Perlen der Weisheit

Model Checking

Alexander Wißpeintner

18. Dezember 2001

Übersicht

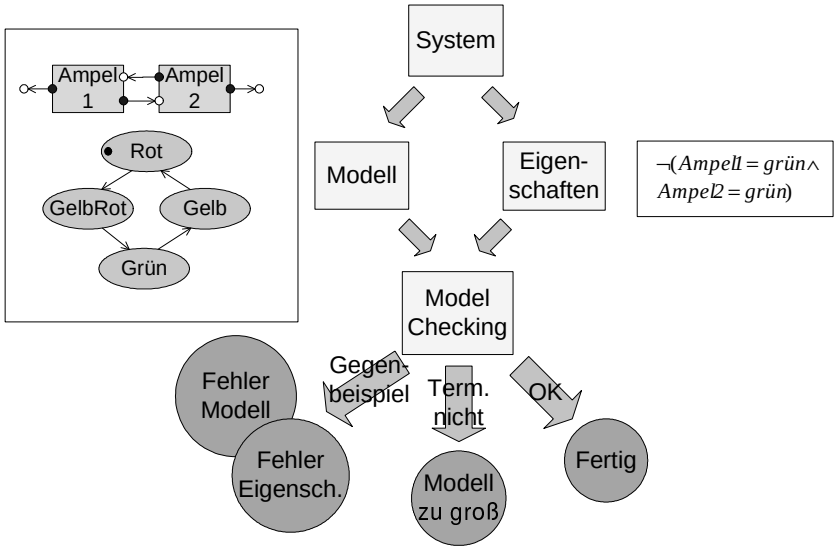
1. Einleitung
2. Temporallogik
3. Explicit State Model Checking
4. Symbolisches Model Checking
5. Zusammenfassung

Model Checking

- Technik zur automatischen Verifikation von Systemeigenschaften
- Nur auf endliche Zustandsräume anwendbar

- Erfolgreiche Anwendung in den Bereichen:
  - Hardware Verifikation
  - Verifikation von Protokollen

Model Checking Prozess



Automatentheroretische Lösung

- Problem: Modell Automat K und Eigenschaft φ gegeben, entscheide ob $K \models \varphi$
- Betrachte K als ω-Automat, wobei alle Zustände akzeptierende Zustände sind
- $L(K)$ ist die Menge der Ausführungssequenzen von K
- $K \models \varphi$
  falls $L(K) \subseteq L(\varphi)$
  falls $L(K) \cap L(\neg\varphi) = \emptyset$
  falls $L(K \times B_{\neg\varphi}) = \emptyset$
- Komplexität $O(|K| \cdot |B_{\neg\varphi}|) = O(|K| \cdot 2^{|\varphi|})$

Model Checking Algorithmen

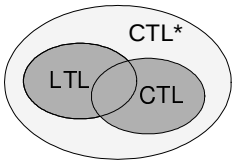
- Explicit State Model Checking
  - Spin von Gerard Holzmann
  - Zustände werden nacheinander explizit betrachtet und für jeden Zustand wird überprüft, ob alle Eigenschaften erfüllt sind
- Symbolisches Model Checking
  - SMV von Ken McMillan
  - Zustandsübergangsgraph wird als boolsche Formel in einem BDD gespeichert
  - Durch symbolische Berechnungen wird gezeigt, ob die Eigenschaft erfüllt ist

Problematisch

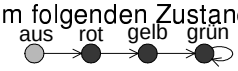
- Systeme besitzen in der Regel eine sehr große Anzahl von Zuständen
- State Explosion Problem: Werden Zustandsautomaten parallel komponiert, dann wächst die Zahl der Systemzustände exponentiell mit der Zahl der Automatenzustände
- Model Checking für große System nicht anwendbar:
  - Laufzeit zu lange
  - Speicherplatz reicht nicht aus

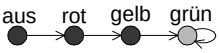
Temporallogik

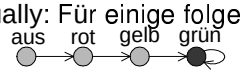
- Keine Logik über Echtzeit
- Kausale Abhängigkeiten über Ereignisse
- Erweiterung der Aussagenlogik
- Unterschiedliche Ausdrucksmächtigkeit verschiedener Varianten

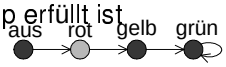


Linear Temporal Logic (LTL)

- Eigenschaften über Ausführungssequenzen des Systems
- A ist die Menge der atomaren Aussagen über das System
- $\forall p \in A$, p ist eine Formel q,r Formeln
- $\neg q$, $q \vee r$, $q \wedge r$, $q \rightarrow r$, $q \Leftrightarrow r$
- $\bigcirc q$, $X\, q$, Next: Im folgenden Zustand soll q gelten
  $\bigcirc rot$ 

- $\Box q$, $G\, q$, Always: Im aktuellen und für alle folgenden Zustände soll q gelten
  $\Box grün$ 

- $\Diamond q$, $F\, q$, Eventually: Für einige folgende Zustände soll q gelten
  $\Diamond gelb$ 

- $q\ U\ p$, Strong Until : Im aktuellen und für alle folgenden Zustände soll q gelten, bis p erfüllt ist
  $rot\ U\ gelb$ 

Beispiele LTL

- Invariante:
  $\Box\neg(Ampel1\_grün \wedge Ampel2\_grün)$
- Reaktion
  $\Box(rot \rightarrow \Diamond grün)$
- Fairness:
  $\Box\Diamond rot \rightarrow \Box\Diamond grün$
- Fortschritt:
  $\Box(rot \rightarrow rot\ \mathrm{U}\ gelbrot\ \mathrm{U}\ grün\ \mathrm{U}\ gelb\ \mathrm{U}\ rot)$
- Stabilität:
  $\Diamond\Box blinktgelb$

2. Temporallogik

Computation Tree Logic (CTL)

- Braching Time Logic: Quantifizierung über Ausführungssequenzen
- A für alle Ausführungspfade
- E für einige Ausführungspfade
- AX, EX Next
- AF, EF Eventually
- AG, EG Always
- AU, EU Until
- Beispiel Reaktion AG($rot \rightarrow$ AF $grün$)
- LTL Formel $\Diamond\Box p$ nicht ausdückbar in CTL
- CTL Formel AG(EF p) nicht ausdrückbar in LTL

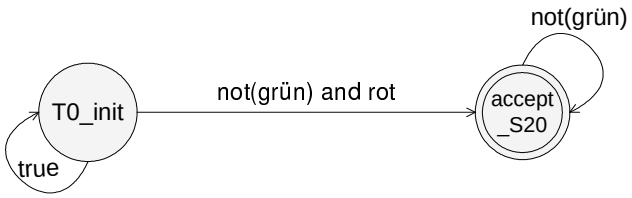
3. Explicit State Model Checking

Explicit State Model Checking mit Spin

- Anwendungsgebiet Protokollverifikation
- Systemmodell in Promela / asynchrone Komm.
- Eigenschaften in Linear Temporal Logic (LTL)
- Idee
  - Umwandlung der negierten Eigenschaft in einen Büchi Automaten
    - Endlicher Automat der unendliche Eingaben akzeptiert, falls der akzeptierende Zustand unendlich oft durchlaufen wird
  - Nach und nach Berechnung des Zustandsübergangsgraphen des Systemmodells $\times$ Eigenschaftsautomat und Suche von akzeptierenden Zyklen => Eigenschaft nicht erfüllt

Umwandlung LTL in Büchi Automat

$$\neg\ \Box(rot \rightarrow \Diamond grün)$$

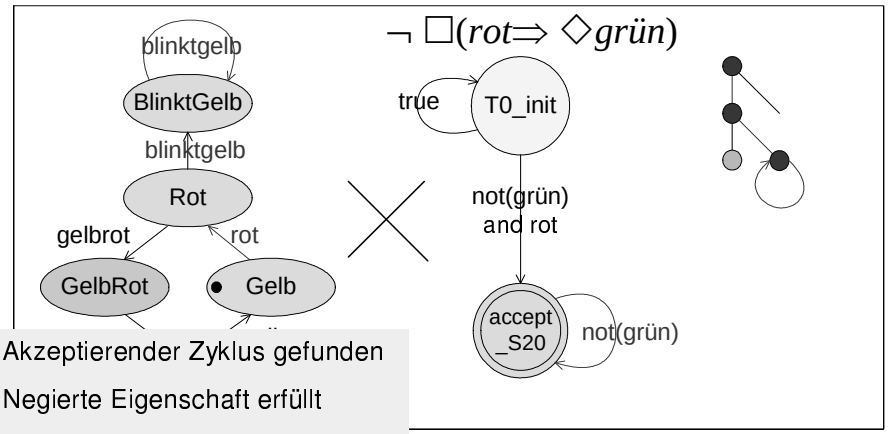


- Automat akzeptiert Ausführungssequenz, falls der akzeptierende Zustand unendlich oft durchlaufen wird

On-the-fly Model Checking

- Zustandsübergangsgraph des Systems wird während der Überprüfung der Eigenschaft nach und nach aufgebaut
- Vorteil: Falls das System die Eigenschaft nicht erfüllt, kann dies erkannt werden bevor das gesamte System betrachtet wurde

Beispiel



Akzeptierender Zyklus gefunden

Negierte Eigenschaft erfüllt

Eigenschaft nicht erfüllt

Gegenbeispiel auf Suchstack

Partial Order Reduction

- Problem: Asynchrones Kommunikationsmodell
  ⇒ Viele mögliche Systemabläufe
  ⇒ Zustandsraum explodiert
- Beobachtung: Gültigkeit der Eigenschaft oft unabhängig von der Reihenfolge der Ausführung nebenläufiger unabhängiger Ereignisse.
- Reduzierter Zustandsraum mit Klassen von Ausführungssequenzen die für die zu prüfende Eigenschaft äquivalent sind.
- Auswertung vor Beginn der Suche
- Effektivität von Partial Order Reduction stark von Systemstruktur und Eigenschaft abhängig

3. Explicit State Model Checking

Bit State Hashing

- Problem: Bereits betrachtete Systemzustände müssen erkannt werden ⇒ Zustände speichern ⇒ Out of Memory
- Lösung: Nicht den Zustand speichern, sondern nur im Speicher markieren.
- Zwei statistisch unabhängige Hash Funktionen bilden Zustände auf jeweils zwei Bits im Speicher ab.
- Sind beide Bits markiert, dann wird angenommen, der Zustand wurde bereits besucht.
- Es kann nicht garantiert werden, dass zwei unterschiedliche Zustände nicht auf die selben Speicherstellen abgebildet werden
- ⇒ Keine 100% Abdeckung der Zustände garantiert

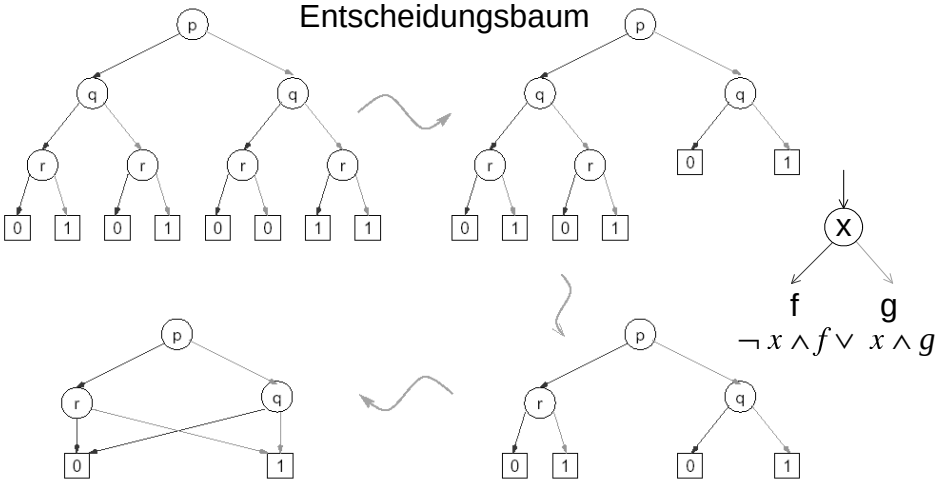
4. Symbolisches Model Checking

Symbolisches Model Checking mit SMV

- Anwendungsgebiet Hardwareverifikation
  - Synchrones Modell
- Darstellung des Zustandsübergangsgraphen als OBDD
- Eigenschaft in CTL
- Ermittlung der Menge von eigenschaftserfüllenden Zuständen durch Fixpunktberechnung über logische Operationen auf dem BDD

Binary Decission Diagrams (BDD)

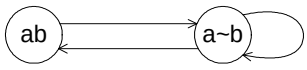
Aussagenlogische Funktion $(\neg p \wedge r) \vee (p \wedge q)$ als Entscheidungsbaum



Ordered BDDs

- Variablen treten auf allen Pfaden in der selben Reihenfolge auf
- Lücken sind erlaubt
- Logische Operationen lassen sich effizient auf OBDDs durchführen
- Kripke Struktur M=(S,R,L)
- Kodierung von Zuständen als Binärzahl
- Transitionen durch Konjunktion von Start- und Folgezustand

a', b' Folgezustand

- 

$(a \wedge b \wedge a' \wedge \neg b') \vee$
$(a \wedge \neg b \wedge a' \wedge b') \vee$
$(a \wedge \neg b \wedge a' \wedge \neg b')$

- OBDD repräsentiert Mengen von Zuständen und Transitionen

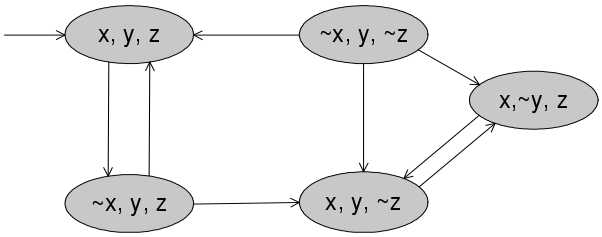
CTL symbolisches Model Checking

- CTL Operationen reduzierbar auf EX, EU und EG
- Erfüllungsmenge $||\text{EX } \varphi|| = R^{-1}(||\varphi||) =$
  $\{s : \exists\, t : (s,t) \in R \text{ und } t \in ||\varphi||\}$

- Rekursiv EG $\varphi \Leftrightarrow \varphi \wedge \text{EX EG } \varphi$
- Erfüllungsmenge $||\text{EG } \varphi|| = ||\varphi|| \cap R^{-1}(||\text{EG } \varphi||)$
- $||\text{EG } \varphi||$ ist größter Fixpunkt von $f(M) = ||\varphi|| \cap R^{-1}(M)$
- Größter Fixpunkt $\nu f = \bigcap_{i \geq 0} f^i(S)$

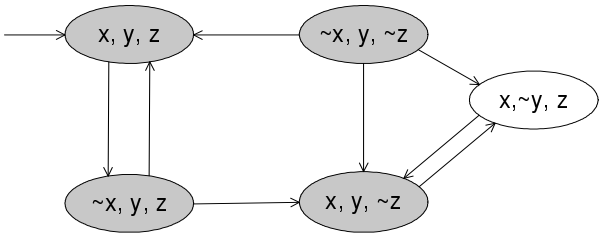
- EU ebenfalls durch Fixpunktberechnung

Beispiel



Berechne $||\text{EG } y||$
$f^0(S)=S$

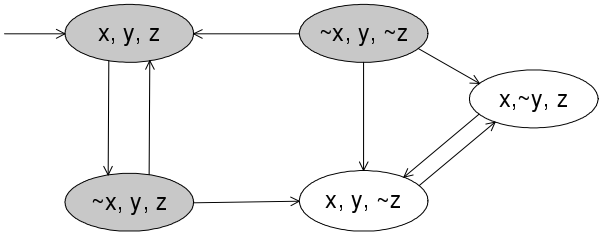
1. Iteration



Berechne ||EG y||

$f^1(S) = ||y|| \cap R^{-1}(S)$

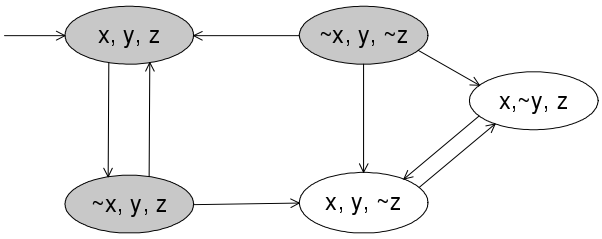
2. Iteration



Berechne ||EG y||

$f^2(S) = ||y|| \cap R^{-1}(f^1(S))$

3. Iteration



Berechne ||EG y||

$f^3(S) = ||y|| \cap R^{-1}(f^2(S)) = f^2(S)$

Weiterführende Techniken

- Kompositionales Verifizieren
  - Eigenschaften auf Komponenten zeigen
  - Über Komponenteneigenschaften auf Systemeigenschaften schließen
- Abstraktion
  - Datenabstraktion, Modellabstraktion
- Kombinierte Ansätze deduktive Beweissysteme und Model Checking
- Paralleles Model Checking
- Jede Menge Tricks um etwas größere Systeme verifizieren zu können: „20% weniger Speicherbedarf dafür 30% längere Laufzeit“

Literatur

- E. Clarke, O. Gumberg, D. Peled, *Model Checking*, MIT Press, 1999.
- G. Holzmann, *Design and Validation of Computer Protocolls*, Pentice Hall, 1990.
  http://cm.bell-labs.com/cm/cs/what/spin/Doc/Book91.html
- G. Holzmann, *The Model Checker Spin*, IEEE Transactions on Software Engineering, Vol. 23, No. 5, 1997.
  http://cm.bell-labs.com/cm/cs/who/gerard/gz/ieee97.ps.gz
- K. L. McMillan, *Symbolic Model Checking: an approach to the state explosion problem*, CMU Tech Rpt. CMU-CS-92-131, 1992.
  http://www-cad.eecs.berkeley.edu/~kenmcmil/thesis.ps

Zusammenfassung

- Model Checking toll für kleine Systeme
- Lässt sich zum „Debugging“ einsetzen – Gegenbeispiele
- Probleme:
  - Festgelegte Eigenschaften müssen korrekt sein
  - Überprüfbare Systemgröße stark beschränkt: Speicherbedarf / Laufzeit
- Unterschiedliche Model Checking Algorithmen / Tools für verschiedene Anwendungsgebiete
- Model Checking wird in der Industrie eingesetzt