

Model Checker SPIN

Hauptseminar: Modellüberprüfung

Kathrin Ott

15. Februar 2002

Inhaltsverzeichnis

1	Einleitung	3
2	SPIN und PROMELA	4
2.1	Entstehung	4
2.2	Struktur	4
2.3	Promela	6
2.4	Eigenschaften	8
3	Temporale Logik	9
3.1	LTL	10
4	Algorithmus	12
4.1	Nested Depth-First-Search	12
4.2	Büchiatomat	12
4.3	Partial Order Plan	13
5	Zusammenfassung	15
6	Referenzen	16

1 Einleitung

SPIN (Simple Promela INterpreter) ist ein Softwarepaket, das in den 80er Jahren von einer Forschungsgruppe der Bell Labs entwickelt wurde. Dieses Softwarepaket wird zur formalen Überprüfung verteilter Systeme (z.B. der Software eines Telefonsystems) verwendet. Mit den Überprüfungsmodellen von SPIN kann man die Korrektheit von Prozeßinteraktionen überprüfen, beispielweise Nachrichtenübertragung in gepufferten Kanälen. Das Design von SPIN ermöglicht eine effiziente Überprüfung von asynchronen Softwaresystemen. Es beinhaltet eine Spezifikationssprache, Logik, Überprüfungsprozeduren, Reduktionstechniken und Zustandskodierungsmethoden [1].

In den folgenden Kapiteln werde ich auf die unterschiedlichen Details von SPIN eingehen, beginnend mit der Entstehung (Kapitel 2.1) von SPIN in den 80er Jahren. Die Struktur (Kapitel 2.2) von SPIN läßt sich in mehrere Schritte einteilen. Die Überprüfung beginnt mit der Spezifizierung eines high-level Modells und dessen Überprüfung auf Syntaxfehler. Dann wird ein optimiertes Überprüfungsprogramm mit SPIN generiert. Dieses Programm wird dann mit verschiedenen Kompilierungszeiten kompiliert, um den entsprechenden Reduktionsalgorithmus zu finden. Das jeweilig positive oder negative Ergebnis wird entsprechend behandelt. In Kapitel 2.3 wird auf die Designsprache von SPIN namens PROMELA (PROcess MEta LAnguage) eingegangen. PROMELA's Semantik kann als ein Modell gesehen werden, das einen oder mehrere Prozesse, keine oder mehrere Variablen und keine oder mehrere Nachrichten Kanäle enthält. Die Aktionen der Prozesse werden durch eine Art Maschine zeitlich gesteuert[5]. In Kapitel 2.4 werden die Eigenschaften von SPIN gezeigt, die dieses Softwarepaket von anderen Softwarepaketen zur Modellüberprüfung unterscheidet.

In Kapitel 3 wird auf Temporale Logik und Lineare Temporale Logik (LTL) eingegangen. Temporale Logik unterscheidet sich von der herkömmlichen Logik durch die zeitliche Abhängigkeit. LTL ist eine bestimmte Temporale Logik, die den Zustandsgraphen linear durchläuft und deren Operatoren die Eigenschaften jedes möglichen Berechnungspfad beschreiben [4]. LTL wird verwendet, weil SPIN Korrektheitseigenschaften akzeptiert, welche in LTL ausgedrückt werden können.

In Kapitel 4 wird auf einige der Algorithmen eingegangen, die von SPIN verwendet werden, wie z.B. Nested Depth-First-Search (4.1), Büchautomat (4.2) und Partial Order Plan (4.3). Diese (Reduktions-) Algorithmen dienen zur Optimierung von Zeit und Speicherraum der Simulationsdurchläufe und derer Ergebnisse.

2 SPIN und PROMELA

2.1 Entstehung

SPIN hat seine Wurzeln in den anfänglichen Protokollüberprüfungssystemen, welche auf on-the-fly (d.h. es wird kein Zustandsgraph konstruiert) Reachability Analyse der 80er Jahre basiert. Der Zweck dieses ersten Überprüfungs-tools war, dass es in der Lage war Probleme mit praktischer Signifikanz lösen zu können. Die grundlegende Berechnungskomplexität eines zu lösenden Problems fordert eine sorgfältige Auswahl der Eigenschaften, welche unterstützt werden sollen. Man muss also zwischen praktischer Signifikanz des Werkzeugs und Berechnungskomplexität abwägen. Die Vorgänger von SPIN haben dafür nur Standardsicherheitseigenschaften und eine limitierte Auswahl von „Liveness“ Eigenschaften überprüft. Somit wurden anfangs nur logische Modellüberprüfungstechniken verwendet, welche auf logischen mathematischen Ausdrücken beruhten. Diese Technik wurde später durch die Automatentheorie erweitert, was schließlich zur Grundlage von temporaler logischer Modellüberprüfung im SPIN System führte.

Seit 1991 ist SPIN kostenlos erhältlich. Es kann von der Webseite der Bell-Labs bezogen werden. Mittlerweile ist das System auf tausenden Rechnern installiert. Es wird für verschiedene wissenschaftliche und praktische Zwecke genutzt.

Seit 1995 finden jährlich internationale Workshops statt. Diese Workshops befassen sich mit der Erweiterung von Spin und PROMELA, sowie der Verringerung von Speicherverbrauch und Zeit und der Vereinfachung der einzelnen Algorithmen.

2.2 Struktur

In der folgenden Abbildung wird die Grundstruktur der Arbeitsweise von SPIN gezeigt. Zuerst wird die Spezifikation eines High Level Models eines nebenläufigen Systems, oder eines verteilten Algorithmus im Front-End XSPIN erstellt. Die Designspezifikation wird in PROMELA geschrieben. XSPIN ist ein Programm, welches SPIN im Hintergrund ablaufen lässt und den Output von SPIN graphisch darstellt. XSPIN ist somit eine grafische Oberfläche, die die Resultate von SPIN anschaulicher macht. Die Befehle für SPIN lassen sich einfach aus dem Menü von XSPIN auswählen und XSPIN weiß, wann und wie es von SPIN generierten Code ausführen kann [3]. Es vereinfacht somit die Handhabung von SPIN. Nachdem man die Syntaxfehler im High Level Modell korrigiert hat, wird eine interaktive Simulation durchgeführt bis sich das Design wie geplant verhält. Im folgenden Schritt, wird mit Hilfe von SPIN ein optimiertes on-the-fly Prüfungsprogramm einer High Level Spezifikation generiert. Dieses Prüfungsprogramm wird wahlweise mit verschiedenen Kompilierungszeiten kompiliert und ausgeführt. Dadurch lassen sich dann die entsprechenden Reduzierungsalgorithmen ermitteln, auf die in Kapitel 4 eingegangen wird. Sollte das zu prüfende Programm nicht die erwarteten Resultate erbringen, so kann dies auf Fehler

im Programm zurückzuführen sein. Falls somit eine Voraussetzung nicht erfüllt wird und ein Gegenbeispiel gefunden wird, kann dieses in einen interaktiven Simulator zurückgeführt werden. Dann kann im Detail die Ursache untersucht und möglicherweise gefunden und entfernt werden [1]. Die Korrektheitsanforderungen, die an das zu prüfende System gestellt werden, werden in LTL Syntax formuliert. Wie das genau gemacht wird, wird in Kapitel 3 beschrieben.

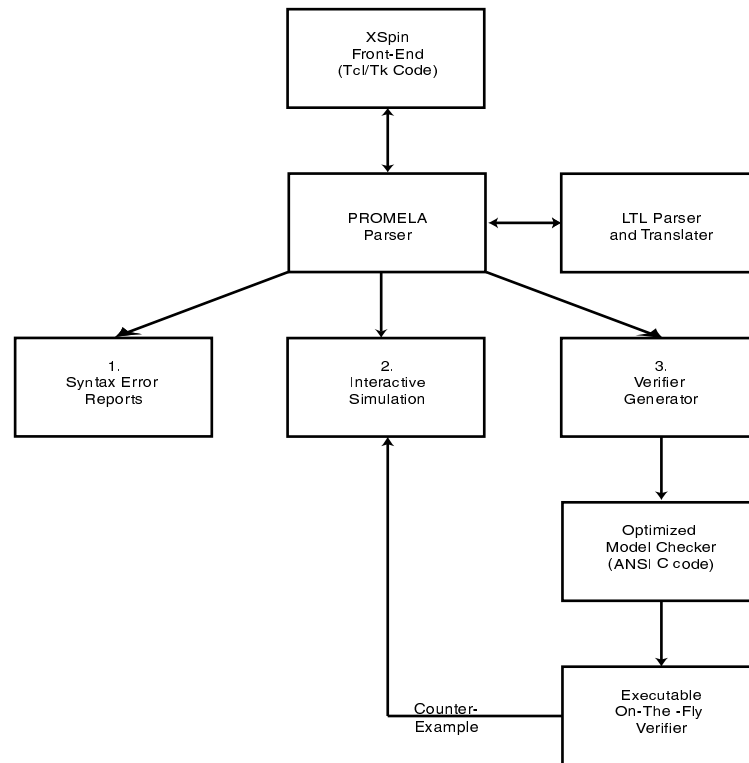


Abbildung 1: Struktur der Simulation und Verifikation von SPIN

Die Beschreibung eines nebenläufigen Systems in PROMELA besteht aus einem oder mehreren benutzerdefinierten Prozessvorlagen oder Prototypdefinitionen und zumindest einer Prozessrealisierung. Somit kann jeder laufende Prozess weitere asynchrone Prozesse instanziiieren, wenn er das Prozessstemplate benutzt, welches die Verhalten der verschiedenen Prozesse definiert. SPIN übersetzt dann jedes Prozessstemplate in einen Zustandsautomaten. Das globale Verhalten des nebenläufigen Systems wird durch das Berechnen eines Produkts von Automaten erhalten. Das Produkt ist asynchron und überlappend. Es gibt jeweils einen Automat pro asynchronem Prozessverhalten. Das daraus resultierende globale Systemverhalten ist selbst wiederum durch einen Automaten repräsentiert. Dieses überlappende Produkt wird normalerweise als Zustandsraum des Systems bezeichnet. Man kann das Produkt durch einen Graphen repräsentieren, den

man allgemein auch Reachabilitygraph nennt.

Zur Überprüfung eines Systems spezifiziert SPIN eine Korrektheitsanforderung in Form einer temporalen Logikformel. Diese Formel wird dann in einen Büchautomaten konvertiert und berechnet das synchrone Produkt dieser Anforderung und den Automaten, der den globalen Zustandsraum repräsentiert. Das Resultat ist wiederum ein Büchautomat. Falls die Sprache, die von diesen Automaten akzeptiert wird leer ist, bedeutet dies, dass die ursprüngliche Anforderung an das gegebene System nicht erfüllt werden. Falls die Sprache nicht leer ist, beinhaltet sie genau die Verhalten, welche die originale temporalen Logikformel erfüllen. In SPIN werden Korrektheitsanforderungen (und temporallogische Formeln) benutzt um fehlerhafte bzw. unerwünschte Systemverhalten zu formulieren. Der Überprüfungsvorgang bezeugt dann entweder, dass solche Verhalten nicht möglich sind oder er hält detaillierte Beispiele von passenden Verhalten bereit.

Im schlimmsten Fall hat der globale Reachabilitygraph die Größe des Kartesischen Produkts aller Teilsysteme. Die Spezifikationssprache Promela ist so definiert, dass jeder Teil immer einen streng begrenzten Umfang hat. Dies gilt für Prozesse, welche nur endlich viele Kontrollzustände haben können. Es gilt aber auch für alle lokalen und globalen Variablen, welche nur endlich viele unterschiedliche Werte haben können. Ebenso gilt es für alle Nachrichtenkanäle mit gebundener und benutzerdefinierter Kapazität. Obwohl in der Praxis die Größe der globalen Reachability größtmäßig nie den schlimmsten Fall erreicht, kann der Teil des kartesischen Produkts, der erreicht wird leicht sehr groß werden, bis er vollständig konstruiert ist. Es wurden einige Techniken entwickelt um die Komplexität dieses Problem zu managen.

2.3 Promela

SPIN akzeptiert Designspezifikationen, welche in der Designsprache PROMELA (PROzess MEta LAnguage) geschrieben sind. Korrektheitsanforderungen, welche in der Syntax von Standard LTL spezifiziert sind werden von SPIN akzeptiert. Um ein Design zu prüfen wird ein formales Modell gebaut, welches PROMELA verwendet. Dies bedeutet, dass alle Korrektheitseigenschaften automatisch formal innerhalb der Beschränkungen entscheidbar werden. Beschränkungen können beispielsweise durch die Größe des Problems und berechenbarer Ressourcen, welche vom Model Checker genutzt werden können, um die Beweise zu erbringen, hervorgerufen werden. Natürlich haben alle Prüfungssysteme physikalische Grenzen, wie z.B. Problemgröße, Maschinenspeichergöße und maximale (geplante) Laufzeit. Diese Beschränkungen sind ein oft vernachlässigtes Problem in formaler Verifizierung. Deshalb untersucht man die Grenzen des Model Checkers explizit und bietet Strategien zur Erleichterung des Problems an [1].

Wie schon erwähnt, können die Korrektheitsanforderungen in der Syntax von LTL ausgedrückt werden. Eine temporallogische Aussage: p bleibt wahr zumindest solange bis q wahr wird, kann wie folgt in PROMELA Syntax formuliert werden.

```

$ spin -f „ $\Box(pUq)$ “
never {
T0:
    if
    :: (p) → goto T0
    :: (q) → goto accept
    fi;
accept:
    if
    :: ((p) || (q)) → goto T0
    fi;
}

```

Die Aussage: zu jeder Zeit in der Ausführung wird p zumindest noch einmal wahr werden, kann wie folgt ausgedrückt werden.

```

$ spin -f „ $\Box(\Diamond p)$ “
never {
T0:
    if
    :: (true) → goto T0
    :: (p) → goto accept
    fi;
accept:
    if
    :: (true) → goto T0
    fi;
}

```

(Die beiden Programme entsprechen den Büchautomaten in Abbildung 3 in Kapitel 4.2.)

T0 und accept sind die zwei Zustände des Automaten. Je nach Eingabe, p oder q, wird der Zustand geändert oder bleibt gleich. In diesem Programm kommen nur if (..fi) Abfragen vor.

Die Syntax von PROMELA ist jedoch weit umfangreicher. Beispielsweise können mit #define <Variablenname> <Integer> am Anfang eines Programms Variablen definiert werden. Es können Typen und Variablen von Typen definiert werden, z.B. mtype = A,B,C; und mtype ele = A; d.h., dass ele eine Variable vom Typ A ist. Eine weitere wichtige Anweisung, neben der if Anweisung ist do (..od). Außer Variablen können natürlich auch Nachrichten, Kanäle, Prozesse und Übergänge definiert werden [5]. Dies wird vorallem benötigt, wenn ein verteiltes System überprüft wird. Ich werde darauf allerdings nicht genauer eingehen.

2.4 Eigenschaften

Spin hat einige Eigenschaften, die diese Software von anderen unterscheidet.

- Die Prüfungsmodelle von SPIN konzentrieren sich darauf, die Korrektheit von Prozessinteraktionen zu belegen. Sie versuchen auch so viel wie möglich von internen, aufeinanderfolgenden Berechnungen zu abstrahieren. Prozessinteraktionen können in SPIN als Rendezvous Grundzustände, als asynchrone Nachrichtenweiterleitung durch gepufferte Kanäle, durch Zugriff auf gemeinsam genutzte Variablen oder als irgendeine Kombination dieser spezifiziert werden.
- SPIN findet logische Designfehler im Design verteilter Systeme, wie z.B. Betriebssystemen, Datenkommunikationsprogrammen, Schaltsystemen, nebenläufigen Algorithmen, Schienensignalprotokollen etc.
- SPIN ist ein Werkzeug, das die logische Konsistenz einer Spezifikation überprüfen kann. Es meldet deadlocks, unspezifizierte Empfänge, Flag-Unvollständigkeiten, Laufbedingungen und nicht berechnete Aussagen über die Geschwindigkeit von Prozessen.
- SPIN arbeitet on-the-fly, was bedeutet, dass es keinen globalen Zustandsgraphen konstruiert, was normalerweise als eine Voraussetzung zur Überprüfung jedes Systems gilt.
- SPIN kann als ein komplettes LTL Modellüberprüfungssystem benutzt werden, welches alle Korrektheitsvoraussetzungen, die in der LTL ausgedrückt werden können, unterstützt. Es kann aber auch als ein effizienter on-the-fly Überprüfer für mehr Sicherheit und Liveness Eigenschaften verwendet werden. Zur Vollständigkeit muss man jedoch erwähnen, dass es auch viele Eigenschaften gibt, welche ohne den Gebrauch von LTL ausgedrückt und überprüft werden können.
- SPIN unterstützt sowohl Rendezvous als auch gepufferte Nachrichtenweiterleitung und Kommunikation durch gemeinsam genutzten Speicher. Gemischte Systeme die synchrone und asynchrone Kommunikation benutzen, werden von SPIN auch unterstützt.
- SPIN unterstützt zufällige, interaktive und gelenkte Simulation und Teilbeweistechniken sowie eingehende Beweistechniken.
- SPIN ist so designed, dass es reibungslos mit der Problemgröße wachsen kann und sogar sehr große Problemumfänge bewältigen kann. Um die Überprüfungsdurchläufe zu optimieren, nützt SPIN die Partial Order Reduction Technik aus und (optional) BDD (Binary Decision Diagram) ähnliche Speichertechniken effizient aus.
- SPIN Software ist in ANSI standard C geschrieben und ist auf allen Versionen der UNIX Betriebssysteme lauffähig. Es kann auch auf allen Standard

PCs welche unter Linux, Windows95/98 oder WindowsNT kompiliert werden.

SPIN kann in drei Grundfunktionen benutzt werden:

1. als ein Simulator, der ein schnelles Prototyping mit zufälligen, gelenkten oder interaktiven Simulationen ermöglicht.
2. als ein vollständiger Zustandsraumanalysierer, der in der Lage ist, uneingeschränkt die Richtigkeit der vom Benutzer spezifizierten Korrektheitsansprüche zu beweisen.
3. als ein Bit-State Analysierer, der sogar sehr große Protokollsysteme mit maximaler Abdeckung des Zustandsraums validieren kann.

Viele der Eigenschaften von SPIN können ohne den Gebrauch von linearer temporaler Logik (LTL) ausgedrückt und überprüft werden. Die Korrektheitseigenschaften können (mittels Behauptungen) als System oder Prozessinvariante oder als übliche linear-temporallogische Bedingungen spezifiziert werden. Die LTL Bedingungen können wiederum entweder direkt mit der Syntax von LTL, oder indirekt als Büchautomat angegeben werden. Wie SPIN LTL und den Büchautomaten verwendet wird in den folgenden Kapiteln erklärt.

3 Temporale Logik

Temporale Logik ist eine Erweiterung der „klassischen“ (mathematischen) Logik, bei der Aussagen zu verschiedenen Zeitpunkten (meist als ‚bis‘, ‚seit‘, ‚im nächsten Zustand‘, etc. angegeben) verschiedene Wahrheitswerte haben können. Temporale Logiken werden verwendet um etwas über die Verhalten, die von einem Zustandsübergangsgraph abgeleitet werden können, auszusagen [4]. Ein endliches Zustandssystem kann durch ein Tupel, der Zustandsübergangsgraph genannt wird, beschrieben werden:

$$M = \langle S, T, R, L \rangle$$

- **S** ist eine endliche Menge von Zuständen
- **T** \subseteq **S** ist eine Menge von Grundzuständen
- **R** \subseteq **S** \times **S** ist die Übergangsrelation, die mögliche Übergänge von Zustand zu Zustand beschreibt
- **L** ist eine Funktion

Das Verhalten in einem Zustandsübergangsgraph erhält man wie folgt. Man startet an einem Zustand $s \in T$ und hängt dann wiederholt Zustände, welche durch **R** erreichbar sind, an. Es wird vorausgesetzt, dass die Übergangsrelationen **R** vollständig sind. Als Konsequenz existieren unendliche viele Verhalten des Systems. Nachdem ein Zustand mehr als einen Nachfolger haben kann, kann

man sich die Struktur aufgespannt als einen unendlichen Baum vorstellen. Dieser Baum repräsentiert alle möglichen Ausführungen eines Systems, welches am Anfangszustand startet. In der Grafik sieht man einen Zustandsübergangsgraph und seinen aufgespannten Baum vom Zustand „A“ aus.

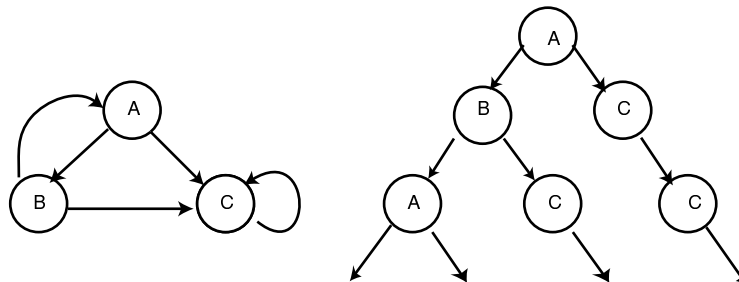


Abbildung 2: Normale und lineare Darstellungen eines Zustandsübergangsgraphen

3.1 LTL

Eine spezielle temporale Logik ist die Linear Temporal Logig (LTL). In LTL können die Operatoren die Eigenschaften von allen möglichen Berechnungspfaden beschreiben.

SPIN akzeptiert Korrektheitseigenschaften die sich in LTL Formeln ausdrücken lassen, welche in einen Büchautomaten übersetzt werden können [1]. SPIN verrichtet die Umwandlung zum Büchautomaten mechanisch. Formal generierte Automaten akzeptieren nur die unendlichen Systemausführungen, welche die dazu passenden LTL Formel erfüllen.

LTL Formeln können benutzt werden um sowohl Sicherheitseigenschaften als auch Livenessseigenschaften auszudrücken. Eine LTL Formel f kann ein Symbol p enthalten, kombiniert mit unären oder binären, booleschen oder zeitlichen Operatoren. Zum Beispiel kann die Aussage: p bleibt wahr zumindest solange bis q wahr wird, als $\Box(pUq)$ dargestellt werden.

Die Aussage: zu jeder Zeit in der Ausführung wird p zumindest noch einmal wahr werden als $\Box(\Diamond p)$ dargestellt werden.

Weitere LTL Grammatik:

```

f ::=
      p
      | true
      | false
      | (f)
      | f binop f
      | unop f
  
```

unop::= \square (immer)
 | \diamond (eventuell)
 | ! (logische Negation)

binop::= U (starkes bis)
 | $\&\&$ (logisches und)
 | $\|\|$ (logisches oder)
 | \rightarrow (Implikation)
 | \leftrightarrow (Gleichheit)

Man benutzt Korrektheitsforderungen um Verhalten, von denen angenommen wird das sie unmöglich seien, um ein System zu formulieren. Damit versucht man beispielsweise potentiell mögliche Angriffe gegen korrektes Verhalten des Systems zu finden. Man kann jede positive LTL Formel durch voranstellen eines logischen Negationsoperators in eine negative umformen und umgekehrt. Zuerst mag es scheinen, als ob es keinen Unterschied machen würde, aber es gibt einen Unterschied.

Eine positive Forderung verlangt, dass die Sprache des Systems in der Sprache der Forderungen eingeschlossen ist. Dagegen verlangt eine negative Forderung den Beweis, dass die Schnittmenge der Sprache des Systems und der Forderung leer ist. Die Größe des Zustandsraums für den „positiven“ Beweis ist höchstens so groß wie das kartesische Produkt des Systems und der Forderung und ist mindestens so groß wie deren Summe. Um Leerheit bei einer negativen Forderung zu beweisen ist im schlimmsten Fall die Größe des Zustandsraums immer noch die Größe des Kartesischen Produkts des Systems und der Anforderungen. Im besten Fall ist die Größe des Zustandsraums jedoch Null.

Die Schnittmenge beinhaltet keinen Zustand, wenn keine Anfangsmenge eines ungültigen Verhaltens welches durch die Forderungen repräsentiert wird im System erscheint. Deswegen arbeitet SPIN mit negativen Korrektheitsforderungen und löst das Verifikationsproblem durch Schnittmengen von Sprachen.

Ein Büchautomat akzeptiert eine Systemausführung nur dann, wenn die Ausführung den Büchautomaten zwingt durch einen oder mehrere seiner akzeptierenden Zustände unendlich oft durchzugehen. Man nennt solch ein Verhalten Akzeptierungszyklus. Hinzuzufügen ist noch, dass eine unendliche Ausführung in einem endlichen System nur dann existiert, wenn das Verhalten zyklisch ist. Man will, dass keine Ausführungssequenz dieses Systems mit der negierten Korrektheitsforderung übereinstimmt. Dazu genügt es die Abwesenheit des Akzeptanzzyklus in der kombinierten Ausführung (System und Büchautomat, der die Forderungen vertritt) zu zeigen. Die kombinierte Ausführung ist durch ein synchrones Produkt eines Systems und einer Forderung formal definiert.

Die ganze Berechnung, die mit den individuellen simultanen Komponenten und einem einzelnen Büchautomaten (der den Korrektheitsanspruch repräsentiert) beginnt, wird von SPIN übernommen. Dies geschieht in einer einzigen Prozedur, welche den „nested depth first search“ Algorithmus benutzt. Der Algorithmus terminiert, wenn ein Akzeptanzzyklus gefunden ist, wodurch dann

ein Gegenbeispiel zu den Korrektheitsforderungen gestellt wird, oder wenn kein Gegenbeispiel existiert, nachdem das komplette Produkt der Schnittmenge berechnet wurde.

4 Algorithmus

SPIN verwendet verschiedene Algorithmen und Methoden, beispielsweise um einen Graphen zu durchlaufen. Dabei ist eine Methode um den Ablauf in einem Zyklus zu erkennen sehr bedeutend. Von der Methode wird verlangt, dass sie mit allen Arten der Überprüfung, welche erschöpfende Suche, Bitzustandszerlegung und Partial Order Reduction Techniken einschließen, kompatibel sind.

4.1 Nested Depth-First-Search

Damit ein Akzeptanzzyklus im Reachabilitygraph existiert, muss mindestens ein akzeptierter Zustand vom Anfangszustand (der Wurzel des Graphs) und von sich selbst aus erreichbar sein. Die erste depth first Suche ermittelt welcher akzeptierten Zustände vom Anfangszustand des Systems erreichbar sind. Die zweite Suche beginnt an jeden akzeptierten Zustand der gefunden wurde. Dann wird überprüft ob der Zustand von sich selbst aus erreichbar ist. Falls er es ist, wurde auch eine komplette Ausführungssequenz, welche den Akzeptanzzyklus enthält konstruiert.

In SPIN gleicht diese Ausführungssequenz immer einem Gegenbeispiel einer benutzerdefinierten Korrektheitsforderung. Die Ausführungssequenz kann als Beweis dafür, dass die Korrektheitsforderung für das System wie es spezifiziert ist ungültig ist ausgegeben werden.

Der nested depth first Suchalgorithmus ist jedoch leider nicht in der Lage alle möglichen Akzeptanzzyklen, die im Reachabilitygraph auftauchen können zu finden. Es kann jedoch bewiesen werden, dass zumindest ein solcher Zyklus entdeckt wird, falls ein oder mehrere existieren. Weil Akzeptanzzyklen in SPIN Gegenbeispiele zu Korrektheitsforderungen erzeugen, genügt es für den Zweck der Überprüfung immer entweder ihre Abwesenheit oder Anwesenheit zu zeigen.

In SPIN ist der nested depth first Suchalgorithmus noch um einen optionalen weak fairness constraint erweitert. Damit wird jedem Prozess, der mindestens einen Übergang enthält, der unendlich lang aktiv bleibt, garantiert diesen Übergang innerhalb endlicher Zeit auszuführen.

4.2 Büchiatomat

Ein Automat, der von SPIN generiert werden kann, kann in PROMELA Syntax geschrieben werden. Nimmt man die zwei LTL Formeln $\Box(pUq)$ (p bleibt wahr zumindest solange bis q wahr wird) und $\Box(\Diamond p)$ (zu jeder Zeit in der Ausführung wird p zumindest noch einmal wahr werden) und macht zwei Automaten daraus, so beinhalten beiden einen nicht akzeptierenden Zustand (den Anfangszustand des Büchiatomaten, T0) und einen akzeptierenden Zustand (accept).

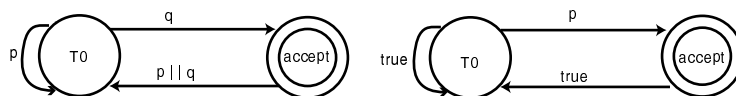


Abbildung 3: Büchiautomaten

Die Zustände T0 und accept beinhalten beide eine nichtdeterministische Auswahl (if..fi) im PROMELA Programm (Seite 7). Die Zustände T0 haben zwei möglichen Übergängen p und q bzw. p und true. Für die Zustände accept, gibt es nur eine Option p || q bzw. true. Jeder Zustandsübergang im Automaten ist von der vorgegebenen Formel abhängig, z.B. sind bei diesen if..fi Konstrukten die Konditionen an den Zustandsübergängen des Automaten angebracht und nicht an den Zuständen selbst.

Um eine LTL-Formel in einen Automaten zu konvertieren besitzt SPIN einen Übersetzungsalgorithmus, der die Zustände für den Büchiautomaten errechnet. Dazu muss der Übersetzungsalgorithmus eine Menge von Unterformeln berechnen, welche in jedem erreichbaren Zustand und in jedem nachfolgenden Zustand erfüllbar sein müssen. Die Formel wird zuerst in eine Normalform umgewandelt, die nur an den atomaren Aussagen Negierungen hat. Ein Anfangszustand wird geschaffen, der mit einer passenden Formel und einer Pseudokante markiert wird. Der Rest des Automaten wird dann rekursiv berechnet. Bei jedem Durchlauf wird eine Unterformel bearbeitet, die noch zu erfüllen ist, wobei je nach führendem Operator der aktuelle Zustand in zwei neue Zustände aufgeteilt wird, die jeweils unterschiedliche Teile der Subformel enthalten. In der letzten Phase der Übersetzung werden manche der Zustände als akzeptierend identifiziert, je nach Anwesenheit oder Abwesenheit von Unterformeln mit until Operatoren [1].

4.3 Partial Order Plan

SPIN benützt Partial Order Reduction Methoden um die Anzahl der erreichbaren Zustände, welche zur Vervollständigung der Überprüfung erforscht werden müssen zu reduzieren. Die Reduktion basiert auf einer Beobachtung. Diese Beobachtung zeigt, dass die Gültigkeit einer LTL Formel oft unsensibel gegenüber der Reihenfolge, in welcher simultane und unabhängig voneinander ausgeführte Ereignisse in der depth first Suche ineinander verschachtelt sind, ist. Anstatt einen erschöpfenden Zustandsraum zu generieren, der alle Ausführungssequenzen als Pfade enthält, kann der Prüfer einen verkleinerten Zustandsraum generieren. Dieser verringerte Zustandsraum beinhaltet nur Vertreter von verschiedenen Ausführungssequenzen, welche für eine gegebene Korrektheitseigenschaft nicht unterscheidbar sind. Die Implementierung dieser Reduzierungsmethode basiert auf einer statischen Reduzierungstechnik. Diese Technik identifiziert Fälle, bevor die eigentliche Überprüfung beginnt, bei denen partial order reduction Regeln sicher angewandt werden können, wenn die Überprüfung selbst durchgeführt wird. Diese statische Reduzierungsmethode verhindert den Laufzeitoverhead mit dem partial order reduction Strategien in der Vergangenheit Probleme hatten.

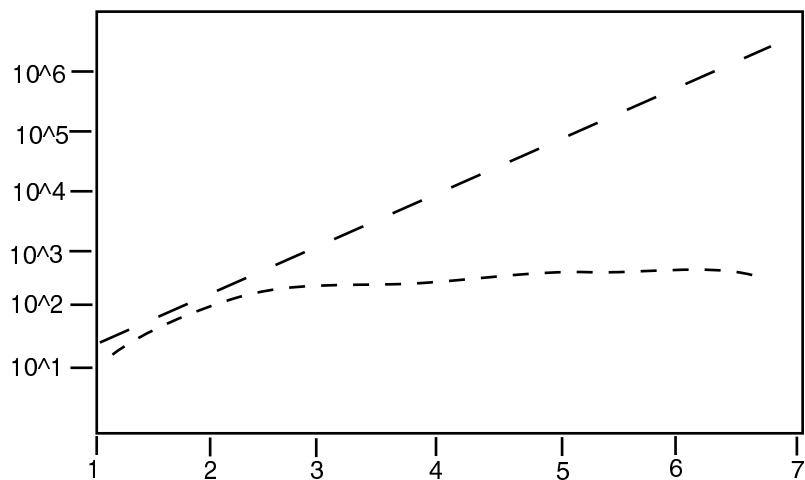


Abbildung 4: Auswirkung der Partial Order Reduction

Das Schaubild zeigt die Anzahl von erreichbaren Zuständen, die für eine vollständige Überprüfung eines bestimmten Modells nötig sind. Man sieht, dass das exponentielle Wachstum der Anzahl von Prozessen auf lineares Wachstum reduziert werden kann. Normalerweise ist die Reduktion der Zustandsgröße und der Speichergröße linear zur Größe des Modells. Damit wird soviel Speicher und Zeit wie möglich gespart.

SPIN hat zwei wichtige Vorteile gegenüber herkömmlichen Reduzierungsmethoden. Seine statische Reduzierungsmethode führt bei eingehender Suche nicht zu einem bemerkbaren Anstieg an Speicherverbrauch. Die Reihenfolge von Prozessen und Variablen machen ihr nichts aus.

Betrachtet man die Speicherverwaltung in SPIN so kann im schlimmsten Fall die Anzahl der Prozesse exponentiell mit der Größe der überlappenden Prozesse wachsen. Man kann anhand der Produktgröße (Anzahl der erreichbaren Systemzustände R) den Speicherraum und die Zeit festlegen, welche benötigt werden um bestimmte Überprüfungen durchzuführen. Um Sicherheitseigenschaften zu beweisen wachsen die Berechnungskosten von Zeit und Speicherraum linear zur Anzahl der Zustände, die von R aus erreichbar sind. Um Livenesseigenschaften zu beweisen bedarf es doppelt solanger Zeit, aber kaum bemerkbarer Zunahme an Speicher. Für LTL Eigenschaften verändert sich der Speicherbedarf kaum, dafür kann die Zeit um einen exponentiellen Faktor wachsen.

Meist haben LTL Eigenschaften nicht mehr als zwei oder drei Operatoren und sind somit verglichen mit der Komplexität des Systems selbst wenig komplex. Man kann einen Modellüberprüfer konstruieren, der wenig Speicher braucht, dafür steigt aber die Laufzeit in unerwünschtem Maße. Daher versucht man Techniken zu entwickeln, die Laufzeit und Speicherverbrauch optimieren.

5 Zusammenfassung

Zum Schluß möchte ich noch einmal die wichtigen Details von SPIN zusammenfassen. SPIN ist ein Softwarepaket zur Softwareverifizierung von verteilten Systemen. Zur Prüfung eines Systems werden Testprogramme in PROMELA spezifiziert. Eine Testspezifikation kann als LTL Formel formuliert werden, die als ein Büchautomat dargestellt werden kann. Mehrere Anfragen sind ein Produkt von Büchautomaten. SPIN erzeugt dann Simulationsläufe, die das System testen und speichert die Ergebnisse. Aus den Ergebnissen werden dann neue Simulationsläufe erzeugt. Da dies zeit- und speicheraufwendig ist, werden die Durchläufe mit Hilfe von Reduktionsalgorithmen optimiert. SPIN wird ständig erweitert, so gibt es z.B. XSPIN, das durch seine grafische Oberfläche benutzerfreundlicher ist.

SPIN gehört zu dem relativ jungen Forschungszweig der nebenläufigen Systeme. SPIN zeichnet dabei aus, dass es einen Unterschied zwischen Verhalten und Verhaltensanforderungen macht. Die Anforderungen und Verhalten werden auf ihre interne und gegenseitige Gleichheit überprüft. Das Design wird solange verbessert bis eine gewisse Korrektheit angenommen werden kann und erst dann wird das System vollständig implementiert.

SPIN gewinnt bei dem Designen von verteilten Systemen immer mehr an Beachtung, was nicht zuletzt darauf zurückzuführen ist, dass formale Methoden immer mehr an Bedeutung gewinnen.

6 Referenzen

[1] Gerard J.Holzmann, The Model Checker SPIN
IEEE TRANSACTION ON SOFTWARE ENGINEERING, VOL. 23, NO. 5,
MAY 1997

[2] On-the-fly, LTL Model Checking with SPIN
<http://netlib.bell-labs.com/netlib/spin/whatispin.html>

[3] SPIN Verifier's Roadmap: Using XSPIN
<http://cm.bell-labs.com/cm/cs/what/spin/Man/GettingStarted.html>

[4] Temporale Logik
<http://nusmv.irst.itc.it/NuSMV/papers/sttt.j/html/node3.html>

[5] PROMELA
<http://cm.bell-labs.com/cm/cs/what/spin/Man/Intro.html>