In *Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen.* by Christoph Scholl, Sefan Disch (Eds.). GI/ITG/GMM-Workshop, Shaker Verlag, Aachen, March 3-5, 2008

Symbolische Modellprüfung Aktor-orientierter High-level SystemC-Modelle mit Intervalldiagrammen

Jens Gladigau, Frank Blendinger, Christian Haubelt und Jürgen Teich
Lehrstuhl für Informatik 12 (Hardware-Software-Co-Design)
Universität Erlangen-Nürnberg
{gladigau, blendinger, haubelt, teich}@codesign.informatik.uni-erlangen.de

Diese Arbeit stellt einen Ansatz vor, mit dem Aktor-orientierte SystemC-Modelle eingebetteter Systeme schon zu Beginn des Entwurfsflusses, auf der Systemebene, formal verifiziert werden können. Verifikation der Modelle zu diesem frühen Entwicklungszeitpunkt ermöglicht es, Fehler im Design zu erkennen und zu beseitigen, die später hohe zusätzliche Kosten verursachen würden. Für die abstrakte Modellierung auf Systemebene ist die Klasse der Aktor-orientierten Modelle mit asynchroner Kommunikation über FIFO-Kanäle besonders gut geeignet und weit verbreitet. Eine sehr effiziente symbolische Darstellung solcher Modelle wird durch Intervalldiagramme erreicht, die binären Entscheidungsdiagrammen (BDDs) ähneln. Basierend auf diesen Intervalldiagrammen haben wir einen CTL-Modellprüfer entwickelt und eine Möglichkeit zur automatischen formalen Verifikation von Aktor-orientierten SystemC-Modellen geschaffen. Wir zeigen das Potential der Methodik im Vergleich zu BDD-basierten Ansätzen, indem wir den entwickelten Modellprüfer dem bekannten Werkzeug NuSMV gegenüberstellen.

1 Einleitung und Abgrenzung

Die ständig steigende Komplexität zu entwickelnder eingebetteter Systeme und immer kürzer werdende Produktzyklen erfordern eine Entwicklung auf immer höherer Abstraktionsebene. Die abstrakte Ebene, auf der Entwickler eingebettete Systeme meist modellieren, ist die Systemebene (*electronic system level*). Dort ist für einzelne Modellteile eines Systems noch nicht entschieden, ob sie im Endprodukt als Hardware- oder als Software-Komponente implementiert werden. Sehr vorteilhaft ist dabei eine Aktor-orientierte Modellierung auf Systemebene [LNW03]. Bei diesem Ansatz stehen Nebenläufigkeit von Komponenten (den Aktoren) und deren Kommunikation untereinander über Kanäle im Vordergrund. Der modulare Charakter unterstützt Wiederverwendbarkeit und ermöglicht eine schnelle Entwicklung. SystemC [GLMS02], eine C++-Bibliothek, ist als Modellierungssprache unter Systementwicklern etabliert, und findet sowohl in der Industrie als auch im akademischen Umfeld eine große Akzeptanz. Mit ihrem Konzept der Module und Kanäle unterstützt SystemC sehr gut die Aktor-orientierte Modellierung. Ein weiterer Vorteil von

SystemC ist das große Spektrum seiner Einsatzfähigkeit: Es reicht von der Modellierung auf Systemebene bis hinunter zu Registertransfer- oder Blockebene. Dadurch sind Teile eines SystemC-Modells auf Systemebene einfach sowohl nach Hardware als auch nach Software verfeinerbar, womit auch eine frühe Exploration des Entwurfsraums unterstützt wird.

Mit SystemC entwickelte Modelle ermöglichen durch die direkte Ausführbarkeit auf einfache Weise ihre Simulation. Damit ist eine Möglichkeit gegeben, das korrekte Verhalten eines entwickelten Systems unter bestimmten Bedingungen zu überprüfen, etwa durch Einfügen von (automatisch generierten) *Assertions* [LBW+07]. Während die Simulation allerdings nur einen Teil aller möglichen Zustände des Zustandsraumes eines Modells abdeckt, wird bei formaler Verifikation der gesamte Zustandsraum betrachtet. Ein formal verifiziertes Modell ist fehlerfrei im Sinne der Spezifikation – jedes mögliche Verhalten und alle Zustände wurden berücksichtigt. Eine fehlerfreie Simulation dagegen lässt solche Rückschlüsse nicht zu. Deshalb ist die Möglichkeit der formalen Verifikation von SystemC-Modellen wünschenswert, wofür wir in dieser Arbeit einen Ansatz vorstellen.

Eine der bekanntesten Formen der automatischen formalen Verifikation ist die symbolische Modellprüfung [McM93]. Dabei wird das zu prüfende Modell symbolisch beschrieben, was klassisch durch binäre Entscheidungsdiagramme (BDDs) [Bry86] geschieht. Als Eingabe für den Modellprüfer dient diese Beschreibung zusammen mit einer Spezifikation, gegeben etwa durch in einer Temporallogik formulierte Eigenschaften, die dann vollautomatisch geprüft werden. Sind alle Eigenschaften und damit die Spezifikation erfüllt, ist das Modell formal verifiziert. BDD-basierte Modellprüfung wie auch SAT-Solver basierte Modellprüfung zeigen ihre Stärke bei der Modulverifikation auf Registertransferebene. Allerdings ist nicht geklärt, ob sich die gleichen Verfahren auch für die symbolische Modellprüfung auf Systemebene eignen. Durch formale Verifikation auf Systemebene wird es möglich, schon am Anfang des Entwurfsflusses eines eingebetteten Systems, Fehler oder Schwächen im Design des Gesamtsystems zu erkennen. Deren spätere Entdeckung kann erheblichen Mehraufwand oder gar eine Neuentwicklung erfordern, was meist hohe Kosten verursacht – dies kann durch die Möglichkeit einer Verifikation schon zu Beginn des Systemdesigns vermieden werden.

Die Entwicklung formaler Verifikationstechniken für SystemC-Modelle befindet sich noch an ihrem Anfang [Var07]. Ein Ansatz zur Verifikation von SystemC-Modellen stammt von Große, Kühne und Drechsler [GKD06]. Dort wird ein SystemC-Modell zunächst in einen endlichen Automaten überführt. Dieser wird zusammen mit der zu verifizierenden Eigenschaft in ein Erfüllbarkeitsproblem umgewandelt, welches dann ein SAT-Solver löst. Der Ansatz in der vorliegenden Arbeit unterscheidet sich von [GKD06] in zwei zentralen Punkten. In unserer Arbeit beschränkt sich das SystemC-Modell nicht auf zyklenakkurate Modelle der Registertransferebene, die eine Antwort nach einer definierten Anzahl an Taktzyklen liefern müssen – eine weit höhere Abstraktionsebene ist das Ziel. Des Weiteren wird kein *bounded model checking* [BCCZ99] verwendet. Ein anderer Verifikationsansatz, der SystemC-Modelle als Eingabe verwendet, stammt von Karlsson, Eles und Peng [KEP06]. Dort wird ein SystemC-Modell zuerst in eine Darstellung basierend auf 1-beschränkten Petri-Netzen umgewandelt, und dann mit einem existierenden Modellprüfer geprüft. Diese Darstellung erreicht dabei eine hohe Komplexität. Um Modelle noch größerer Komplexität behandeln zu können, und das bekannte Problem der Zustandsexplosion zu mildern, wird bei unserem Ansatz (im Gegensatz zu [KEP06]) eine abstrakte Darstellung des SystemC-Modells verwendet. Kroening und Sharygina abstrahieren ebenfalls von einem gegebenen SystemC-Modell [KS05]. Sie transformieren das Modell in *labeled Kripke*

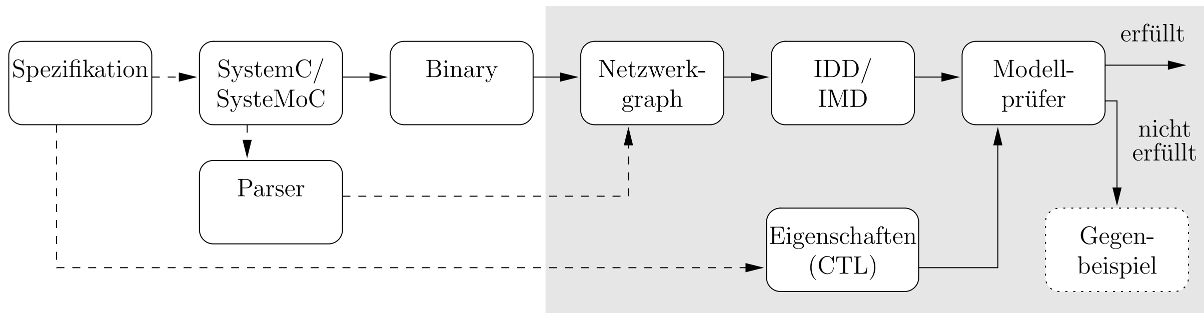

Abbildung 1: Das Vorgehen bei der IDD-basierten Modellprüfung im Überblick. Gestrichelte Übergänge sind optional, der Fokus dieser Arbeit liegt auf der schattierten Fläche. Das übersetzte SystemC-Programm (Binary) erzeugt eine Ausgabe, aus der Intervalldiagramme generiert werden.

structures (LKS), das heißt, in gerichtete Graphen, bei denen sowohl die Knoten (mit atomaren Präpositionen) als auch die Kanten (mit Ereignissen für Synchronisation) markiert sind. Mit einem SAT-basierten Ansatz werden aus den LKS einzeln abstrakte Repräsentationen generiert und deren parallele Komposition anschließend verifiziert. Dies erlaubt, zusammen mit einer vorher erfolgten Klassifizierung in Hardware- und Software-Threads, die formale Verifikation komplexer Systeme. Während dieser Ansatz für bereits weit entwickelte SystemC-Programme mit vorliegender Aufteilung in entsprechende Hardware- und Software-Threads gut geeignet ist, greift unser Ansatz früher im Entwurfsfluss – noch vor der eigentlichen HW/SW-Partitionierung und entsprechender Implementierung.

In dieser Arbeit stellen wir die Entwicklung eines symbolischen Modellprüfers auf Basis von Intervalldiagrammen für Aktor-orientierte SystemC-Modelle vor. Der Vorteil der Intervalldiagramme liegt dabei in der kompakten Repräsentation von Modellzuständen, insbesondere der Füllstände von Kommunikationskanälen. Sie sind damit sehr gut für die von uns betrachtete Klasse von Aktor-orientierten Modellen mit asynchroner Kommunikation über FIFO-Kanäle geeignet, wie sie zur Modellierung auf Systemebene verwendet werden.

Abbildung 1 zeigt den grafischen Überblick über den hier vorgestellten automatisierten Verifikationsansatz. Zunächst wird aus der Spezifikation eines eingebetteten Systems unter Verwendung des SysteMoC-Frameworks [FHT06] ein ausführbares SystemC-Modell auf der Systemebene erstellt. Ein Merkmal des mit dem Framework übersetzten ausführbaren Programms ist es, eine abstrakte Repräsentation des Modells auszugeben, unter anderem den so genannten Netzwerkgraphen. Dieser beinhaltet insbesondere alle Module und deren Vernetzung untereinander. In einem optionalen Schritt kann der Netzwerkgraph mit zusätzlichen Informationen angereichert werden, die durch den Übersetzungsschritt verloren gehen – etwa Variablennamen. Diese Informationen können in zukünftigen Arbeiten zur Verfeinerung unserer Methode dienen. Aus der abstrakten Repräsentation als Netzwerkgraph wird die symbolische Darstellung des Systems durch Intervalldiagramme erzeugt. Diese symbolische Repräsentation und die Spezifikation, als in *computational tree logic* (CTL) [CES86] formulierte Eigenschaften, dienen dem von uns entwickelten Modellprüfer als Eingabe. Dieser bestätigt entweder die Einhaltung der gegebenen Formeln oder liefert ein Gegenbeispiel.

Der vorliegende Beitrag ist wie folgt aufgebaut: Den verwendeten Modellierungsansatz eingebetteter Systeme stellen wir in Abschnitt 2 vor. Die abstrakte symbolische Darstel-

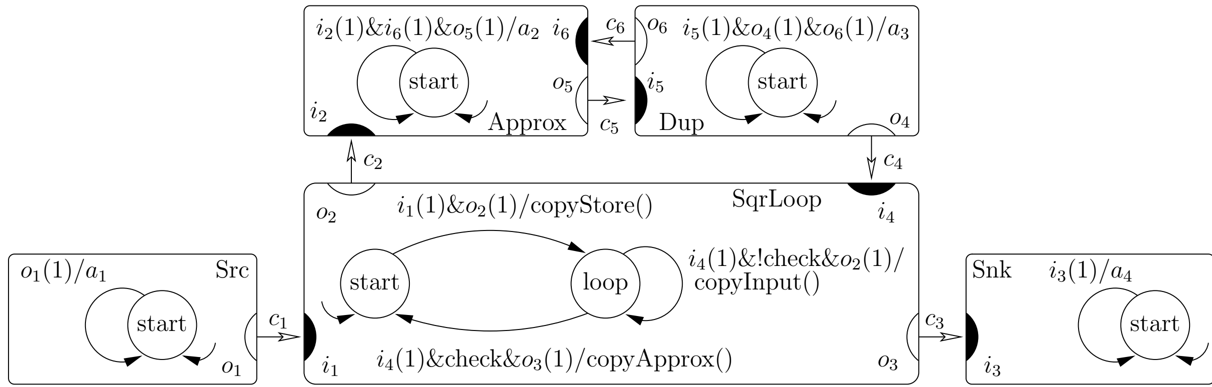

Abbildung 2: Newtons approximative Wurzelberechnung Aktor-orientiert modelliert. Zu sehen sind fünf Aktoren, die über sechs FIFO-Kanäle miteinander verbunden sind. In jedem Aktor ist die Zustandsmaschine gezeigt, die das Kommunikationsverhalten des Aktors darstellt.

lung dieser Systeme für die Modellprüfung wird in Abschnitt 3 erläutert. Abschnitt 4 beschreibt die symbolische CTL-Modellprüfung. Dieser Beitrag schließt mit einigen Experimenten und Vergleichen mit dem Modellprüfer NuSMV in Abschnitt 5.

2 Modellierung in SystemC

Für die Modellierung nutzen wir das SysteMoC-Framework [FHT06]. Es basiert auf dem Aktor-orientieren Ansatz [LNW03] und kann als eine Umsetzung von *FunState* [TSZ+99]-Konzepten in SystemC interpretiert werden. Für die Modellierung von eingebetteten Systemen ist ein Aktor-orientierter Ansatz durch den modularen Charakter vorteilhaft, der schnelle Entwicklung, Wiederverwendung und aktorweise Verfeinerung unterstützt. Eine strikte Trennung von Kommunikation und Verhalten, wie sie von dem (hierarchischen) FunState-Modell für interne Entwurfsrepräsentation umgesetzt wird, ermöglicht zudem, das Kommunikationsverhalten eines Modells zu betrachten, ohne die komplexe Funktionalität im Detail zu analysieren oder zu kennen. In FunState wird das Verhalten durch ein Petri-Netz ähnliches Netzwerk beschrieben. Die Kommunikation bestimmen Zustandsautomaten, indem sie die Transitionen des Petri-Netzes steuern. In dem SysteMoC-Framework sind bei der Modellierung die Module, die so genannten Aktoren, über FIFO-Kanäle miteinander verbunden. Die Funktionalität der Aktoren wird in C++-Methoden implementiert. Zusätzlich zu diesen Methoden enthält jeder Aktor genau eine Zustandsmaschine, die beim Schalten von Transitionen Methoden des Aktors aufrufen kann. Damit implementiert die Zustandsmaschinen das Kommunikationsverhalten der einzelnen Aktoren – getrennt von der eigentlichen Datentransformation. Es sei darauf hingewiesen, dass die Verwendung der SysteMoC-Bibliothek keine Einschränkung für unsere Arbeit darstellt, da sich jedes SystemC-Modell, welches ausschließlich auf FIFO-Kommunikation beruht, durch ein äquivalentes SysteMoC-Modell darstellen lässt.

Abbildung 2 zeigt das SysteMoC-Modell des Wurzelapproximationsalgorithmus von Newton. Die Quelle *Src* produziert Zufallszahlen, deren Wurzel die Aktoren *SqrLoop*, *Dup* und *Approx* approximieren. Liegt dieser Wert unter einer gegebenen Fehlerschranke, was *SqrLoop* in der Methode *check* prüft, wird er zur Senke *Snk* ausgegeben. Die Ausführung der Aktoren erfolgt parallel, eine Synchronisation erfolgt über die FIFO-Kommunikation. Aus einem Zustand ausgehende Transitionen können nur geschaltet werden, wenn ei-

Listing 1: SystemC-Quellcode des *SqrLoop*-Aktors aus Abbildung 2.

```
class SqrLoop : public smoc_actor {
public:
  smoc_port_in<double>  i1, i2;                                  // input ports
  smoc_port_out<double> o1, o2;                                  // output ports

  SqrLoop(sc_module_name name) : smoc_actor(name, start) {       // FSM
    start = i1(1)                                   >>           // start → loop
            o1(1)                                   >>
            CALL(SqrLoop::copyStore)                >> loop ;
    loop  = (i2(1) &&  GUARD(SqrLoop::check))       >>           // loop → start
            o2(1)                                   >>
            CALL(SqrLoop::copyApprox)               >> start
          | (i2(1) && !GUARD(SqrLoop::check))       >>           // loop → loop
            o1(1)                                   >>
            CALL(SqrLoop::copyInput)                >> loop ;
  }

private:
  double tmp_i1;

  void copyStore()  { o1[0] = tmp_i1 = i1[0]; }                  // actions
  void copyInput()  { o1[0] = tmp_i1;         }
  void copyApprox() { o2[0] = i2[0];          }
                                                                 // guard
  bool check() const { return fabs(tmp_i1 - i2[0]*i2[0]) < 0.0001; }

  smoc_firing_state start, loop;                                 // FSM states
};
```

ne entsprechende *Aktivierungsregel* erfüllt ist. Diese kann aus einem zusammengesetzten Ausdruck bestehen, und wird in der Darstellung durch einen Schrägstrich von einer Aktion getrennt, die bei Schalten der Transition ausgeführt wird. Beispielsweise wird die Transition von *start* nach *loop* in dem Aktor *SqrLoop* betrachtet, die mit „$i_1(1)\&o_2(1)$ / copyStore()" gekennzeichnet ist. Dies bedeutet, dass aus Kanal c_1 ein Token lesbar und in Kanal c_2 Platz für ein Token vorhanden sein muss. Wenn diese Bedingungen erfüllt sind, und die Transition schaltet, wird als Aktion die Methode *copyStore()* ausgeführt. Nach Schalten der Transition werden entsprechend der Aktivierungsregel Token in den FIFO-Kanälen konsumiert und produziert. Zusätzlich zu Kanalfüllständen kann der Rückgabewert einer Methode in der Aktivierungsregel berücksichtigt werden, etwa der Wert der booleschen Methode *check* in der Transition von *loop* nach *start*. Die Kanaltiefen aller Kanäle c_i in diesem Beispiel sind auf acht beschränkt. Zu Beginn befindet sich ein Token auf Kanal c_6.

Listing 1 zeigt den SystemC-Quellcode für den Aktor *SqrLoop* aus Abbildung 2. Zuerst sind *Ports* definiert (Zeile 3 und 4), die später mit FIFO-Kanälen verbunden werden. Innerhalb des Aktors erfolgt der Zugriff auf die verbundenen FIFO-Kanäle über diese Ports. In dem Konstruktor ist die Zustandsmaschine implementiert (Zeile 7 bis 15), deren Transitionen Klassen-Methoden des Moduls als Aktionen aufrufen (Zeile 9, 12, 15) oder in Aktivierungsregeln auswerten (Zeile 10 und 13). Innerhalb einer Funktion darf nur auf Token zugegriffen werden, die durch die Aktivierungsregel abgesichert sind – die Laufzeitumgebung überprüft dies.

Die übersetzten SystemC-Programme, die das SysteMoC-Framework verwenden, können

unter anderem einen Netzwerkgraphen ausgeben. Die Ausgabe enthält die abstrakte Repräsentation des Modells. Genauer: alle Aktoren, deren Zustandsmaschinen inklusive der Aktivierungsregeln und Namen der Aktionen, und die Vernetzung der Aktoren untereinander, inklusive der Tiefe der FIFO-Kanäle.Der Ausgabe fehlt also die eigentliche Implementierung der Methoden. Aber durch die Zustandsmaschinen und die Vernetzung der Aktoren ist das Kommunikationsverhalten des Modells vollständig beschrieben. Abstrahiert wird von dem Inhalt der FIFO-Kanäle, nur deren Füllstand wird betrachtet, und Methoden in den Aktivierungsregeln werden ignoriert. Diese abstrakte Sichtweise auf ein System wird im Weiteren symbolisch dargestellt und zur Modellprüfung verwendet.

3 Intervalldiagramme

Voraussetzung für symbolische Modellprüfung von SystemC-Modellen ist deren symbolische Darstellung. Sowohl für Zustände, als auch für die Zustandsübergangsrelation eines Modells, verwenden wir Intervalldiagramme [ST98]. Mit diesen lassen sich Aktororientierte Modelle, wie die vorgestellten SysteMoC-Modelle mit ihrer asynchronen Kommunikation über FIFO-Kanäle, sehr kompakt und damit effizient repräsentieren. Intervalldiagramme werden nur informell eingeführt, für eine genaue Definition verweisen wir auf [ST98]. Es gibt zwei Arten von Intervalldiagrammen: Intervallentscheidungsdiagramme und Intervallabbildungsdiagramme. Im Folgenden stehen dafür die englischen Akronyme IDD (interval decision diagram) und IMD (interval mapping diagram). IDDs werden für die Darstellung von Modellzuständen, IMDs für die Darstellung von Zustandsübergangsrelationen eingesetzt.

Die bekannteste Art von Entscheidungsdiagrammen sind die binären Entscheidungsdiagramme (BDDs) [Bry86]. IDDs basieren auf dem gleichen Konzept. Wesentlicher Unterschied zu BDDs ist, dass den Variablen nun Definitionsbereiche zugeordnet sind, nicht nur binäre Werte, und an ausgehenden Kanten Intervalle aus dem Definitionsbereich der Variablen annotiert sind. Für Intervalldiagramme ist, im Gegensatz zu BDDS, keine Progammierbibliothek verfügbar. Deshalb wurde eine eigene Entwicklung verwendet. Beispiele für IDDs zeigen Abbildung 3a) und 3b). Der abstrakte Zustand eines Modells, wie in Abschnitt 2 beschrieben, lässt sich wie folgt als IDD kodieren: Für jeden FIFO-Kanal c_i wird eine Variable benötigt, deren Definitionsbereich dem Intervall $[0, n_i]$, mit n_i gleich der Kanaltiefe von c_i, entspricht; für jeden Aktor q_j mit mehr als einem Zustand wird eine Variable mit dem Definitionsbereich $[0, s_i - 1]$, s_i gleich der Anzahl der Zustände von q_j, benötigt. Damit ist der Startzustand des SqrRoot-Beispiels aus Abbildung 2 als IDD in Abbildung 3a) dargestellt. Die Definitionsbereiche sind für die c_i $[0, 8]$, und für q_1 $[0, 1]$, wobei 0 für den Zustand *start* und 1 für den Zustand *loop* steht. Im Startzustand des Modells sind alle Kanäle bis auf c_6 leer. Kanal c_6 enthält ein Initialtoken. Abbildung 3b) stellt die Erreichbarkeitsmenge des SqrRoot-Beispiels dar.

Ein IMD zeigt Abbildung 3c). Darin sind die drei Transitionen des Aktors *SqrLoop* dargestellt – jeder der drei Pfade von der Wurzel zu den Terminalknoten entspricht einer der Transitionen. Im Unterschied zu IDDs sind an den Kanten zwei Intervalle und eine Funktion annotiert, die der Änderung einer Variablen durch die Transition entsprechen. Aus Platzgründen werden die für die Modellprüfung benötigten symbolischen Operationen auf Intervalldiagrammen (wie die *image*-Operation) hier nicht beschrieben. Diese sind ausführlich in [Str00] vorgestellt. Die Komplexität der Operationen entspricht den vergleichbaren Operationen auf BDDs.

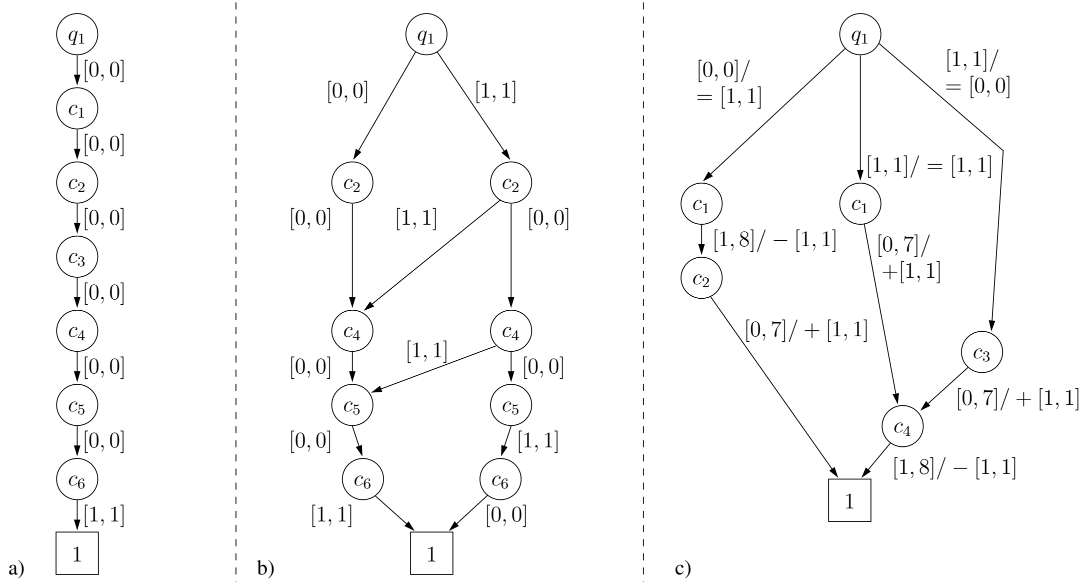

Abbildung 3: a) Anfangszustand und b) Erreichbarkeitsmenge des *SqrRoot*-Beispiels aus Abbildung 2 als IDD. c) ist die Zustandsübergangsrelation des Aktors *SqrLoop* als IMD. Kanten zu den '0'-Terminalknoten sind ausgelassen.

4 Symbolische CTL-Modellprüfung

Wie in Abbildung 1 dargestellt, erhält der Modellprüfer als Eingabe die abstrakte Repräsentation (Initialzustand als IDD und Zustandsübergangsrelation als IMD) eines Modells und eine Spezifikation. Entweder erfüllt das Modell die Spezifikation, und dies wird ausgegeben, oder die Spezifikation wird verletzt, und der Entwickler erhält ein Gegenbeispiel. Die Spezifikation wird in der temporalen Logik CTL formuliert. Deren atomare Präpositionen sind Werte der Variablen der Intervalldiagramme und beziehen sich damit auf die Füllstände von FIFO-Kanälen oder die Zustände der Zustandsmaschinen in den Aktoren. Für das Beispiel aus Abbildung 2 lässt sich etwa die Aussage ,,Immer wenn *Src* ein Token produziert, soll *Snk* auch ein Token konsumieren können“ in CTL als $\mathrm{AG}\,(c_1 > 0 \rightarrow \mathrm{AF}\, c_3 > 0)$ formulieren. Die Überprüfung solcher Formeln erfolgt mit Hilfe bekannter Fixpunktalgorithmen [BCMD90].

Durch die abstrakte Darstellung lassen sich nur Eigenschaften bezüglich Füllstände oder Zustände von Zustandsmaschinen in CTL formulieren. Außerdem kann der Modellprüfer Eigenschaften als nicht erfüllt erkennen, die das zugrunde liegende Modell tatsächlich erfüllt (engl. *false negatives*). Um spezifischere (etwa datenabhängige) Eigenschaften zu prüfen, muss die symbolische Repräsentation verfeinert werden. Eine Verfeinerung ist auch nötig, um *false negatives* zu vermeiden (etwa ähnlich [CGJ+03]).

5 Experimente

In diesem Abschnitt werden Ergebnisse erster Experimente mit dem von uns entwickelten Modellprüfer vorgestellt. Um dessen Effizienz abzuschätzen, wurde die Laufzeit mit der des Modellprüfers NuSMV [CCG+02] verglichen. Als Testmodell haben wir ein Kanban-System gewählt, das in [MV00] untersucht wurde. Dies ist zwar kein eingebettetes System, lässt sich aber mit den gleichen Methoden modellieren und überprüfen. Kanban-Systeme werden zur Produktionsablaufsteuerung eingesetzt und verwenden dazu so ge-

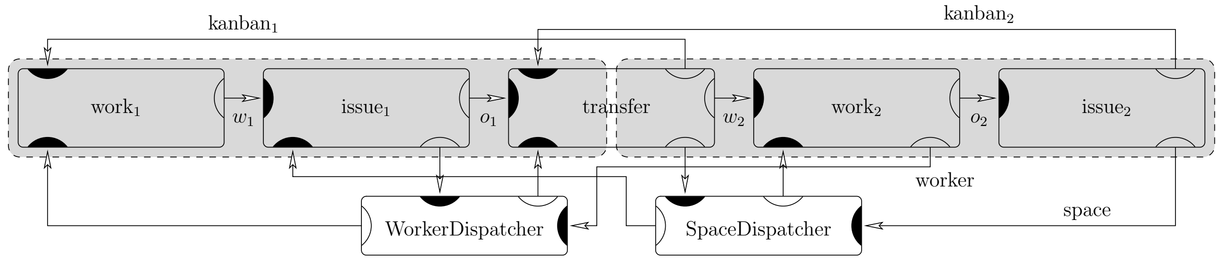

Abbildung 4: Ein Kanban-System mit zwei Produktionsabschnitten und geteilten Ressourcen. Die initiale Anzahl an Kanban-Karten wird durch Initialtoken in den FIFO-Kanälen $kanban_1$ und $kanban_2$ festgelegt, die der geteilten Arbeiter und des Ablageplatzes in den Kanälen *worker* und *space*.

nannte Kanban-Karten. In solchen Systemen darf ein Produktionsabschnitt nur produzieren, wenn eine Kanban-Karte verfügbar ist. Zusätzlich müssen alle zur Produktion benötigten Ressourcen vorhanden sein. Ein solches Modell ist in Abbildung 4 gezeigt. Das Modell ist in zwei Produktionsabschnitte (graue Kästen) aufgeteilt und kann in vier Parametern variiert werden: (1) die Anzahl der Kanban-Karten für den ersten Produktionsabschnitt k_1, (2) die Anzahl der Karten für den zweiten Abschnitt k_2, (3) die Anzahl der zur Verfügung stehen Arbeiter w, die auf beide Abschnitte verteilt werden, und (4) die Anzahl der Behälter für ein produziertes Teil s. Die Ressourcen *worker* und *space* sind geteilt, was zu Verklemmungen führen kann [MV00]. Der Vektor $(k_1, k_2, w, s)^T$ bestimmt die initiale Verteilung von Token in den entsprechenden Kanälen und damit die Komplexität der Modellprüfung. Die FIFO-Kanalgröße aller Kanäle wird auf $f_{size} = max(k_1, k_2, w, s)$ gesetzt. Aus Platzgründen sind die Zustandsmaschinen in Abbildung 4 nicht gezeigt, sondern werden nur textuell beschrieben. Die Zustandsmaschinen der oberen fünf Aktoren enthalten jeweils nur einen Zustand mit einer Selbsttransition, in der aus allen Eingangskanälen ein Token gelesen, und auf allen Ausgangskanälen ein Token produziert wird. Die Zustandsmaschinen der unteren beiden Aktoren enthalten jeweils einen Zustand mit vier Selbsttransitionen. In diesen wird je ein Token konsumiert und eines produziert, in der Art, die alle vier möglichen Verteilungen von Token an den Eingängen zu den Ausgängen abdeckt.

Die IDD/IMD-Repräsentationen der SystemC-Modelle werden automatisch generiert, der Aufwand dafür ist linear zu der Größe des Modells und marginal. Die NuSMV Modelle wurden von Hand implementiert. Messungen mit verschiedenen Vektoren $(k_1, k_2, w, s)^T$ und verschiedenen Variablenordnungen wurden durchgeführt, und sind in Abbildung 5 gezeigt. Aus Platzgründen sind nicht alle Vektoren auf der x-Achse aufgetragen. Da unser Modellprüfer zur Zeit noch keine Gegenbeispiele erzeugt, wurde dieses Merkmal zur Zeitmessung in NuSMV ebenfalls ausgeschaltet. Es wurden immer die beiden CTL-Formeln $\mathrm{EF}\,(o_2 = f_{size})$ und $\mathrm{AG}\,(\mathrm{AF}\,\mathit{INIT_STATE})$ mit $\mathit{INIT_STATE} = (kanban_1 = k_1) \& (kanban_2 = k_2) \& (worker = w) \& (space = s))$ geprüft.

NuSMV wird seit Jahren entwickelt und verwendet ein performantes BDD-Paket, deshalb liegt die Ausführungsdauer unseres Modellprüfers über der von NuSMV. Dies erklärt sich mit dem frühen Entwicklungsstand unseres Intervalldiagrammpakets. Das Paket ist außerdem noch nicht auf Speichereffizienz hin entwickelt, weshalb wir Messungen bezüglich Speicherverbrauch nicht durchgeführt haben.

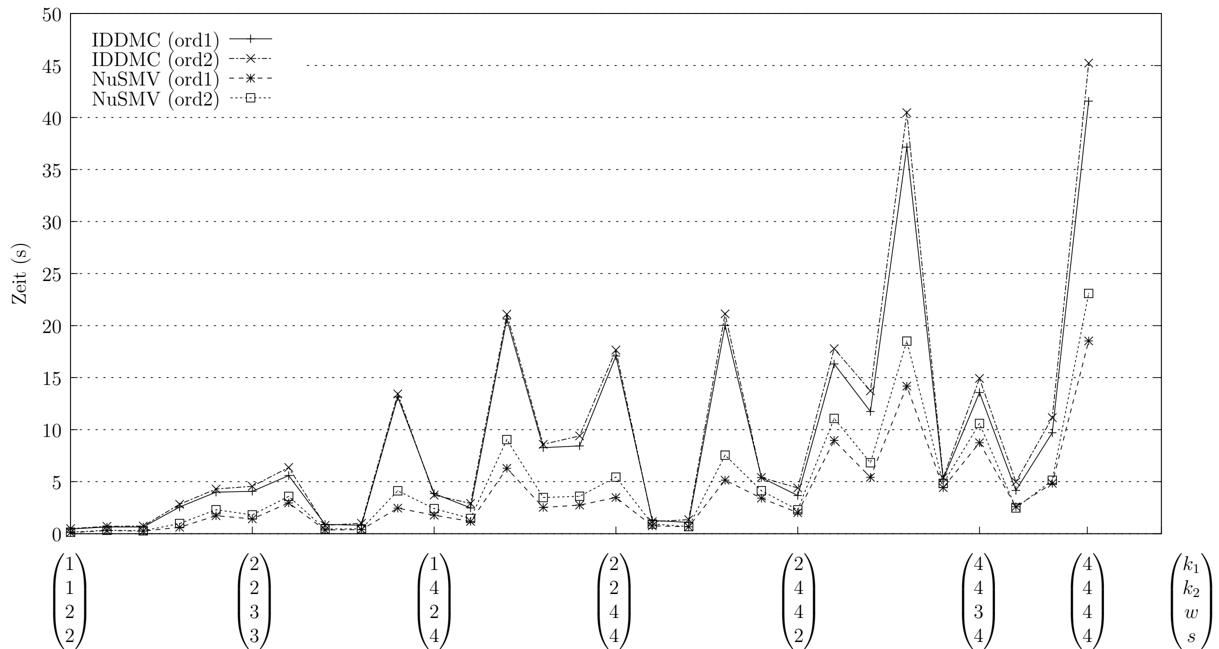

Abbildung 5: Laufzeit der beiden Modellprüfer bei verschiedenen Variablenordnungen und verschiedenen Vektoren $(k_1, k_2, w, s)^T$ für das Kanban-Modell.

6 Zusammenfassung und Ausblick

In diesem Beitrag haben wir gezeigt, wie man Aktor-orientiert modellierte SystemC-Programme abstrakt als Intervalldiagramme repräsentieren kann. Auf Basis dieser Intervalldiagramme wurde von uns eine formale Verifikation von CTL-Formeln implementiert. SystemC-Modelle lassen sich damit bereits automatisch verifizieren. In ersten Experimenten haben wir unseren Ansatz mit dem Werkzeug NuSMV verglichen. Die Ergebnisse sind vielversprechend, da sich das verwendete Intervalldiagrammpaket noch in einem sehr frühen Entwicklungsstadium mit wenig Optimierung befindet, während NuSMV ein ausgereiftes Programm ist. Viele für BDD-Implementierungen bekannte Optimierungen können auf das Intervalldiagrammpaket übertragen werden, und damit dessen Effizienz steigern, was das vorgestellte Verfahren wesentlich beschleunigt. Außerdem können IDDs durch ihre kompakte Repräsentation in Bezug auf den Speicherverbrauch Vorteile gegenüber BDD-basierten Ansätzen haben, wie andere Ergebnisse zeigen [Str00]. Dieser Vorteil kann die Analyse komplexerer Systeme ermöglichen. Dies werden wir in weiterführenden Arbeiten untersuchen.

Literatur

[BCCZ99] Biere, Armin, Alessandro Cimatti, Edmund M. Clarke und Yunshan Zhu: *Symbolic Model Checking without BDDs.* In: *TACAS '99: Proceedings of the 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems*, Seiten 193–207, London, UK, 1999. Springer-Verlag.

[BCMD90] Burch, J.R., E.M. Clarke, K.L. McMillan und D.L. Dill: *Sequential circuit verification using symbolic model checking.* In: *Design Automation Conference, 1990. Proceedings. 27th ACM/IEEE*, Seiten 46–51, 1990.

[Bry86] Bryant, Randal E.: *Graph-Based Algorithms for Boolean Function Manipulation.* IEEE Trans. Comput., 35(8):677–691, 1986.

[CCG+02] Cimatti, Alessandro, Edmund M. Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani und Armando Tacchella: *NuSMV 2: An OpenSource Tool for Symbolic Model Checking.* Seiten 359–364. Springer-Verlag, 2002.

[CES86] Clarke, E. M., E. A. Emerson und A. P. Sistla: *Automatic verification of finite-state concurrent systems using temporal logic specifications.* ACM Trans. Program. Lang. Syst, 8:244–263, 1986.

[CGJ+03] Clarke, Edmund, Orna Grumberg, Somesh Jha, Yuan Lu und Helmut Veith: *Counterexample-guided abstraction refinement for symbolic model checking.* J. ACM, 50(5):752–794, 2003.

[FHT06] Falk, Joachim, Christian Haubelt und Jürgen Teich: *Efficient Representation and Simulation of Model-Based Designs in SystemC.* In: *Proc. FDL'06*, Seiten 129 – 134, Darmstadt, Germany, September 2006.

[GKD06] Grosse, Daniel, Ulrich Kühne und Rolf Drechsler: *HW/SW co-verification of embedded systems using bounded model checking.* In: *GLSVLSI '06: Proceedings of the 16th ACM Great Lakes symposium on VLSI*, Seiten 43–48, New York, NY, USA, 2006. ACM Press.

[GLMS02] Grötker, Thorsten, Stan Liao, Grant Martin und Stuart Swan: *System Design with SystemC.* Kluwer Academic Publishers, 2002.

[KEP06] Karlsson, Daniel, Petru Eles und Zebo Peng: *Formal verification of systemc designs using a petri-net based representation.* In: *DATE '06: Proceedings of the conference on Design, automation and test in Europe*, Seiten 1228–1233, 3001 Leuven, Belgium, Belgium, 2006. European Design and Automation Association.

[KS05] Kroening, D. und N. Sharygina: *Formal verification of SystemC by automatic hardware/software partitioning.* In: *Formal Methods and Models for Co-Design, 2005. MEMOCODE '05. Proceedings. Third ACM and IEEE International Conference on*, Seiten 101–110, 2005.

[LBW+07] Lämmermann, S., J. Behrend, R. J. Weiss, J.Ruf, T. Kropf und W. Rosenstiel: *UML/SysML-Systemanalyse zur Generierung von formalen Verifikationseigenschaften für verschiedene Abstraktionsebenen.* In: *10. Workshop Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen*, Seiten 29–38, Erlangen, Germany, März 2007.

[LNW03] Lee, Edward A., Stephen Neuendorffer und Michael J. Wirthlin: *Actor-Oriented Design of Embedded Hardware and Software Systems.* Journal of Circuits, Systems, and Computers, 12(3):231–260, 2003.

[McM93] McMillan, K. L.: *Symbolic Model Checking.* Kluwer Academic Publishers Norwell, MA, USA, 1993.

[MV00] Magnino, F. und P. Valigi: *A Petri net approach to deadlock analysis for classes of kanban systems.* In: *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, Band 3, Seiten 2877–2882 vol.3, 2000.

[ST98] Strehl, Karsten und Lothar Thiele: *Symbolic model checking of process networks using interval diagram techniques.* In: *ICCAD '98: Proceedings of the 1998 IEEE/ACM international conference on Computer-aided design*, Seiten 686–692, San Jose, California, United States, 1998. ACM.

[Str00] Strehl, K.: *Symbolic Methods Applied to Formal Verification and Synthesis in Embedded Systems Design.* Doktorarbeit, Eidgenössische Technische Hochschule (ETH) Zürich, 2000.

[TSZ+99] Thiele, L., K. Strehl, D. Ziegengein, R. Ernst und J. Teich: *FunState-an internal design representation for codesign.* In: *Computer-Aided Design, 1999. Digest of Technical Papers. 1999 IEEE/ACM International Conference on*, Seiten 558–565, 1999.

[Var07] Vardi, Moshe Y.: *Formal Techniques for SystemC Verification; Position Paper.* In: *Design Automation Conference, 2007. DAC '07. 44th ACM/IEEE*, Seiten 188–192, 2007.