LoLA wird Pfadfinder

Karsten Schmidt
Institut für Informatik
Humboldt–Universität zu Berlin
10099 Berlin
e–mail: kschmidt@informatik.hu-berlin.de

6. August 1999

Zusammenfassung

LoLA ist ein erreichbarkeitsgraphbasiertes Werkzeug für Stellen/Transitions–Netze. Hier geht es um LoLAs Fähigkeit, erreichbare Zustände mit vorgegebenen Eigenschaften zu suchen und im Erfolgsfall einen solchen Zustand mit einem Weg von der Anfangsmarkierung dorthin anzugeben. Die Besonderheit besteht darin, daß während der Suche nur zwei Zustände gespeichert werden: der Anfangszustand und der aktuelle Zustand. Anstatt also den Zustandsraum systematisch zu durchmustern (und dabei meist am Speicherende zu scheitern), navigiert LoLA ohne Kenntnis der schon betretenen Zustände durch den Erreichbarkeitsgraph. Daß sie trotzdem ihre Aufgabe erfolgreich löst, liegt an ihren spezifischen Fähigkeiten: Schnelligkeit, Bescheidenheit, Zielstrebigkeit, Neugier und Ausdauer.

1 Wer ist LoLA?

LoLA enstand mit dem Ziel der Validierung von Reduktionstechniken für Erreichbarkeitsanalyse. Sie kann Erreichbarkeits- und Überdeckbarkeitsgraphen (letztere wie bei KARP und MILLER [5], d.h. ohne FINKELs Verbesserungen [3]) bauen, und zwar unter Verwendung symmetrischer und/oder sturer Reduktion. Für symmetrisch reduzierte Graphen [4, 2, 9] wird ein polynomiell großes Erzeugendensystem der Netzsymmetriegruppe berechnet und zur Graphreduktion verwendet, indem zu jeder Äquivalenzklasse von Zuständen bezüglich der Symmetrierelation jeweils nur ein Vertreter gespeichert wird (siehe auch [6]). Für sture Reduktion [10] wird in jedem Zustand eine Teilmenge von Transitionen berechnet und nur die schaltbaren Transitionen aus dieser Menge weiterverfolgt. Die Menge ist so beschaffen, daß die zu analysierende Eigenschaft bewahrt wird. Gegenwärtig werden als Eigenschaften Erreichbarkeit einer Markierung, Beschränktheit von Stellen bzw. des Netzes, tote Transitionen, Reversibilität, Homestates sowie Formeln einer branching–time–temporal–logic mit „möglich“ (CTL:EF) und „immer“ (CTL: AG) als Temporaloperatoren unterstützt (siehe [8, 7]).

Neben diesen Leistungen kann LoLA nun auch als Pfadfinder arbeiten.

2 Was ist ein Pfadfinder?

Ein Pfadfinder bekommt eine Zustandseigenschaft und hat die Aufgabe, unter den erreichbaren Zuständen einen mit dieser Eigenschaft zu finden und den Weg vom Anfangszustand dorthin zu zeigen. Wir nennen Zustände mit der gesuchten Eigenschaft *Zielzustände*.

LoLA kann mit Eigenschaften umgehen, die boolesche Kombinationen von *Elementaraussagen* sind. Eine Elementaraussage hat die Form $s\ R\ k$, wobei s eine Stelle des Netzes, R ein Relationssymbol aus dem Vorrat $\{=, \neq, <, >, \leq, \geq\}$ und k eine natürliche Zahl sind. Ein Zustand m erfüllt eine Elementaraussage $s\ R\ k$, falls die Aussage $m(s)\ R\ k$ mit den üblichen Interpretationen der Relationszeichen zu einer wahren Aussage wird. Damit ist unmittelbar klar, wann ein Zustand eine zusammengesetzte Zustandsaussage erfüllt.

Weil im Vorrat an Relationssymbolen zu jedem Symbol auch das negierte Symbol auftaucht, kann man unter Verwendung von DE MORGANs Formeln Negationszeichen in Zustandsaussagen komplett eliminieren. Wir werden daher nur $\wedge$ und $\vee$ als logische Operatoren betrachten.

3 LoLA ist bescheiden

Diese Aussage bezieht sich auf LoLAs Umgang mit Speicherplatz. Während ihrer Pfadfindertätigkeit speichert LoLA lediglich

- das Netz mit Stellen, Transitionen, Bögen, Vor-, Nachbereichen und einigen anderen Informationen in der Größenordnung des Netzes, die dem schnellen Zugriff auf häufig verwendete Informationen gestatten;
- den Anfangs- und den aktuellen Zustand;
- die Zustandsaussage in Form eines Baumes; dabei werden Hintereinanderausführungen von $\wedge$ und $\vee$ zu je einem Knoten zusammengefaßt;
- den Pfad bis zu einer Länge eines nutzerdefinierten Wertes `MAXPATH`;
- eine Hashtabelle, die zu jedem Hashwert eine natürliche Zahl speichert.

Bei einem Netz von 90.000 Knoten, MAXPATH = 3.000.000 und einer Hashtabellengröße von 655.360 passen alle Datenstrukturen bequem in einen 64MB Hauptspeicher.

4 LoLA ist schnell

Alle während einer Pfadsuche von LoLA auszuführenden Tätigkeiten erfordern Zugriffe nur in der näheren Umgebung (im Petrinetzsinn) einer Transition. Im einzelnen sind zu einer aktuellen Markierung folgende Schritte auszuführen:
Formelwert bestimmen. Von einer Stelle s aus sind alle Elementaraussagen zugreifbar, die sich auf s beziehen und werden nur dann, wenn s in der Umgebung der zuletzt geschalteten Transition liegt, überprüft. Nur, wenn sich der Wert eines Knotens ändert, wird der darüberliegende Knoten neu überprüft. Die Knoten für $\wedge$ und $\vee$ führen Buch über die Anzahl der erfüllten Teilaussagen. Diese Anzahl reicht aus, um deren Wahrheitswert zu determinieren.
Menge der schaltbaren Transitionen bestimmen. Sei t die zuletzt geschaltete Transition. Eine Transition t' wird nur in zwei Fällen bei der aktuellen Markierung auf Konzession geprüft. Der erste Fall liegt vor, wenn t' vor dem Schalten von t Konzession hatte und t Marken aus dem Vorbereich von t' entfernt. Der zweite Fall liegt vor, wenn t' vor dem Schalten von t keine Konzession hatte und t Marken im Vorbereich von t' erzeugt. Bei allen anderen Transitionen kann sich Konzessioniertheit gegenüber der alten Markierung nicht geändert haben.
Eine zu schaltende Transition auswählen. Siehe hierzu die Abschnitte über Zielstrebigkeit und Neugier.

Die ausgewählte Transition schalten. Lediglich die Stellen in der Umgebung der geschalteten Transition werden beeinflußt.
Der neuen Markierung einen Hashwert zuordnen. LoLA verwendet eine gewichtete Markensumme (modulo der Tabellengröße) als Hashfunktion. Damit ist die Wertänderung durch das Schalten einer Transition konstant und wird bei der Transition gespeichert.

Wegen der lokalen Ausführbarkeit aller Operationen bleibt die Geschwindigkeit von LoLA (gemessen in geschalteten Transitionen pro Sekunde, $\frac{TR}{s}$) auch bei wachsender Netzgröße nahezu gleich.

Der Verzicht auf die Speicherung und Verwaltung besuchter Markierungen beschleunigt LoLAs Arbeit enorm (neben den netten Auswirkungen auf den Speicherplatz). Die Markierungsverwaltung ist die einzige Tätigkeit in anderen LoLA–Komponenten, die nicht inkrementell implementiert ist. So kann man den Zeitbedarf für Markierungsverwaltung etwa an folgenden Beispielen ablesen: Für ein Netz mit 25 Stellen schafft LoLA mit Markierungsverwaltung etwa 25.000 $\frac{TR}{s}$, für ein Netz mit 5.000 Stellen dagegen nur 400 $\frac{TR}{s}$. Ohne Markierungsverwaltung, d.h. beim Pfadfinden, erreicht LoLA bei beiden Netzen um die 500.000 $\frac{TR}{s}$. Alle Zahlen beziehen sich auf LINUX, Pentium II, 400 MHz.

5 LoLA ist zielstrebig

Da LoLA keine Buchführung über die schon besuchten Zustände betreibt, muß sie das systematische Durchforsten des Zustandsraumes durch eine Intuition darüber ersetzen, wo sich Zielzustände befinden könnten, und dann einfach in die entsprechende Richtung losmarschieren. Diese Intuition bezieht LoLA durch die Auswertung sturer Mengen [10] in ihrer Version für Erfüllbarkeit von Zustandsaussagen [8]. Bei einer Markierung m ist eine sture Menge eine Menge U von Transitionen mit der Eigenschaft, daß unter den kürzesten Wegen zu einem Zielzustand garantiert einer ist, der mit einer Transition aus U beginnt. Somit besteht der erste Teil von LoLAs Intuition darin, eine solche Menge U zu berechnen (das geht sehr effizient) und Transitionen außerhalb von U aus jeglicher weiteren Betrachtung auszuschließen.

Darüberhinaus wertet LoLA aber die genauere Struktur der sturen Menge U aus. Bei der Berechnung einer sturen Menge gehen wir davon aus, daß die aktuelle Markierung m selbst kein Zielzustand ist (sonst wäre LoLA längst mit der Ergebnispräsentation beschäftigt).

Eine sture Menge wird inkremetell berechnet. Man startet mit einer Attraktormenge U_0, die die Eigenschaft hat, daß jeder Weg zu einem Zielzustand garantiert Transitionen aus U_0 enthält. Eine solche Menge kann man in der Tat aus aktueller Markierung und Zustandseigenschaft bestimmen, wie die folgende Tabelle zeigt (beachte, daß ein rekursiver Abstieg nur in nicht erfüllte Teilformeln vorkommt, also nach wie vor davon ausgegangen werden kann, daß der aktuelle Zustand die (Teil)–formeln *nicht* erfüllt).

Konstrukt ϕ	Attraktormenge U_ϕ
$s < k$	$s^\bullet$
$s \leq k$	$s^\bullet$
$s = k$	$s^\bullet$, falls $m(s) > k$,
	${}^\bullet s$, falls $m(s) < k$
$s > k$	${}^\bullet s$
$s \geq k$	${}^\bullet s$
$s \neq k$	${}^\bullet s \cup s^\bullet$
$\phi_1 \wedge \phi_2$	U_{ϕ_1}, falls ϕ_1 nicht erfüllt ist,
	U_{ϕ_2}, falls ϕ_2 nicht erfüllt ist
$\phi_1 \vee \phi_2$	$U_{\phi_1} \cup U_{\phi_2}$

Die Attraktormenge wird anschließend iterativ zu einer schwach sturen Menge vergrößert, wobei in jedem Iterationsschritt zu jeder konzessionierten Transition alle dazu im Konflikt stehenden Transitionen und zu jeder nicht konzessionierten Transition die Vortransitionen *einer* nicht ausreichend markierten Stelle hinzugenommen werden.

Man kann nun Transitionen in der sturen Menge danach klassifizieren, in welchem Iterationsschritt sie in die sture Menge aufgenommen wurden (Attraktormenge = Aufnahme in Schritt 0).

Dann kann man erstens beobachten, daß durch Schalten von Transitionen außerhalb der Attraktormenge ein Nichtzielzustand garantiert in einen Nichtzielzustand überführt wird. Zweitens kann sich an der Schaltfähigkeit einer im Iterationsschritt i aufgenommenen Transition nur durch Schalten einer spätestens im Schritt $i+1$ aufgenommenen Transition etwas ändern. Um also eine Änderung des Gesamtformelwertes möglichst schnell zu provozieren (und worum sonst geht es?), liegt es nahe, früh in die sture Menge aufgenommene Transitionen gegenüber spät aufgenommenen Transitionen zu bevorzugen.

Diese Greedy-Heuristik ist der Kern von LoLAs Intuition über die Richtung von Zielzuständen. Sie wird implementiert, indem Transitionen in der Reihenfolge, in der sie berechnet werden, als Kandidaten zum Schalten vorgeschlagen werden. Zu jedem Kandidaten wird (siehe Neugier) eine Münze geworfen, die darüber entscheidet, ob der Kandidat sich als zu schaltende Transition qualifiziert. Ist eine Transition selektiert, kann die Berechnung der sturen Menge unterbrochen werden, so daß auch hier wegen der meist hohen Priorität früher Transitionen normalerweise nur lokaler Berechnungsaufwand besteht.

6 LoLA ist neugierig

Bevor sich ein Kandidat als Schalttransition qualifiziert, überprüft LoLA, wie interessant eine Fortsetzung mit dem Kandidaten sein könnte. Hier kommt die Hashtabelle ins Spiel. Sie speichert zu jedem Hashwert, wie oft eine Markierung mit diesem Hashwert schon aktuelle Markierung war. Führt die Kandidatentransition zu einem noch nie besuchten Hashwert, so ist der Folgezustand garantiert neu und LoLAs Neugier läßt sie den Kandidaten mit Wahrscheinlichkeit 1 selektieren. Je größer die Zahl der vorangegangenen Besuche bei einem Hashwert ist, desto eher liegt es nahe, daß der Folgezustand schon einmal besucht wurde, und LoLAs Interesse (d.h. Auswahlwahrscheinlichkeit) sinkt. Wenn h der Hashwert der durch das Schalten des Kandidaten t entstehenden Markierung ist, so selektiert LoLA t mit Wahrscheinlichkeit $\frac{K}{K+h}$. Dabei ist K eine fest implementierte Konstante, die steuert, wie stark LoLAs Interesse durch steigende Besuchszahlen absinken soll. Experimente für eine glückliche Wahl von K stehen noch aus.

Ist die Liste der Kandidaten erschöpft, ohne daß LoLA einen davon selektiert hat, wird aus der Menge aller abgelehnten Kandidaten eine Transition (gleichverteilt) gewählt.

Mit Zielstrebigkeit und Neugier überlagern sich zwei Heuristiken, deren Dominanz sich im Laufe der Zeit verschiebt. Zu Anfang, d.h. bei kleinen Besuchszahlen, bestimmt

hauptsächlich die Zielstrebigkeit über die Transitionsauswahl, während später die Neugier die Oberhand bekommt. Sind schließlich die Besuchszahlen so groß, daß alle Kandidaten abgelehnt werden, dominiert die ungewichtete zufällige Auswahl, d.h. es findet nach dem Versagen der Heuristiken letztendlich blinde Suche im (immer noch stur reduzierten) Erreichbarkeitsgraphen statt.

7 LoLA hat Ausdauer

In vielen Fällen führen Zielstrebigkeit und Neugier dazu, daß LoLA ihr Ziel auf Anhieb erreicht. Garantieren kann das natürlich niemand. Deshalb wird beim Erreichen einer nutzerdefinierten Maximalpfadlänge die Suche abgebrochen und eine neue Suche gestartet (also zur Anfangsmarkierung zurückgesetzt). Als Lehre aus dem gescheiterten Versuch werden die in der Hashtabelle vermerkten Besuchszahlen beibehalten, so daß aufgrund geänderter Wahrscheinlichkeiten (und wegen der Zufallsnatur) neue Wege beschritten werden. Erreicht LoLA einen Deadlock, wird ebenfalls neu gestartet.

Wie dem auch sei, LoLA wird niemals aufgeben. Wird sie auf eine Eigenschaft angesetzt, die gar nicht durch erreichbare Markierungen erfüllt wird, wird sie diese Tatsache nie und nimmer zur Kenntnis nehmen, sondern immer wieder von vorn neue Pfadfindeversuche starten, bis daß der Anwender sie dabei unterbricht. Es gibt natürlich auch erreichbare Zielzustände, bei denen LoLAs Intuition sie in die Irre führt und sie erst nach astronomischer Versuchszahl finden läßt. Daher lautet also LoLAs Antwortspektrum „Ja, der Zustand ist definitiv erreichbar und ich kann sogar sagen, wie“ bzw. „Hm, ich kann nicht sagen, ob der Zustand erreichbar ist“. Dies ist genau das duale Antwortspektrum zu Methoden, die auf der Auswertung der Petrinetz–Zustandsgleichung mit Methoden der linearen Programnmierung beruhen (dort: nein/vielleicht).

8 Erste Erfahrungen

Das bisher größte getestete System hatte 90.000 Knoten (50.000 Stellen, 40.000 Transitionen). Die Eigenschaft war eine Konjunktion aus 9.999 Elementaraussagen. Das System hat $3^{10.000} - 1$ erreichbare Zustände. Genau einer davon erfüllt die Eigenschaft. Der kürzeste Pfad zu diesem Zustand hat die Länge 29.997.

LoLA meisterte die gestellte Aufgabe (mit 400MHz auf 64MB unter LINUX) in 80 Sekunden. Dabei entfielen 70 Sekunden auf das Einlesen von Netz und Eigenschaft, nur wenig mehr als 0 Sekunden auf die eigentliche Pfadsuche und 10 Sekunden auf das Schreiben des etwa 50.000 Transitionen langen Pfades.

Ein Suchverfahren mit klassischer Markierungsverwaltung müßte vermutlich selbst bei perfekter Reduktion wenigstens die etwa 30.000 Markierungen speichern, die auf einem kürzesten Weg zur Zielmarkierung liegen. Das sind bei 1 Bit pro Stelle und Markierung (das Netz ist sicher) also mindestens $30000 \cdot 50000$ Bit oder 187,5 MB. Dieses Speichervolumen ist mit den oben genannten Hardwarevoraussetzungen nicht sinnvoll zu beherrschen.

Durch ihre Bescheidenheit kann LoLA also Probleme angehen, die außerhalb der Reichweite von systematischer Suche liegen. Der Preis dafür ist das reduzierte Antwortspektrum. Den Nachteil der Unkenntnis schon besuchter Zustände versucht LoLA durch ihre Schnelligkeit und ihre Intuition (Zielstrebigkeit/Neugier) zu kompensieren.

BDD-basierte Methoden ([1]) können im Prinzip große Zustandsräume beherrschen. Derzeitige Methoden dürften aber mit einer Stellenzahl von 50.000 und einer Suchtiefe von 30.000 auch noch ihre Schwierigkeiten haben. Daneben erfordert die Pfadbestimmung in BDD-basierten (symbolischen) Verfahren einen gewissen Rechenaufwand, während in LoLA ein Pfad auf kanonische Weise mitgeliefert wird.

Der von LoLA gefundene Pfad ist in der Regel nicht der kürzestmögliche zu dem gefundenen Zustand. Manchmal enthält er sogar Kreise. Dennoch ist der Pfad oft deutlich kürzer als Pfade, die durch tiefensuchbasierte Verfahren geliefert werden. Kürzere Pfade verbessern die Nutzbarkeit für Simulationen oder suboptimale Schedulings.

Um LoLAs Unfähigkeit zu negativen Antworten zu kompensieren, könnte LoLA wettbewerbsparallel mit anderen Werkzeugen (Pfadfindern und Nichterreichbarkeitsfeststellern) betrieben werden.

Literatur

[1] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Fifth Annual IEEE Symposium on Logic in Computer Science*, 1990.

[2] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. On well–formed colored nets and their symbolic reachability graph. *Proc. of the 11th Int. Conf. on Application and Theory of Petri Nets*, 1990.

[3] A. Finkel. A minimal coverability graph for Petri nets. *Proc. of the 11th International Conference on Application and Theory of Petri nets*, pages 1–21, 1990.

[4] Huber, A. Jensen, Jepsen, and K. Jensen. Towards reachability trees for high–level Petri nets. In *Advances in Petri Nets 1984, LNCS 188*, pages 215–233, 1984.

[5] R. M. Karp and R. E. Miller. Parallel programm schemata. *Journ. Computer and System Sciences 4*, pages 147–195, 1969.

[6] K. Schmidt. How to calculate symmetries of Petri nets. Accepted for *Acta Informatica*.

[7] K. Schmidt. Stubborn sets for modelchecking the AG/EF fragment of CTL. Informatik-Bericht 123, Humboldt–Universität zu Berlin, 1999.

[8] K. Schmidt. Stubborn sets for standard properties. *20th Int. Conf. Application and Theory of Petri Nets, LNCS 1639*, pages 46–65, 1999.

[9] P. Starke. Reachability analysis of Petri nets using symmetries. *J. Syst. Anal. Model. Simul.*, 8:294–303, 1991.

[10] A. Valmari. Error detection bu reduced reachability graph generation. *Proc. of the 9th European Workshop on Application and Theory of Petri Nets, Venice*, 1988.