

Automatisches Beweisen—Vertiefung

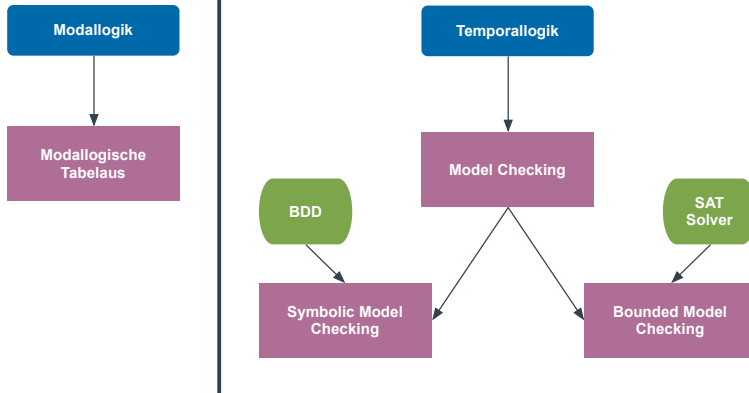
Bounded Model Checking

Christoph Zengler

Arbeitsbereich Symbolisches Rechnen
Prof. Dr. Wolfgang Küchlin
Universität Tübingen

22. Januar 2013

Der Plan für die nächsten drei Wochen



Symbolic Model Checking für CTL

- Kam Anfang der 90er auf
- Idee: Repräsentiere die Kripke Struktur als Boolesche Formel in einem BDD
- Durchbruch für das Model Checking, sehr viel größere Systeme konnten verifiziert werden ($> 10^{20}$ Zustände)

Bounded Model Checking für LTL

- Kam Ende der 90er auf
- Idee: Benutze einen SAT Solver zum Schlussfolgern über den Booleschen Formeln
- Wickle Modelle nur endlich oft ab

Motivation:

- Erfolg von SAT Solvern auf Model Checking übertragen
- Anleihen an Planungsproblemen

Wir wissen:

- LTL Model Checking arbeitet auf Pfaden
- Implizite Allquantifizierung über alle Pfade
- Finde einen Pfad, der die Bedingung verletzt

Neue Idee:

- Untersuche nur Pfade der Länge n
- Sukzessive Erhöhung der Länge (**Bound**)
- vgl. Abstraction Refinement Loop

Definition (Pfad)

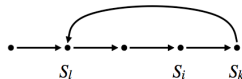
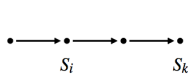
Ein Pfad in einem Modell $\mathcal{M} = (S, \longrightarrow, L)$ ist eine unendliche Folge von Zuständen s_0, s_1, s_2, \dots in S , so dass für jedes $i \geq 0$ $s_i \longrightarrow s_{i+1}$.

- π^i : Suffix des Pfades, beginnend bei dem i -ten Zustand

Unendliche Pfade

- Auch bei endlichem Präfix kann ein Pfad unendlich lang sein
- ⇒ Existenz von Rücksprüngen ([Back Loops](#))
- Existieren keine Rücksprünge, so kann über das Verhalten des Pfades nach dem Präfix nichts gesagt werden





Definition (Schleifen)

Für $l \leq k$ ist ein Pfad $\pi = s_0, s_1, \dots$ eine **(k, l)-Schleife**, wenn

- $s_k \rightarrow s_l$
- $\pi = s_0, s_1, \dots, s_{l-1}, (s_l, \dots, s_k)^*$

π ist eine **k-Schleife**, wenn es ein $k \geq l \geq 0$ gibt, für dass π eine **(k, l)-Schleife** ist

Idee: Bounded Semantics

- Wir betrachten nur endliche Präfixe von Pfaden
- d.h. nur die ersten $k + 1$ Zustände (s_0, \dots, s_k)
- Werte LTL Formeln entlang eines solchen begrenzten Pfad aus
- Ist der Pfad eine k-Schleife, so ändert sich nichts an der Semantik

Definition (Bounded Semantics mit Schleife)

Sei

- \mathcal{M} eine Kripke-Struktur, $k \geq 0$ und
- π eine k -Schleife.

Dann ist eine LTL Formel φ gültig entlang des Pfades π mit Grenze k ($\pi \models_k \varphi$), wenn $\text{holds}(\mathcal{M}, \pi, \varphi) = \text{true}$.

Problem!

In der üblichen (unbounded) Semantik von LTL Model Checking gilt:

- $\mathbf{F} p$ ist auf einem Pfad gültig, wenn wir ein Suffix finden, dass mit p beginnt

Aber

- In der Bounded Semantics von LTL Model Checking hat der Zustand s_k keinen Nachfolger mehr

Bounded Semantics ohne Schleifen

Algorithmus: $\text{holdsBound}(\mathcal{M}, \pi, k, i, \psi)$

Eingabe: Modell $\mathcal{M} = (S, \longrightarrow, L)$, Pfad $\pi = s_0 \longrightarrow \dots$, der keine k -Schleife ist, Bound k , aktuelle Position im Pfad i , LTL Formel ψ

Ausgabe: Evaluation von ψ auf dem Pfad π mit k und i

$\text{holdsBound}(\mathcal{M}, \pi, k, i, \psi) = \psi \text{ match}$

$\top \rightsquigarrow \text{true}$

$| v \in \mathcal{V} \rightsquigarrow \text{if } v \in L(s_i) \text{ then true else false}$

$| \neg \varphi \rightsquigarrow \text{if holdsBound}(\mathcal{M}, \pi, k, i, \varphi) \text{ then false else true}$

$| \varphi_1 \wedge \varphi_2 \rightsquigarrow \text{holdsBound}(\mathcal{M}, \pi, k, i, \varphi_1) \text{ and holdsBound}(\mathcal{M}, \pi, k, i, \varphi_2)$

$| \mathbf{G} \varphi \rightsquigarrow \text{false}$

$| \mathbf{F} \varphi \rightsquigarrow \text{exists } i \leq j \leq k: \text{holdsBound}(\mathcal{M}, \pi, k, j, \varphi)$

$| \mathbf{X} \varphi \rightsquigarrow i < k \text{ and holdsBound}(\mathcal{M}, \pi, k, i + 1, \varphi)$

$| \varphi_1 \mathbf{U} \varphi_2 \rightsquigarrow \text{exists } i \leq j \leq k: \text{holdsBound}(\mathcal{M}, \pi, k, j, \varphi_2) \text{ and}$
 $\text{for all } i \leq n < j: \text{holdsBound}(\mathcal{M}, \pi, k, n, \varphi_1)$

$| \varphi_1 \mathbf{R} \varphi_2 \rightsquigarrow \text{exists } i \leq j \leq k: \text{holdsBound}(\mathcal{M}, \pi, k, j, \varphi_1) \text{ and}$
 $\text{for all } i \leq n \leq j: \text{holdsBound}(\mathcal{M}, \pi, k, n, \varphi_2)$

- Wenn π eine k -Schleife ist, so wird einfach $\text{holds}(\mathcal{M}, \pi^i, \psi)$ aufgerufen

Das Bounded Model Checking Problem

Definition (Bounded Semantic ohne Schleife)

Sei $k \geq 0$ und π ein Pfad, der keine k -Schleife ist. Dann ist eine LTL Formel φ gültig entlang π mit Bound k ($\pi \models_k \varphi$), wenn $\text{holdsBound}(\mathcal{M}, \pi, k, 0, \varphi)$.

- Üblicherweise suchen wir ein Gegenbeispiel, d.h. *existiert ein Pfad, der die Bedingung verletzt*

Lemma

Für eine LTL Formel φ und einen Pfad π gilt $\pi \models_k \varphi$ impliziert $\pi \models \varphi$

Lemma

Existiert ein Pfad π , für den $\pi \models \varphi$ gilt, so existiert auch ein Pfad π' , so dass für ein $k \geq 0$ gilt $\pi' \models_k \varphi$.

Theorem (Bounded Model Checking)

Ein Pfad (Gegenbeispiel) π mit $\pi \models \varphi$ existiert gdw. ein $k \geq 0$ existiert mit $\pi \models_k \varphi$, d.h. $\text{holdsBound}(\mathcal{M}, \pi, k, 0, \varphi)$.

- Generiere eine Aussagenlogische Formel aus den drei Teilen

- 1 Kripke-Struktur \mathcal{M}
- 2 Bound k
- 3 LTL Formel φ

$\text{bmc2sat}(\mathcal{M}, k, \varphi)$

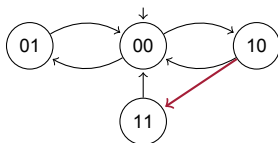
- Diese Formel codiert Constraints an einen Pfad $\pi = s_0 \rightarrow \dots$, der ein Zeuge für φ ist
- s_i steht für Zustand s im Zeitschritt i
- D.h. ist $\text{bmc2sat}(\mathcal{M}, k, \varphi)$ **erfüllbar**, codiert diese erfüllende Belegung ein **Gegenbeispiel** für die Verifikationsbedingung.
- Ist $\text{bmc2sat}(\mathcal{M}, k, \varphi)$ **unerfüllbar**, so ist die Verifikationsbedingung für alle Pfade der Länge k **wahr**.

Bestandteile von $\text{bmc2sat}(\mathcal{M}, k, \varphi)$

- 1 **trans2sat**: Constraints, dass $\pi = s_0, \dots, s_k$ ein gültiger Pfad ist
- 2 **loopCondition**: Schleifenbedingung; ist wahr, wenn π eine Schleife enthält
- 3 **formLoop, formNoLoop**: Constraints, dass π die Formel φ erfüllen muss (mit und ohne Schleife)

Kopieren der Übergangsrelation

- Die Übergangsfunktion muss bei einer Bound von k auch k mal kopiert werden
- ⇒ Es ist sinnvoll sich eine *generische* Version der Übergangsrelation zu speichern, die dann nur noch mit den entsprechenden Zeitschritten parametrisiert wird
- Realisierung wie \mathcal{T}_T
 - Bezeichnung $T(s, s')$



- Zustand: $s[1]s[0]$
- $T(s, s') = (\neg s[1] \wedge (s[0] \leftrightarrow \neg s'[0])) \vee (\neg s[0] \wedge (s[1] \leftrightarrow \neg s'[1])) \vee (s[0] \wedge s[1] \wedge \neg s'[0] \wedge \neg s'[1]) \vee (s[1] \wedge \neg s[0] \wedge s'[0] \wedge s'[1])$

Übersetzung der Übergangsrelation

Algorithmus: $\text{trans2sat}(\mathcal{M}, k)$

Eingabe: Kripke-Struktur $\mathcal{M} = (S, \longrightarrow, L)$, Bound k

Ausgabe: Aussagenlogische Formel, die \longrightarrow beschreibt

$$\mathcal{T}_S(s_0) \wedge \bigwedge_{i=0}^{k-1} T(s_i, s_{i+1})$$

Beispiel (Übersetzen der Übergangsrelation)

- $\text{trans2sat}(\mathcal{M}, 0) = \mathcal{T}_S(s_0)$
- $\text{trans2sat}(\mathcal{M}, 1) = \mathcal{T}_S(s_0) \wedge T(s_0, s_1)$
- $\text{trans2sat}(\mathcal{M}, 2) = \mathcal{T}_S(s_0) \wedge T(s_0, s_1) \wedge T(s_1, s_2)$
- $\text{trans2sat}(\mathcal{M}, 3) = \mathcal{T}_S(s_0) \wedge T(s_0, s_1) \wedge T(s_1, s_2) \wedge T(s_2, s_3)$

Übersetzung der Schleifenbedingung

- Übersetzung der Formel hängt davon ab, ob der Pfad π eine k -Schleife ist oder nicht.
- Codierung der Schleifenbedingung: Gibt es einen Rücksprung von s_k zu einen früheren Zustand

Algorithmus: $\text{loopCondition}(\mathcal{M}, k)$

Eingabe: Kripke-Struktur $\mathcal{M} = (S, \longrightarrow, L)$, Bound k

Ausgabe: Aussagenlogische Formel, die true ist, gdw. es einen Rücksprung vom Zustand s_k gibt

$$\bigvee_{l=0}^k T(s_k, s_l)$$

Beispiel (Übersetzen der Schleifenbedingung)

- $\text{loopCondition}(\mathcal{M}, 0) = T(s_0, s_0)$
- $\text{loopCondition}(\mathcal{M}, 1) = T(s_1, s_0) \vee T(s_1, s_1)$
- $\text{loopCondition}(\mathcal{M}, 2) = T(s_2, s_0) \vee T(s_2, s_1) \vee T(s_2, s_2)$

Übersetzung von Formeln (Mit Schleifen)

Definition (Nachfolger in einer Schleife)

Seien $l, i \leq k$. Dann ist $\text{succ}(i)$ für ein i in einer (k, l) -Schleife definiert als

$$\text{succ}(i) = \begin{cases} i + 1 & \text{für } i < k \\ l & \text{für } i = k \end{cases}$$

Algorithmus: $\text{formLoop}(\psi, k, l, i)$

Eingabe: LTL Formel ψ , Bound k , Nr. des Rücksprung Zustands l , Nr. des aktuellen Zustands i

Ausgabe: Aussagenlogische Formel, die sicherstellt, dass ein Pfad π , der eine (k, l) -Schleife ist, ψ erfüllt

$\text{formLoop}(\psi, k, l, i) = \psi \text{ match}$

$\top \rightsquigarrow \top$

$| \ v \in \mathcal{V} \rightsquigarrow v(s_i) \text{ (Indizierung von } v \text{ mit dem Zeitschritt } i)$

$| \ \neg \varphi \rightsquigarrow \neg \text{formLoop}(\varphi, k, l, i)$

$| \ \varphi_1 \wedge \varphi_2 \rightsquigarrow \text{formLoop}(\varphi_1, k, l, i) \wedge \text{formLoop}(\varphi_2, k, l, i)$

$| \ \dots$

Übersetzung von Formeln (Mit Schleifen)

Algorithmus: $\text{formLoop}(\psi, k, l, i)$

| ...
| $\mathbf{G} \varphi \rightsquigarrow \text{formLoop}(\varphi, k, l, i) \wedge \text{formLoop}(\mathbf{G} \varphi, k, l, \text{succ}(i))$
| $\mathbf{F} \varphi \rightsquigarrow \text{formLoop}(\varphi, k, l, i) \vee \text{formLoop}(\mathbf{F} \varphi, k, l, \text{succ}(i))$
| $\mathbf{X} \varphi \rightsquigarrow \text{formLoop}(\varphi, k, l, \text{succ}(i))$
| $\varphi_1 \mathbf{U} \varphi_2 \rightsquigarrow \text{formLoop}(\varphi_2, k, l, i) \vee$
 $(\text{formLoop}(\varphi_1, k, l, i) \wedge \text{formLoop}(\varphi_1 \mathbf{U} \varphi_2, k, l, \text{succ}(i)))$
| $\varphi_1 \mathbf{R} \varphi_2 \rightsquigarrow \text{formLoop}(\varphi_2, k, l, i) \wedge$
 $(\text{formLoop}(\varphi_1, k, l, i) \vee \text{formLoop}(\varphi_1 \mathbf{R} \varphi_2, k, l, \text{succ}(i)))$

- Speichert man die Subterme eindeutig (z.B. durch das Einführen neuer Variablen für Subterme), so ist die entstehende Formel linear in der Größe von ψ und k

Übersetzung von Formeln (Ohne Schleifen)

Algorithmus: $\text{formNoLoop}(\psi, k, i)$

Eingabe: LTL Formel ψ , Bound k , Nr. des aktuellen Zustands i

Ausgabe: Aussagenlogische Formel, die sicherstellt, dass ein Pfad π , der **keine** k -Schleife ist, ψ erfüllt

if $i > k$ then return \perp else ψ match

$\top \rightsquigarrow \top$

| $v \in \mathcal{V} \rightsquigarrow v(s_i)$ (Indizierung von v mit dem Zeitschritt i)

| $\neg \varphi \rightsquigarrow \neg \text{formNoLoop}(\varphi, k, i)$

| $\varphi_1 \wedge \varphi_2 \rightsquigarrow \text{formNoLoop}(\varphi_1, k, i) \wedge \text{formNoLoop}(\varphi_2, k, i)$

| $\mathbf{G} \varphi \rightsquigarrow \text{formNoLoop}(\varphi, k, i) \wedge \text{formNoLoop}(\mathbf{G} \varphi, k, i + 1)$

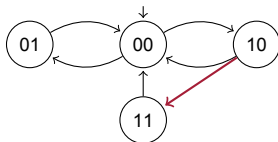
| $\mathbf{F} \varphi \rightsquigarrow \text{formNoLoop}(\varphi, k, i) \vee \text{formNoLoop}(\mathbf{F} \varphi, k, i + 1)$

| $\mathbf{X} \varphi \rightsquigarrow \text{formNoLoop}(\varphi, k, i + 1)$

| $\varphi_1 \mathbf{U} \varphi_2 \rightsquigarrow \text{formNoLoop}(\varphi_2, k, i) \vee$
 $\quad (\text{formNoLoop}(\varphi_1, k, i) \wedge \text{formNoLoop}(\varphi_1 \mathbf{U} \varphi_2, k, i + 1))$

| $\varphi_1 \mathbf{R} \varphi_2 \rightsquigarrow \text{formNoLoop}(\varphi_2, k, i) \wedge$
 $\quad (\text{formNoLoop}(\varphi_1, k, i) \vee \text{formNoLoop}(\varphi_1 \mathbf{R} \varphi_2, k, i + 1))$

Übersetzung von Formeln (Beispiele)



Wir möchten verifizieren, dass wir nie in den Zustand 11 kommen, d.h.
 $G \neg(s[1] \wedge s[0])$, d.h. wir suchen einen Pfad für $\varphi = F(s[1] \wedge s[0])$

Beispiel (Ohne Schleifen)

- $\text{formNoLoop}(\varphi, 2, 0) = (s_0[1] \wedge s_0[0]) \vee \text{formNoLoop}(\varphi, 2, 1)$
 - $\text{formNoLoop}(\varphi, 2, 1) = (s_1[1] \wedge s_1[0]) \vee \text{formNoLoop}(\varphi, 2, 2)$
 - $\text{formNoLoop}(\varphi, 2, 2) = (s_2[1] \wedge s_2[0]) \vee \text{formNoLoop}(\varphi, 2, 3)$
 - $\text{formNoLoop}(\varphi, 2, 3) = \perp$
- $\Rightarrow \text{formNoLoop}(\varphi, 2, 0) = (s_0[1] \wedge s_0[0]) \vee (s_1[1] \wedge s_1[0]) \vee (s_2[1] \wedge s_2[0])$

- Übersetzung für Version mit Schleifen identisch

Übersetzung des gesamten Systems

Algorithmus: $\text{bmc2sat}(\mathcal{M}, k, \varphi)$

Eingabe: Kripke-Struktur \mathcal{M} , Bound k , LTL Formel φ

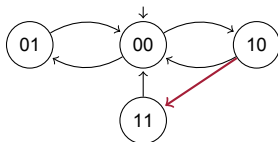
Ausgabe: Aussagenlogische Formel, die true ist, wenn eine Pfad π mit Länge k existiert, der φ erfüllt

$$\begin{aligned} & \text{trans2sat}(\mathcal{M}, k) \wedge \\ & [(\neg \text{loopCondition}(\mathcal{M}, k) \wedge \text{formNoLoop}(\varphi, k, 0)) \vee \\ & \bigvee_{l=0}^k (\text{T}(s_k, s_l) \wedge \text{formLoop}(\varphi, k, l, 0))] \end{aligned}$$

Erklärung

- $\text{trans2sat}(\mathcal{M}, k)$: k -faches Abwickeln der Übergangsrelation
- $\neg \text{loopCondition}(\mathcal{M}, k) \wedge \text{formNoLoop}(\varphi, k, 0)$: Der Pfad ist keine k -Schleife, also benutze die Codierung der Formel ohne Schleife
- $\bigvee_{l=0}^k (\text{T}(s_k, s_l) \wedge \text{formLoop}(\varphi, k, l, 0))$: Für ein (oder mehrere) l ist der Pfad eine (k, l) -Schleife; für dieses l muss dann auch die Codierung der Formel mit Schleife gelten

Finales Beispiel mit Bound $k = 2$



Suche einen Pfad, auf dem $\mathbf{F}(s[0] \wedge s[1])$ gilt

- $T(s, s') = (\neg s[1] \wedge (s[0] \leftrightarrow \neg s'[0])) \vee (\neg s[0] \wedge (s[1] \leftrightarrow \neg s'[1])) \vee (s[0] \wedge s[1] \wedge \neg s'[0] \wedge \neg s'[1]) \vee (s[1] \wedge \neg s[0] \wedge s'[0] \wedge s'[1])$
- $\text{trans2sat}(\mathcal{M}, 2) = \mathcal{T}_S(s_0) \wedge T(s_0, s_1) \wedge T(s_1, s_2)$
- $\text{loopCondition}(\mathcal{M}, 2) = T(s_2, s_0) \vee T(s_2, s_1) \vee T(s_2, s_2)$
- $\text{formNoLoop}(\varphi, 2, 0) = (s_0[1] \wedge s_0[0]) \vee (s_1[1] \wedge s_1[0]) \vee (s_2[1] \wedge s_2[0])$
- $\text{formLoop}(\varphi, 2, 2, 0) = (s_0[1] \wedge s_0[0]) \vee (s_1[1] \wedge s_1[0]) \vee (s_2[1] \wedge s_2[0])$
- $\text{bmc2sat}(\mathcal{M}, k, \varphi) = \text{trans2sat}(\mathcal{M}, k) \wedge [(\neg \text{loopCondition}(\mathcal{M}, k) \wedge \text{formNoLoop}(\varphi, k, 0)) \vee \bigvee_{l=0}^k (T(s_k, s_l) \wedge \text{formLoop}(\varphi, k, l, 0))]$

$\text{bmc2sat}(\mathcal{M}, k, \varphi)$ hat eine erfüllende Belegung:

$$\{\neg s_0[1], \neg s_0[0], s_1[1], \neg s_1[0], s_2[1], s_2[0]\}$$

Vollständigkeit

- Bounded Model Checking ist offensichtlich nicht vollständig
- Es gibt Techniken, für bestimmte Fälle eine maximale Grenze k zu bestimmen (z.B. Programmverifikation: obere Grenze einer Schleife ist bekannt)
- Prädestiniert zum Suchen von Fehlern, nicht zum Beweisen der Abwesenheit von Fehlern

Erweiterungen

- Das Benutzen von QBF-Solvern anstelle von SAT-Solvern erspart das k -fache Kopieren der Übergangsrelation
- Es gibt auch SMT-basierte Bounded Model Checker

Implementierungen

- Implementierung für das Software Bounded Model Checking: cbmc zum Verifizieren von C-Programmen



Literaturhinweis

- *M. Huth & M. Ryan.* **Logic in Computer Science Section 6.3.** Cambridge University Press, 2004.
 - *E. M. Clarke, O. Grumberg & D. A. Peled.* **Model Checking.** The MIT Press, 1999.
-
- *A. Biere et al.* **Bounded Model Checking.** Highly Dependable Software, Academic Press, 2003.
 - *A. Armando, J. Mantovani & L. Platania.* **Bounded Model Checking of Software using SMT Solvers Instead of SAT Solvers.** International Journal on Software Tools for Technology Transfer, 2009.
 - *N. Dershowitz, Z. Hanna & J. Katz.* **Bounded Model Checking with QBF.** Proceedings of the SAT 2005.



Web Links

- <http://nusmv.fbk.eu/> — Symbolic Model Checker NuSMV
- <http://www.cprover.org/cbmc/> — Software Bounded Model Checker für C Programme