

Modellreduktionstechniken für symbolische Kellersysteme

Dirk Richter

Oberseminar Softwaretechnik und Programmiersprachen
Vortrag am 03.06.2008

Martin-Luther-Universität Halle-Wittenberg
Naturwissenschaftliche Fakultät III, Institut für Informatik,
Lehrstuhl Softwaretechnik und Programmiersprachen,
<http://swt.informatik.uni-halle.de/>

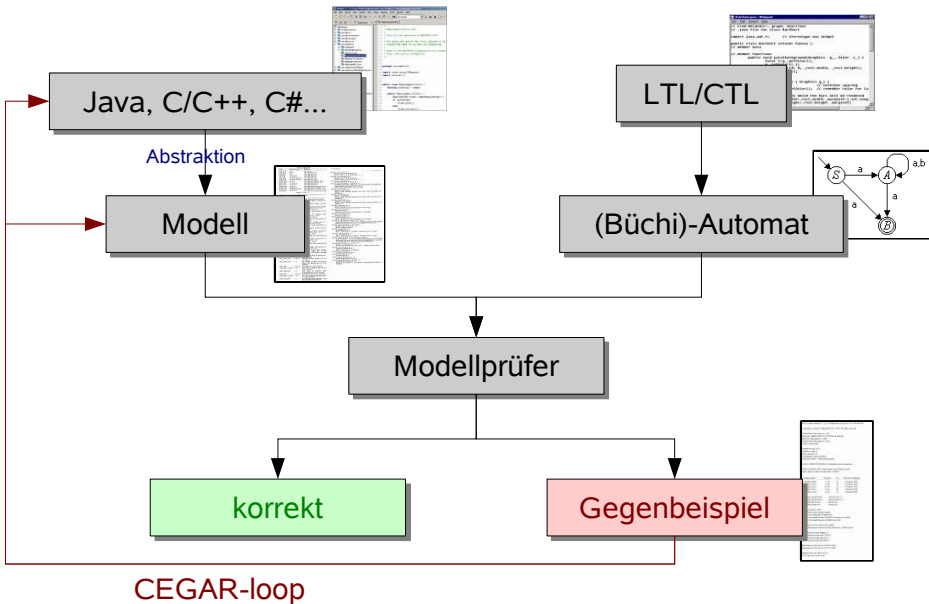
Inhaltsverzeichnis

- 1 Überblick
 - Begriffe
 - Quellcode → PDS
 - Remopla (SPDS) - Beispiel (mit JMoped generiert)
- 2 Modellreduktion via Modellanalysen
 - Äquivalenzanalyse
 - Optimale/minimale Repräsentantenwahl
 - R-Entscheidungsanalyse
 - Stotterreduktion
 - Einsatz des SMT-Solvers Yices
 - Intervallanalyse
- 3 Ergebnisse
 - Laufzeitvergleich der Versionen
 - Ausblick
 - Fragen?

wichtigste/neuste Punkte

- nur informale Erklärungen + Beispiele (Verständlichkeit)
- Komplexitätsresultat für minimale Repräsentantenwahl
- Blick auf Invarianz von Aussagen temporaler Formeln
- Eigenschaften zur Intervall-Analyse
- neue Ergebnisse 4-8 Bit Integer (1-6 zuvor)

Software Modellprüfung (engl. Software Model Checking)



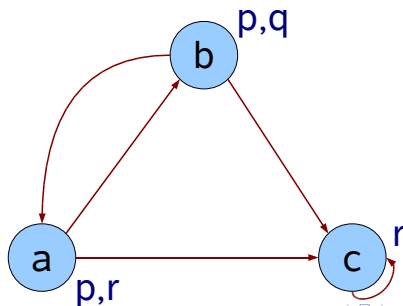
Kripkestruktur

Def. Kripkestruktur

$$M = (S, \rightarrow, L)$$

$$\rightarrow \subseteq S \times S$$

$$L : S \rightarrow \mathcal{P}(\text{Atome})$$



Erreichbarkeitsproblem

spezielles Erreichbarkeitsproblem $E(M, s, z)$

geg.: Kripkestruktur $M = (S, \rightarrow, L)$, $s, z \in S$

ges.: $s \rightarrow^* z$ Existiert Pfad von s nach z ?

verallgemeinertes Erreichbarkeitsproblem

statt $s, z \in S$ für Teilmengen $s, z \subseteq S$

- trivial für explizite/endliche Strukturen (Tiefensuche)

Kellersystem (PDS)

Kellersystem (Pushdown System)

$\mathcal{P} = (P, \Gamma, \hookrightarrow)$

P ... Zustände

Γ ... Kelleralphabet

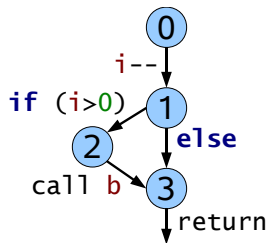
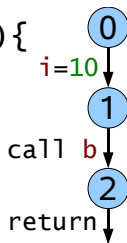
$\hookrightarrow \subseteq (P \times \Gamma) \times (P \times \Gamma^*)$... Transitionen

- Kellerautomat ohne Eingabe
- (p, w) Konfiguration, falls $p \in P, w \in \Gamma^*$
- $\rightarrow \subseteq (P \times \Gamma^*) \times (P \times \Gamma^*)$ Erweiterung von \hookrightarrow auf Konfigurationen:
 $(p, aw) \rightarrow (q, bw) :\Leftrightarrow (p, a) \hookrightarrow (q, b)$

Beispiel für Call-By-Value-Semantik (97 Transitionen)

```
void a(){
  int i=10;
  b(i);
}
```

```
void b(int i){
  i--;
  if (i>0)
    b(i);
}
```



Kontrollfluss ohne Daten...

Kontrollfluss und Daten... $x_i \in \{0, 1\}$ jetzt: **deterministisch**

$(p, a_0) \hookrightarrow (p, a_1)$

$(p, a_0) \hookrightarrow (p, (a_1, 1010))$

$(p, a_1) \hookrightarrow (p, b_0 a_2)$

$(p, (a_1, x_1 x_2 x_3 x_4)) \hookrightarrow (p, (b_0, x_1 x_2 x_3 x_4)(a_2, x_1 x_2 x_3 x_4))$

$(p, a_2) \hookrightarrow (p, \varepsilon)$

$(p, (a_2, x_1 x_2 x_3 x_4)) \hookrightarrow (p, \varepsilon)$

$(p, b_0) \hookrightarrow (p, b_1)$

$(p, (b_0, x_1 x_2 x_3 x_4)) \hookrightarrow (p, (b_1, x_1 x_2 x_3 x_4 - 1))$

$(p, b_1) \hookrightarrow (p, b_2)$

$(p, (b_1, x_1 x_2 x_3 x_4)) \hookrightarrow (p, (b_2, x_1 x_2 x_3 x_4)) \quad x_1 x_2 x_3 x_4 > 0000$

$(p, b_1) \hookrightarrow (p, b_3)$

$(p, (b_1, 0000)) \hookrightarrow (p, (b_3, 0000))$

$(p, b_2) \hookrightarrow (p, b_0 b_3)$

$(p, (b_2, x_1 x_2 x_3 x_4)) \hookrightarrow (p, (b_0, x_1 x_2 x_3 x_4)(b_3, x_1 x_2 x_3 x_4))$

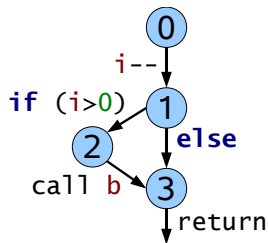
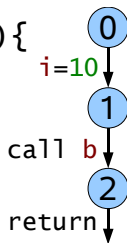
$(p, b_3) \hookrightarrow (p, \varepsilon)$

$(p, (b_3, x_1 x_2 x_3 x_4)) \hookrightarrow (p, \varepsilon)$

Beispiel für Call-By-Value-Semantik (97 Transitionen)

```
void a(){
  int i=10;
  b(i);
}
```

```
void b(int i){
  i--;
  if (i>0)
    b(i);
}
```



Kontrollfluss ohne Daten...

Kontrollfluss und Daten... $x_i \in \{0,1\}$ jetzt: **deterministisch**

$(p, a_0) \hookrightarrow (p, a_1)$

$(p, a_1) \hookrightarrow (p, b_0 a_2)$

$(p, a_2) \hookrightarrow (p, \varepsilon)$

$(p, b_0) \hookrightarrow (p, b_1)$

$(p, b_1) \hookrightarrow (p, b_2)$

$(p, b_1) \hookrightarrow (p, b_3)$

$(p, b_2) \hookrightarrow (p, b_0 b_3)$

$(p, b_3) \hookrightarrow (p, \varepsilon)$

$(p, a_0) \hookrightarrow (p, (a_1, 1010))$

$(p, (a_1, x_1 x_2 x_3 x_4)) \hookrightarrow (p, (b_0, x_1 x_2 x_3 x_4)(a_2, x_1 x_2 x_3 x_4))$

$(p, (a_2, x_1 x_2 x_3 x_4)) \hookrightarrow (p, \varepsilon)$

$(p, (b_0, x_1 x_2 x_3 x_4)) \hookrightarrow (p, (b_1, x_1 x_2 x_3 x_4 - 1))$

$(p, (b_1, x_1 x_2 x_3 x_4)) \hookrightarrow (p, (b_2, x_1 x_2 x_3 x_4)) \quad x_1 x_2 x_3 x_4 > 0000$

$(p, (b_1, 0000)) \hookrightarrow (p, (b_3, 0000))$

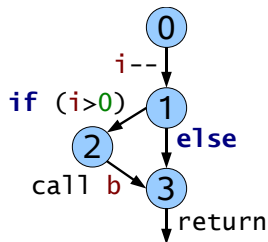
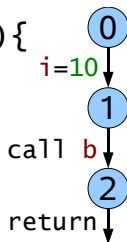
$(p, (b_2, x_1 x_2 x_3 x_4)) \hookrightarrow (p, (b_0, x_1 x_2 x_3 x_4)(b_3, x_1 x_2 x_3 x_4))$

$(p, (b_3, x_1 x_2 x_3 x_4)) \hookrightarrow (p, \varepsilon)$

Beispiel für Call-By-Value-Semantik (97 Transitionen)

```
void a(){
  int i=10;
  b(i);
}
```

```
void b(int i){
  i--;
  if (i>0)
    b(i);
}
```



Kontrollfluss ohne Daten...

Kontrollfluss und Daten... $x_i \in \{0, 1\}$ jetzt: **deterministisch**

$(p, a_0) \hookrightarrow (p, a_1)$

$(p, a_0) \hookrightarrow (p, (a_1, 1010))$

$(p, a_1) \hookrightarrow (p, b_0 a_2)$

$(p, (a_1, x_1 x_2 x_3 x_4)) \hookrightarrow (p, (b_0, x_1 x_2 x_3 x_4)(a_2, x_1 x_2 x_3 x_4))$

$(p, a_2) \hookrightarrow (p, \varepsilon)$

$(p, (a_2, x_1 x_2 x_3 x_4)) \hookrightarrow (p, \varepsilon)$

$(p, b_0) \hookrightarrow (p, b_1)$

$(p, (b_0, x_1 x_2 x_3 x_4)) \hookrightarrow (p, (b_1, x_1 x_2 x_3 x_4 - 1))$

$(p, b_1) \hookrightarrow (p, b_2)$

$(p, (b_1, x_1 x_2 x_3 x_4)) \hookrightarrow (p, (b_2, x_1 x_2 x_3 x_4)) \quad x_1 x_2 x_3 x_4 > 0000$

$(p, b_1) \hookrightarrow (p, b_3)$

$(p, (b_1, 0000)) \hookrightarrow (p, (b_3, 0000))$

$(p, b_2) \hookrightarrow (p, b_0 b_3)$

$(p, (b_2, x_1 x_2 x_3 x_4)) \hookrightarrow (p, (b_0, x_1 x_2 x_3 x_4)(b_3, x_1 x_2 x_3 x_4))$

$(p, b_3) \hookrightarrow (p, \varepsilon)$

$(p, (b_3, x_1 x_2 x_3 x_4)) \hookrightarrow (p, \varepsilon)$

```

define DEFAULT_INT_BITS 4
int heap[15];
int ptr(4);
module void a_main_ALjava_lang_String_V(int v0(4));
module void a_a_V();
module void a_b_I_V(int v0(4));
init jinit;

module void jinit()
{
ptr=1, A i (0,14) heap[i]=0;
goto a_main_ALjava_lang_String_V;
}

module void a_main_ALjava_lang_String_V(int v0(4))
{
a_main_ALjava_lang_String_V:   a_a_V();
a_main_ALjava_lang_String_V3:  return;
}

module void a_a_V()
{
int v0(4);
int s0;
a_a_V:   s0=10;
a_a_V2: v0=s0;
a_a_V3: s0=v0;
a_a_V4: a_b_I_V(s0);
a_a_V7: return;
}

module void a_b_I_V(int v0(4))
{
int s0;
a_b_I_V:   v0=v0-1;
a_b_I_V3:  s0=v0;
a_b_I_V4:  if
:: s0<=0 -> goto a_b_I_V11;
:: else -> skip;
fi;
a_b_I_V7:  s0=v0;
a_b_I_V8:  a_b_I_V(s0);
a_b_I_V11: return;
}

AssertError: goto AssertError;
Exception: goto Exception;
HeapOverflow: goto HeapOverflow;
StringBuilderOverflow: goto StringBuilderOverflow;

```


Temporale Logiken – Syntax

- spezifizieren Eigenschaften der Modelle
- Boolesche Ausdrücke über Atome der Kripkestruktur (\neg, \wedge, \vee)
+ Quantoren $X, F, G, U, R, (A, E)$
- sprechen über Berechnungspfade (LTL, LTL-X)
- sprechen über Berechnungsbaum (CTL, CTL*, ACTL)

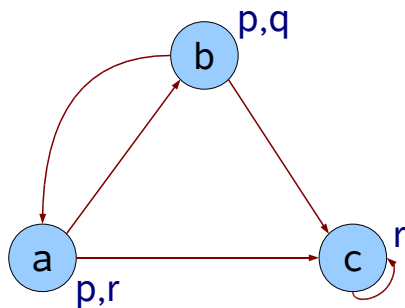
LTL

Quantoren A, E nicht erlaubt

- $LTL \subsetneq CTL^*$

Temporale Logiken – Semantik an Beispielen

- $EX p$
- $AFr \wedge p$
- $G p \vee r$
- $r U q$



Temporale Logiken – Beispiele als Motivation

- Erreichbarkeitsproblem $E(M, s, z) \Leftrightarrow$ Baum von s erfüllt **EFz**
- Verwendung nichtinitialisierter Variablen:

$$\mathbf{G} (Dec_v \rightarrow \mathbf{X}(\neg Use_v \mathbf{U}(Assign_v \vee Kill_v)))$$

- doppeltes Freigeben von Speicher:

$$\mathbf{G} (Free_v \rightarrow \mathbf{X}(\neg Free_v \mathbf{U}(Assign_v \vee Kill_v)))$$

- Abwesenheit von Deadlocks:

$$\mathbf{G} (RaiseEvent_i \rightarrow \mathbf{F}EnterEventHandler_i) \wedge$$

$$\mathbf{G} (EnterEventHandler_i \rightarrow \mathbf{F}EndEventHandler_i)$$

- teils mittels JML, SAL oder direkt im Assert-Style formulierbar

→ dafür: Erreichbarkeitstest $Assert(false)$ ausreichend

Stotter-Invarianz

stotter-frei

w -Wort $w = w_0 w_1 w_2 \cdots \in \Sigma^\omega$

- entweder: $\forall i : w_i \neq w_{i+1}$
- oder: $\forall i : w_0 = w_i$

Stotter-Invarianz

Sprache $L \subseteq \Sigma^\omega$ heißt stotter-invariant gdw.

$\forall w = w_0 w_1 w_2 \in \Sigma^\omega, \forall n_0, n_1, \dots \in \mathbb{N}$:

$$w \in L \Leftrightarrow w_0^{n_0} w_1^{n_1} \cdots \in L$$

Stotter-Invarianz

- in LTL-X/CTL*-X verboten: neXt-Operator
 - alle stotter-invarianten temporalen Eigenschaften in LTL-X/CTL*-X ausdrückbar (Peled, Wilke)
 - alle LTL-X/CTL*-X Formeln sind stotter-invariant (Lamport)
- Klasse LTL-X mit stotter-invarianten Formeln gleichsetzbar
- ω -Wort = Folge von Prädikaten einer Konfigurationenfolge eines Kellersystems (nur für Formel entscheidende Prädikate)

Statement-Slicing

- statische Untersuchung des interprozeduralen Kontrollflusses
 - Vorwärts + Rückwärts: pot. zu Fehlern führende Konfigur.
- Rest: entfernen aus SPDS-Modell

Motivation - Konstanten-Faltung/-Propagation

$x = 2; y = x + 3 * 2;$

- Faltung vs. Propagation

$x = 2; y = x + 6;$ vs. $x = 2; y = 2 + 3 * 2;$

- Artefakte durch automatische Transformation aus Quellcode

→ z.B. Simulation Operandenstack aus Javabytecode durch lokale Variablen

- einfachere und weniger BDD-Operationen im Modellprüfer
- **Semantikerhalt** für LTL und CTL* Formeln

Äquivalenzanalyse

Copy-Propagation-Analyse + Konstanten-Faltung + -Propagation

- Ziel: Aufbrechen von Copy-Chains durch Ersetzungen
→ Äquivalenzklassen auf Variablen und konst. Ausdrücken
- Verringerung Zustandsraum durch Wegfall lokaler Variablen
- für gängige Hochsprachen **unentscheidbar** (Postsches Korrespondenzproblem)
- aber: für SPDSs **entscheidbar**
- **Semantikerhalt** für LTL und CTL* Formeln nach Update
(z.B. Formel spricht über wegoptimierte Variable)

Äquivalenzanalyse in Aktion

| | |
|--------------------|--------|
| $x1 = a, x2 = x1;$ | push a |
| $x1 = a, x2 = x1;$ | push a |
| $x1 = x1 + x2;$ | add |

transformiert:

| | |
|--------------------|--------|
| $x1 = a, x2 = x1;$ | push a |
| $x1 = a, x2 = a;$ | push a |
| $x1 = a + a;$ | add |

- ang. keine weitere lesende Verwendung von $x2$
→ aus Modell entfernbar

Verwendung

- totale Ordnung auf Variablen
- ⇒ eindeutige Repräsentanten für Äquivalenzklassen
- so wählen, dass die wenigsten Variablen gebraucht werden

Ersetzung lesender Verwendungen durch äquivalente ...

- 1 ... Konstante
 - 2 ... kleinste Paramtervariable aus zugehöriger Methode
 - 3 ... kleinste globale Variable
 - 4 ... kleinste lokale Variable (keine weitere Art von Variablen)
- ⇒ größte Variablen im Idealfall nicht mehr verwendet
 - ⇒ keine neuen **False Negatives** (Äquivalenzumformung)

Optimale/minimale Repräsentantenwahl

- Ziel: minimale Anzahl benötigter Variablen
 - a priori vermutet: totale Ordnung auf Variablen genügt.
 - Vermutung **nun** bestätigt
 - lediglich nötig: 2-Teilung (Bipartitionierung)
- durch Ordnung induziert bei Variablenersetzung
- Reduktion Überdeckungsproblem
- NP-vollständig
- ⇒ Heuristik für Ordnung: Anz. Klassen je Variable (derzeit)
- denn Intuitiv: Je öfter Variable x in Äquivalenzklassen vorkommt, desto öfter kann sie andere Variablen ersetzen.

Was ist Überdeckungsproblem?

Gegeben: Überdeckungsmatrix A

| | x_1 | x_2 | x_3 | x_4 |
|-------|-------|-------|-------|-------|
| m_1 | 1 | 1 | 0 | 1 |
| m_2 | 0 | 1 | 0 | 1 |
| m_3 | 1 | 0 | 1 | 1 |
| m_4 | 0 | 0 | 1 | 1 |

- Menge m_i überdeckt Objekt x_j
- minimale Überdeckung aller x_j : $\{m_1, m_3\}$
- NP-vollständig (technische Inform.)

minimale Variablenwahl als Überdeckungsproblem

{a=b, c=d}

L1: use(a,b,c), c=a;

{a=b=c}

L2: use(b,d);

zu überdeckende Variablenverwendungen (Def-Use-Ziele):

| | $L1_a$ | $L1_b$ | $L1_c$ | $L2_b$ | $L2_d$ |
|-----|--------|--------|--------|--------|--------|
| a | 1 | 1 | 0 | 1 | 0 |
| b | 1 | 1 | 0 | 1 | 0 |
| c | 0 | 0 | 1 | 1 | 0 |
| d | 0 | 0 | 1 | 0 | 1 |

→ minimale Variablenwahl: {a, d}

⇒ optimale Ordnungen: alle der Form $[a, d, \dots]$ und $[d, a, \dots]$

Transformation nach Variablenwahl $\{a, d\}$

```
# {a=b, c=d}
```

```
L1: use(a, b, c), c=a;
```

```
# {a=b=c}
```

```
L2: use(b, d);
```

```
L1: use(a, a, d), c=a;
```

```
L2: use(a, d);
```

Variable c hat keine **lesende** Verwendung mehr → entfernbär

```
L1: use(a, a, d);
```

```
L2: use(a, d);
```

minimale Repräsentantenwahl NP-vollständig

- gerade: minimale Variablenwahl als Überdeckungsproblem
 - ! "Ein Apfel ist eine Frucht, also sind alle Früchte Äpfel."
 - Reduktion des Überdeckungsproblems auf Repräsentantenwahl
 - **Gegeben: beliebige** Überdeckungsmatrix $A = (x_{ij})$
 - konstruktion Programm
- durch Repräsentantenwahl Überdeckung gelöst

Konstruktion Programm zu gegebener Überdeckungsmatrix

| A | x_1 | x_2 | x_3 | x_4 |
|-------|-------|-------|-------|-------|
| m_1 | 1 | 1 | | 1 |
| m_2 | | 1 | | 1 |
| m_3 | 1 | | 1 | 1 |
| m_4 | | | 1 | 1 |

| B | $L1_{x_1}$ | $L2_{x_2}$ | $L3_{x_3}$ | $L4_{x_4}$ |
|-------|------------|------------|------------|------------|
| x_1 | 1 | | | |
| x_2 | | 1 | | |
| x_3 | | | 1 | |
| x_4 | | | | 1 |
| m_1 | 1 | 1 | | 1 |
| m_2 | | 1 | | 1 |
| m_3 | 1 | | 1 | 1 |
| m_4 | | | 1 | 1 |

```
# {x1=m1=m3}
L1: use(x1,1);
# {x2=m1=m2}
L2: use(x2,2);
# {x3=m3=m4}
L3: use(x3,3);
# {x4=m1=m2=m3=m4}
L4: use(x4,4);
```

```
module use(int v, int p) {
  print(v);
  xi=undef, mi=undef; # i in [1..4]
  if
    :: p==1 -> m1=x2,m2=x2;
    :: p==2 -> m3=x3,m4=x3;
    :: p==3 -> m1=x4,m2=x4,m3=x4,m4=x4;
  fi; return; }
```

Konstruktion Programm zu gegebener Überdeckungsmatrix

| A | x_1 | x_2 | x_3 | x_4 |
|-------|-------|-------|-------|-------|
| m_1 | 1 | 1 | | 1 |
| m_2 | | 1 | | 1 |
| m_3 | 1 | | 1 | 1 |
| m_4 | | | 1 | 1 |

| B | $L1_{x_1}$ | $L2_{x_2}$ | $L3_{x_3}$ | $L4_{x_4}$ |
|-------|------------|------------|------------|------------|
| x_1 | 1 | | | |
| x_2 | | 1 | | |
| x_3 | | | 1 | |
| x_4 | | | | 1 |
| m_1 | 1 | 1 | | 1 |
| m_2 | | 1 | | 1 |
| m_3 | 1 | | 1 | 1 |
| m_4 | | | 1 | 1 |

```

# {x1=m1=m3}
L1: use(x1,1);
# {x2=m1=m2}
L2: use(x2,2);
# {x3=m3=m4}
L3: use(x3,3);
# {x4=m1=m2=m3=m4}
L4: use(x4,4);
    
```

```

module use(int v, int p) {
  print(v);
  xi=undef, mi=undef; # i in [1..4]
  if
    :: p==1 -> m1=x2,m2=x2;
    :: p==2 -> m3=x3,m4=x3;
    :: p==3 -> m1=x4,m2=x4,m3=x4,m4=x4;
  fi; return; }
    
```

Repräsentantenwahl für konstruiertes Programm

| B | $L1_{x_1}$ | $L2_{x_2}$ | $L3_{x_3}$ | $L4_{x_4}$ |
|-------|------------|------------|------------|------------|
| x_1 | 1 | | | |
| x_2 | | 1 | | |
| x_3 | | | 1 | |
| x_4 | | | | 1 |
| m_1 | 1 | 1 | | 1 |
| m_2 | | 1 | | 1 |
| m_3 | 1 | | 1 | 1 |
| m_4 | | | 1 | 1 |

- Repräsentantenwahl: $\{m_1, m_3\}$ (gleiche Lösung wie vorhin)
- wenn ursprüngliches Überdeckungsproblem $A = (x_{ij})$ lösbar
→ \exists **optimale** Lösung ohne x_i (Austausch: z.B. x_3 statt m_3)

Transformation für Programmblock L1..L4

- Repräsentantenwahl: $\{m_1, m_3\}$

$\{x_1=m_1=m_3\}$

L1: use(x1, 1);

$\{x_2=m_1=m_2\}$

L2: use(x2, 2);

$\{x_3=m_3=m_4\}$

L3: use(x3, 3);

$\{x_4=m_1=m_2=m_3=m_4\}$

L4: use(x4, 4);

L1: use(m1, 1);

L2: use(m1, 2);

L3: use(m3, 3);

L4: use(m1, 4);

Semantikerhalt für LTL und CTL* Formeln

- Update: Wann? Wie?
 - Ersetzung lesender Verwendung: Variablenwerte **unverändert**
- Wahrheitswert Prädikate + LTL und CTL* Formeln identisch
- Problem: "wegoptimierte" Variable (c von vorhin)
 - i.A. **nicht** durch **eine einzige** Variable ausdrückbar
 - ortsabhängige Substitution in Formel nötig:

| | |
|-----------------|------------|
| $\#\{c=e\}$ | |
| L1: $a=c, c=d;$ | L1: $a=e;$ |
| $\#\{c=d\}$ | |
| L2: $b=c;$ | L2: $b=d;$ |
| $\#\{b=c=d\}$ | |
| L3: $a=b;$ | L3: $a=b;$ |

konforme Variablentransformation

$\#\{c=e\}$

L1: $a=c, c=d;$

$\#\{c=d\}$

L2: $b=c;$

$\#\{b=c=d\}$

L3: $a=b;$

L1: $a=e;$

L2: $b=d;$

L3: $a=b;$

konforme Variablentransformation

A_{Li} ... Äquivalenzklassen, R ... Repräsentantenwahl

$M_{Li} : Vars \rightarrow Vars$

M_{Li} konform $:\Leftrightarrow \forall v \in Vars : M_{Li}(v) \in R \wedge A_{Li}(M_{Li}(v), v)$

ortsabhängige Substitution - am Beispiel

 $\#\{c=e\}$ L1: $a=c, c=d;$ L1: $a=e;$ $\#\{c=d\}$ L2: $b=c;$ L2: $b=d;$ $\#\{b=c=d\}$ L3: $a=b;$ L3: $a=b;$

- **Geg.:** Formel ϕ , konforme Variablentransformationen M_{Li}
 - z.B. $\phi = (p \Rightarrow XGp)$ mit Prädikat $p = (c \equiv 1)$
- Prädikatsubstitution für "wegoptimierte" Variablen
- ersetze p durch $\bigwedge_{Li \in Label} (Li \Rightarrow p[c \rightarrow M_{Li}(c)])$

ortsabhängige Substitution - am Beispiel

 $\#\{c=e\}$ L1: $a=c, c=d;$ $\#\{c=d\}$ L2: $b=c;$ $\#\{b=c=d\}$ L3: $a=b;$ L1: $a=e;$ L2: $b=d;$ L3: $a=b;$

- ersetze p durch $\bigwedge_{Li \in Label} (Li \Rightarrow p[c \rightarrow M_{Li}(c)])$
- $(L1 \Rightarrow (e \equiv 1)) \wedge (L2 \Rightarrow (d \equiv 1)) \wedge (L3 \Rightarrow (c \equiv b))$
- unbefriedigend: MC exponentiell in der Länge der Formel
- Relativierung: nur in Anzahl unterschiedlicher Teilformeln (Etesami 00)

Das funktioniert aber nicht immer!

$\#\{c=e\}$

L1: $a=c, c=d;$

$\#\{c=d\}$

L2: $b=c, c=undef;$

$\#\{b=d\}$

L3: $a=b;$

- konforme Variablentransformation $M_{L3}(c)$ wobei $c \notin R$?
- Beobachtung: $\phi = (p \Rightarrow XGp)$ verwendet c **indirekt** über p
→ c nur **durch sich selbst** überdeckbar in L3 $\Rightarrow c \in R$
⇒ ergänze Repräsentantenwahl um indirekte Verwendungen
- erst dann: **Semantikerhalt** für LTL und CTL* Formeln

Indirekte Verwendung per Prädikat - Beispiel

```
#{c=e}
```

```
L1: a=c, c=d;
```

```
#{c=d}
```

```
L2: b=c, c=undef;
```

```
#{b=d}
```

```
L3: a=b;
```

```
#{c=e}
```

```
L1: a=c, c=d;
```

```
#{c=d}
```

```
L2: b=c, c=undef;
```

```
#{b=d}
```

```
L3: a=b, pred_use(c);
```

- L1 + L2: $\text{use}(c) \Rightarrow$ keine indirekte Verwendung nötig
- L3: Einführung indirekter Verwendung *pred_use(c)*

Indirekte Verwendung per Prädikat - Beispiel

$\#\{c=e\}$

L1: $a=c, c=d;$

$\#\{c=d\}$

L2: $b=c, c=undef;$

$\#\{b=d\}$

L3: $a=b;$

$\#\{c=e\}$

L1: $a=c, c=d;$

$\#\{c=d\}$

L2: $b=c, c=undef;$

$\#\{b=d\}$

L3: $a=b, \text{pred_use}(c);$

- L1 + L2: $\text{use}(c) \Rightarrow$ keine indirekte Verwendung nötig
- L3: Einführung indirekter Verwendung $\text{pred_use}(c)$

zugehörige Überdeckungsmatrix und Repräsentantenwahl

$\#\{c=e\}$

L1: a=c, c=d;

$\#\{c=d\}$

L2: b=c, c=undef;

$\#\{b=d\}$

L3: a=b;

$\#\{c=e\}$

L1: a=c, c=d;

$\#\{c=d\}$

L2: b=c, c=undef;

$\#\{b=d\}$

L3: a=b, pred_use(c);

| alt | L1 _c | L1 _d | L2 _c | L3 _b |
|-----|-----------------|-----------------|-----------------|-----------------|
| a | | | | |
| b | | | | 1 |
| c | 1 | | 1 | |
| d | | 1 | 1 | 1 |
| e | 1 | | | |

| neu | L1 _c | L1 _d | L2 _c | L3 _b | L3 _c |
|-----|-----------------|-----------------|-----------------|-----------------|-----------------|
| a | | | | | |
| b | | | | 1 | |
| c | 1 | | 1 | | 1 |
| d | | 1 | 1 | 1 | |
| e | 1 | | | | |

- Repräsentantenwahl: $R_{alt} = \{d, e\}$, $R_{neu} = \{c, d\}$

Definition Lebendigkeit (Muchnick)

Lebendigkeit

Variable x im Punkt p **lebendig** $:\Leftrightarrow$

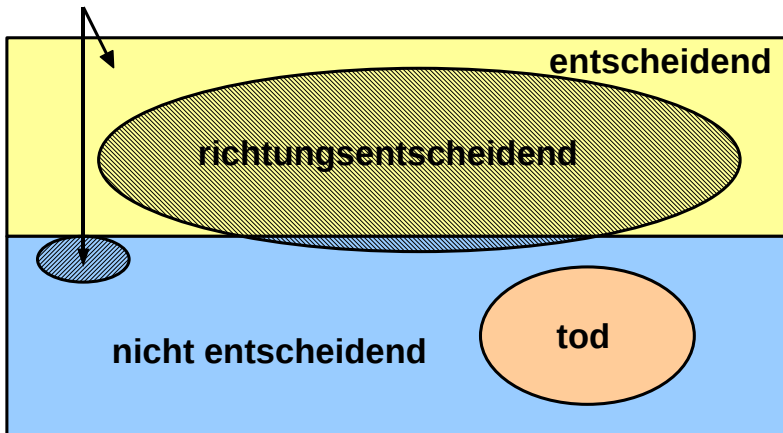
$$\exists q : Use(x)@q \wedge p \rightsquigarrow q \rightsquigarrow r \in Final \wedge (\neg Def(x) \cup q)@p$$

Motivation - R-Entscheidungsanalyse

- überdurchschnittlich viele überflüssige Zuweisungen und Variablen nach voriger Transformation
 - Bestimmung der Variablen, welche über die Erreichbarkeit im Modell **entscheiden**
 - **mehr** als nur Lebendigkeit (z.B. für Erreichbarkeit irrelevante Ausgabe auf Konsole)
 - für gängige Hochsprachen unentscheidbar (Halteproblem)
 - SPDSs: entscheidbar
- Approx.: **Richtung** vom Kontrollfluss beeinflussende Variablen
- richtungsentscheidend (tod \Rightarrow nicht richtungsentscheidend)

Am Modell-Kontrollpunkt p ist eine Variable v ...

Kontrollfluss versackend



potentiell neue False Negatives und Vergleich zu Slicing

```
x = x+1;
goto Error;
```

Entnahme vom eigentlich entscheidenden x führt zu:

```
goto Error;
```

- Abstraktion von Arithmetik auf "unwichtigen" Variablen
- spezielle Form des Slicing (x versackend und entscheidend)

Verwendung

- Elimination seiteneffektfreier Zuweisungen an nicht richtungsentscheidene Variablen
- Modulaufruf $\in rhs$: Seiteneffektanalyse auf Modulebene nötig
- Approximation (Modulaufrufe bleiben erhalten)
- Weiterhin: Identifikation aller nichtlebendigen Variablen, des gesamten toten Codes (und mehr)
- False Negative gdw. v nicht r-entscheidend + entscheidend
- v Kontrollfluss versackend
- arithmetischer Überlauf an unwichtigen Variablen

Semantikerhalt für LTL-X und CTL*-X Formeln

- fehlende Zwischenkonfigurationen
- fehlende "Buchstaben" in Prädikatfolge einer Konfigurationsfolge (Remopla-Programm-Ausführung)
- bestenfalls Semantikerhalt für LTL-X/CTL*-X
- Formel ϕ spricht über Variable $v \Rightarrow v$ **indirekt entscheidend**
- analog zu indirekte Verwendung via Prädikaten bei Äquivalenzanalyse

Motivation - Stotterreduktion

p: a = 5;

pq: a = 5, b = 8;

q: b = 8;

- Stotteräquivalenz bei Partial Order Reduction (Clarke et al.)
- ⇒ **konfliktfreie** symbolische Konfigurationen zusammenlegen
- ⇒ weniger Transitionen + kleinere Modelle
- ⇒ weniger BDD-Operationen im Modellprüfer
- intraprozedurale, flusssensitive, kontextinsensitive, pfadinsensitive Vorwärts-Analyse
- Transformationen **nur** innerhalb von Grundblöcken

lokal optimale \neq global optimale Transformation

```
l1: x = 0;  
l2: y = 6;  
l3: a = x+1;  
l4: b = y*2;
```

Zusammenfassung von l2 und l3 \Rightarrow lokales Optimum:

```
l1: x = 0;  
l2: y = 6, a = x+1;  
l4: b = y*2;
```

Aber: zwei Konfigurationenübergänge ausreichend:

```
l1: x = 0, y = 6;  
l3: a = x+1, b = y*2;
```

△ Semantikerhalt für LTL-X und CTL*-X Formeln ohne x,y,a,b.

→ **indirekte** Abhängigkeiten, dann optimale Reduktion via Sheduling

Einsatz des Yices SMT-Solvers

- Entartungen: $a + 2 > a$ oder $5 > 0$
- Yices: Sieger im SMT-Wettbewerb zur CAV 2007
- Verbesserung einer der Voraussetzungen voriger Analysen:
vollständig bedingte Modelle
- Vereinfachung boolescher Ausdrücke (strength reduction)
- Entfernung von if und do-Zweigen, falls Bedingung $\equiv false$
- nur 1 if-Zweig mit Bed. $\equiv true \Rightarrow$ unbedingte Anweisung
- ersetze *skip* X durch unbedingtes *skip*, falls $X \equiv true$
- Intervall-Informationen für Variablen \Rightarrow mehr Reduktionen

Motivation - Intervallanalyse

- Menge potentieller Variablenwerte durch Intervalle überschätzt
- Verringerung Zustandsraum durch kleinere Wertebereiche
- Verkleinerung Arrays auf maximale Lesegrenzen
- Identifikation potentieller Fehlabbstraktionen (Nichteinhaltung Nebenbedingungen, BUGs)
- Ausschluss semantisch nicht erreichbarer Konfigurationen
- mehr SMT-Reduktionen
- bessere Konstantenerkennung

bessere Konstantenerkennung

```

x = 0;
L: y = x/2;
if
  :: (x = 0) -> x = 1; goto L;
  :: else -> break;
fi

```

- y in punkt L konstant
- Konstantenfaltung/-Propagation: nein
- Intervallanalyse: ja

Best-Case: ArrayUtils

```
public class ArrayUtils
{
    public static void verifyReversal(int[] array)
    {
        int[] copy = new int[array.length];
        for (int i = 0; i < copy.length; i++) {
            copy[i] = array[i];
        }
        reverse(array);
        for (int i = 0; i < copy.length; i++) {
            assert(copy[i] == array[copy.length - 1 - i]);
        }
    }

    public static void main(String[] args) {
        int[] ar = new int[7];
        verifyReversal(ar);
    }
}
```

Best-Case: ArrayUtils

```
public static void reverse(int[] array) {
    if (array == null) {
        return;
    }
    int i = 0; int tmp;
    int j = array.length - 1;
    while (j > i) {
        tmp = array[j];
        array[j] = array[i];
        array[i] = tmp;
        j--; i++;
    }
}
```

- Laufzeiten: 1726 sec (ohne HalSPSI) vs. 1.44 sec (-Oi)
- in Remopla simulierter Heap hat nur **zwei** versch. Werte (0,7)
- Identifikation durch Intervall-Analyse
- ⇒ Reduktion um $5 \cdot 15 = 75$ Bits (also 2^{75} Zustände)
- Datenabstraktion: theoretisch Reduktion um weitere 30 Bits

30er Java-Assert-Benchmark@AMD X2 4200, 2GB RAM

Modellparameter: 4-8 Bit Integer, $5*30*16 = 2400$ Einzeltests

| method(s) | total time | total trtime | timeouts | memouts |
|--------------|------------|--------------|----------|---------|
| -Oaydti | 1.96h | 0.1h | 0% | 6.7% |
| -Oyi | 8.8h | 0.1h | 0.7% | 4% |
| -Ob | 20.6h | 0.1h | 0.7% | 20% |
| -Oaydt | 29.7h | 0.1h | 2% | 18.7% |
| -Oayd | 43.9h | 0.1h | 3.3% | 19.3% |
| ohne HalSPSI | 48.9h | 0.1h | 0.7% | 25.3% |

- 4% vorheriger Laufzeit
- 21% Speicherüberläufe verhindert (mc erst ermöglicht)
- 7%-10% Model-Checker überflüssig (Parameterabhängig)

weitere Eigenschaften

```
<substitutions>
```

```

- <avg method="jmoped_r">0</avg>
- <avg method="b">0</avg>
- <avg method="i">0</avg>
- <avg method="y">0</avg>
- <avg method="aydti">73.9047619047619</avg>
- <avg method="a">90.33333333333333</avg>
- <avg method="ba">107.21428571428571</avg>
- <avg method="reorg_rev">0</avg>
- <avg method="bayd">125.96428571428571</avg>
- <avg method="bad">125.96428571428571</avg>
- <avg method="badt">125.96428571428571</avg>
- <avg method="baydt">125.96428571428571</avg>
- <avg method="ayd">90.33333333333333</avg>
- <avg method="ad">90.33333333333333</avg>
- <avg method="adt">90.33333333333333</avg>
- <avg method="aydt">90.33333333333333</avg>
</substitutions>

```

weitere Eigenschaften

```

<mccabe>
- <avg method="jmoped_r">56.53333333333333 </avg>
- <avg method="b">48.82142857142857 </avg>
- <avg method="i">32.63636363636363 </avg>
- <avg method="y">32.13636363636363 </avg>
- <avg method="aydti">28.047619047619047 </avg>
- <avg method="a">47.2 </avg>
- <avg method="ba">41.32142857142857 </avg>
- <avg method="reorg_rev">56.53333333333333 </avg>
- <avg method="bayd">41.77142857142857 </avg>
- <avg method="bad">42.82142857142857 </avg>
- <avg method="badt">42.82142857142857 </avg>
- <avg method="baydt">41.77142857142857 </avg>
- <avg method="ayd">41.186666666666667 </avg>
- <avg method="ad">51.53333333333333 </avg>
- <avg method="adt">51.53333333333333 </avg>
- <avg method="aydt">41.186666666666667 </avg>
</mccabe>

```

weitere Eigenschaften

```

<ZR>
- <avg method="jmoped_r">255.23333333333332 </avg>
- <avg method="b">251.28571428571428 </avg>
- <avg method="i">144.1818181818182 </avg>
- <avg method="yi">144.1818181818182 </avg>
- <avg method="aydti">102.81904761904762 </avg>
- <avg method="a">255.2 </avg>
- <avg method="ba">251.17857142857142 </avg>
- <avg method="reorg_rev">255.26666666666668 </avg>
- <avg method="bayd">157.07857142857142 </avg>
- <avg method="bad">158.10714285714286 </avg>
- <avg method="badt">157.75 </avg>
- <avg method="baydt">156.72142857142856 </avg>
- <avg method="ayd">148.67333333333335 </avg>
- <avg method="ad">149.73333333333332 </avg>
- <avg method="adt">149.16666666666666 </avg>
- <avg method="aydt">148.14 </avg>
</ZR>

```

weitere Eigenschaften

<simplified_ifs>

```

- <avg method="jmoped_r">0</avg>
- <avg method="b">0</avg>
- <avg method="i">4.454545454545454</avg>
- <avg method="y">4.863636363636363</avg>
- <avg method="aydti">4.333333333333333</avg>
- <avg method="a">4.666666666666667</avg>
- <avg method="ba">4.678571428571429</avg>
- <avg method="reorg_rev">0</avg>
- <avg method="bayd">5.321428571428571</avg>
- <avg method="bad">4.821428571428571</avg>
- <avg method="badt">4.821428571428571</avg>
- <avg method="baydt">5.321428571428571</avg>
- <avg method="ayd">5.166666666666667</avg>
- <avg method="ad">0</avg>
- <avg method="adt">0</avg>
- <avg method="aydt">5.166666666666667</avg>
</simplified_ifs>

```

weitere Eigenschaften

```

<removed_assigns>
- <avg method="jmoped_r">0</avg>
- <avg method="b">0</avg>
- <avg method="i">0</avg>
- <avg method="y">0</avg>
- <avg method="aydti">151.42857142857142</avg>
- <avg method="a">0</avg>
- <avg method="ba">0</avg>
- <avg method="reorg_rev">0</avg>
- <avg method="bayd">193.37857142857143</avg>
- <avg method="bad">193</avg>
- <avg method="badt">193</avg>
- <avg method="baydt">193.37857142857143</avg>
- <avg method="ayd">195.65333333333334</avg>
- <avg method="ad">195.26666666666668</avg>
- <avg method="adt">195.26666666666668</avg>
- <avg method="aydt">195.65333333333334</avg>
</removed_assigns>

```

weitere Eigenschaften

```
<stmts_merged>  
- <avg method="jmoped_r">0</avg>  
- <avg method="b">0</avg>  
- <avg method="i">0</avg>  
- <avg method="y">0</avg>  
- <avg method="aydt">35.095238095238095</avg>  
- <avg method="a">0</avg>  
- <avg method="ba">0</avg>  
- <avg method="reorg_rev">0</avg>  
- <avg method="bayd">0</avg>  
- <avg method="bad">0</avg>  
- <avg method="badt">38</avg>  
- <avg method="baydt">29.085714285714285</avg>  
- <avg method="ayd">0</avg>  
- <avg method="ad">0</avg>  
- <avg method="adt">39.233333333333334</avg>  
- <avg method="aydt">32.113333333333334</avg>  
</stmts_merged>
```

- Kontextsensitivität (Call-Strings vs. Context-Cloning)
 - formale Remopla-Semantik
 - formale Korrektheitsnachweise der Analysen
 - Abstraktion von Daten
- Constraints zur Programm-Pfad-Partitionierung (Bad Honnef)
- Analysen theoretisch exakt durchführbar
- untersuchen: lohnt sich das? Wie gut ist Approximation?
- Wann sind Approximationen besonders gut? (formal nachweisbare Kriterien)
 - weitere Programmiersprachen (Projektarbeit CIL)
 - Analysen direkt in Quellsprache
- ⇒ bessere Modellgenerierung + Vergleich (Java-Abhängigkeit?)
- statt Äquivalenz auf Variablen und Konstanten: komplette Ausdrücke? sinnvoll?
 - frühzeitige MC-Prädikatidentifikation (Einsparung CEGAR-loops)

