhuis, Robert und Albert Oliveras: *DPLL(T) with Exhaustive Theory Propagation and Its Application to Difference Logic.* In: Etessami, Kousha und Sriram K. Rajamani (Herausgeber): *17th Int'l Conf. on Computer Aided Verification (CAV)*, Band 3576 der Reihe *Lecture Notes in Computer Science*, Seiten 321–334, Edinburgh, Scotland, UK, Juli 2005. Springer.

[17] Nopper, Tobias und Christoph Scholl: *Approximate Symbolic Model Checking for Incomplete Designs.* In: Hu, Alan J. und Andrew K. Martin (Herausgeber): *5th Int'l Conf. on Formal Methods in Computer-Aided Design (FMCAD)*, Band 3312 der Reihe *Lecture Notes in Computer Science*, Seiten 290–305, Austin, Texas, USA, September 2004. Springer-Verlag.

[18] Sorea, Maria: *Bounded Model Checking for Timed Automata.* In: Vogler, Walter und Kim Larsen (Herausgeber): *Workshop on Models for Time-Critical Systems (MTCS)*, Band 68(5) der Reihe *Electronic Notes in Theoretical Computer Science*, Seiten 116–134, Brno, Czech Republic, August 2002. Elsevier.