

Variablenelimination für symbolische Modelle (Elimination of variables in symbolic models)

Dirk Richter, Wolf Zimmermann

4. Workshop Modellbasiertes Testen
29.09.2009

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

Software Model Checking

Java, C/C++, C#...

Abstraktion

model

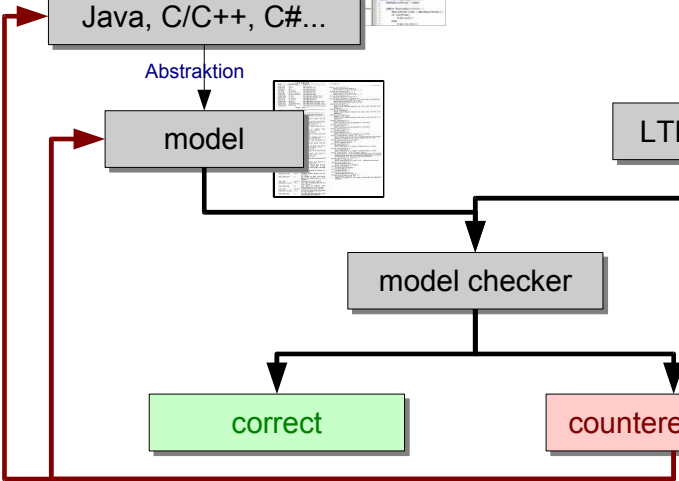
LTL/CTL

model checker

correct

counterexample

CEGAR-loop



small size and complexity of **models** important for

- software model checking
- model based testing
- test data generation
- code generation
- program understanding/analyzing
- simulation

① aim: **exact** recursion behavior

→ more precise results + less false alarms¹

② problem: infinite state space (PDS)

→ a priori compression of interior model structure
(source-to-source)

¹false negatives + false abstractions

Pushdown Systems (PDS)

Pushdown System

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

$P \dots$ states

$\Gamma \dots$ stack alphabet

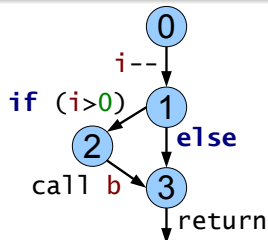
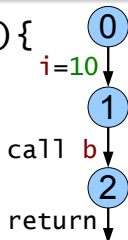
$\hookrightarrow \subseteq (P \times \Gamma) \times (P \times \Gamma^*) \dots$ transitions

- push-down automaton without input
- (p, w) **configuration**, iff $p \in P, w \in \Gamma^*$
- (p, a) is **head** of configuration $(p, aw), a \in \Gamma$

sourcecode \rightarrow SPDS using call-by-value

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

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



control flow without data...

control flow including data... $x_i \in \{0,1\}$

now: **deterministic**

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

New Syntax: Remopla (fragment)

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

```
module void a_a_V() {
  int v0(32);
  int s0(32);
  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; }
```

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

```
module void a_b_I_V(int v0(32)) {
  int s0(32);
  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;}
```

- size of stack alphabet: $2 * 10^{20}$ different symbols

Optimization - Phase I

- Substitute huge set of variables by a small set
 - "variable x can be replaced by y without changing semantics"
- **compute equivalence information** using a given analysis

A sample equivalence analysis

→ **compute equivalence information** using a given analysis

```

module void a_a_V() {
int v0(32);
int s0(32);
a_a_V:  s0=10;      # {s0=10}
a_a_V2: v0=s0;     # {v0=s0=10}
a_a_V3: s0=v0;     # {v0=s0=10}
a_a_V4: a_b_I_V(s0); # {v0=s0=10}
a_a_V7: return; }

module void a_b_I_V(int v0(32)) {
int s0(32);
a_b_I_V:  v0=v0-1;
a_b_I_V3:  s0=v0;      # {s0=v0}
a_b_I_V4:  if
           :: s0<=0 -> goto a_b_I_V11;
           :: else -> skip;    # {s0=v0}
fi;
a_b_I_V7:  s0=v0;      # {s0=v0}
a_b_I_V8:  a_b_I_V(s0); # {s0=v0}
a_b_I_V11: return;}

```

- undecidable for many high level languages
- conservative, interprocedural, contextinsensitive
- ⇒ fast + imprecise (not all equivalences discoverable)

Optimization - Phase II (Variable replacing)

- **replace occurring variables** by representatives or constants

```

module void a_a_V() {
int v0(32);
int s0(32);
a_a_V: s0=10; # {s0=10}
a_a_V2: v0=10; # {v0=s0=10}
a_a_V3: s0=10; # {v0=s0=10}
a_a_V4: a_b_I_V(10); # {v0=s0=10}
a_a_V7: return; }

```

```

module void a_b_I_V(int v0(32)) {
int s0(32);
a_b_I_V: v0=v0-1; # no equivalence
a_b_I_V3: s0=v0; # {s0=v0}
a_b_I_V4: if
:: v0 <=0 -> goto a_b_I_V11;
:: else -> skip; # {s0=v0}
fi;
a_b_I_V7: s0=v0; # {s0=v0}
a_b_I_V8: a_b_I_V(v0); # {s0=v0}
a_b_I_V11: return;}

```

LP-Solution_{a_b_I_V} : v0

LP-Solution_{a_b_I_V3} : v0

LP-Solution_{a_b_I_V4}(s0, v0) = v0

LP-Solution_{a_b_I_V7}(s0, v0) = v0

LP-Solution_{a_b_I_V8}(s0, v0) = v0

- **aim**: eliminate as much variables as possible (next phase)
- covering problem (NP-hard)

Optimization - Phase III (Variable elimination)

- **remove not needed variables from model (never read)**

```
module void a_a_V() {
```

```
a_a_V: skip; # {s0=10}
a_a_V2: skip; # {v0=s0=10}
a_a_V3: skip; # {v0=s0=10}
a_a_V4: a_b_I_V(10); # {v0=s0=10}
a_a_V7: return; }
```

```
module void a_b_I_V(int v0(32)) {
```

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

- **no reading** access of s_0 (both methods)
- **no reading** access of v_0 in `a_a_V`
 - ! side effect analysis (arithmetic overflow etc.)
- size of stack alphabet before: $2 * 10^{20}$
- now: $3 * 10^{10}$ symbols
- reduction by **factor**: $7.4 * 10^9$
- **more precise equivalence information**

Reductions after elimination of variables

191 benchmark files @ AMD 64 X2 4200+, 2GB RAM

Code-Sample	hd^-	hd^+	hd^*
ParameterRestrictions(7Bits)	$3,0 \cdot 10^{51}$	$2,3 \cdot 10^{49}$	$2,3 \cdot 10^{49}$
ConcreteFieldClass(8Bit)	$3,7 \cdot 10^{48}$	$1,4 \cdot 10^{40}$	$1,3 \cdot 10^{40}$
While (8Bit)	$4,3 \cdot 10^{45}$	$1,0 \cdot 10^{42}$	$1,0 \cdot 10^{42}$
average (all 191)	$8,6 \cdot 10^{89}$	$5,2 \cdot 10^{52}$	$2,0 \cdot 10^{52}$

Table: number of heads before (hd^-) and after reduction: approximative (hd^+) and exact version (hd^*)

Model Checking times after elimination of variables

- exact analyses: same effort as MC
- here: conservative version

Sample	4Bit-	4Bit+	5Bit-	5Bit+	6Bit-	6Bit+	7Bit-	7Bit+	8Bit-	8Bit+
ArrayFib	6 s	3 s	67 s	13 s	329 s	120 s	1446 s	522 s	MOut	MOut
ArrayUtils	8 s	1 s	109 s	2 s	453 s	12 s	1729 s	125 s	MOut	1826 s
Concrete...	0 s	0 s	6 s	0 s	39 s	0 s	111 s	1 s	329 s	1 s
Dispatching	329 s	12 s	MOut	758 s	MOut	MOut	MOut	MOut	MOut	MOut
Exceptions...	5 s	0 s	58 s	1 s	264 s	6 s	962 s	15 s	MOut	36 s
Fibonacci	1 s	2 s	5 s	5 s	37 s	14 s	115 s	51 s	328 s	169 s
IntBufferTest	368 s	21 s	MOut	919 s	MOut	MOut	MOut	MOut	MOut	MOut
Isq	0 s	1 s	1 s	1 s	4 s	4 s	30 s	11 s	355 s	44 s
LinkedList	2 s	0 s	12 s	1 s	59 s	5 s	234 s	12 s	702 s	34 s
MemoFib	4 s	0 s	MOut	9 s	MOut	113 s	MOut	502 s	MOut	MOut
Parameter...	5 s	1 s	68 s	9 s	340 s	110 s	1538 s	500 s	MOut	MOut
RecFib	7 s	0 s	7 s	0 s	39 s	1 s	111 s	1 s	317 s	5 s
ShortEval	343 s	20 s	MOut	839 s	MOut	MOut	MOut	MOut	MOut	MOut
While	0 s	0 s	1 s	0 s	1 s	1 s	2 s	1 s	3 s	2 s
false_neg...	339 s	15 s	MOut	741 s	MOut	MOut	MOut	MOut	MOut	MOut
total	1417 s	76 s	334 s	32 s	1565 s	273 s	6278 s	1239 s	2034 s	255 s

model checking times using Moped. "-" without and "+" with variable elimination.

! 4Bit: 0 MOuts vs. 8Bit: 9 MOuts

→ optimized 8Bit: 7 MOuts

Related Work

1. Spin (Bell, Promela): dead-code/var-elimination, constant prop./folding, partial order red., etc.
 - finite state space (DFS, no recursion)
 - slicing: maximal variable elimination not primary aim
 - ⇒ bigger state space (no minimization)
2. Alfred/Moped: infinite state space by PDS, but only trivial transformations
3. Macro Expansion: replace bdd variables by equivalent expressions
 - needs generation of underlying PDS

Summary & Conclusions

- 1 compute equivalence information using given analysis
- 2 replace variable occurrences by best representatives or constants
- 3 remove not needed variables from model (never read)

Take-Home-Messages

- a priori compression of interior structure simplifies models
- experiments: faster model checking
- **enables** model checking for larger bit widths (memout/timeout)
- useful: program analyses (compiler construction)
- combination (e.g. slicing): sometimes model checking **not required**

Thanks for your attention!

Questions?

Overview of further reduction techniques

non-parasitic (remains control/dataflow structure)

- reorganisation of variables
- control flow slicing
- SMT reduction
- constant propagation/-folding
- more trivial transformations

parasitic (changes control/dataflow structure)

- elimination of variables using equivalence information
- flow direction deciding reduction
- stutter reduction
- range reduction using interval analysis

Phase II in detail

{a=b, c=d}

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

{a=b=c}

L2: use(b,d);

variable occurrences to cover

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

→ minimal variable selection: {a, d}

Phase II in detail

{a=b, c=d}

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

=>

L1: use(a,a,d);

{a=b=c}

L2: use(b,d);

=>

L2: use(a,d);

variable occurrences to cover

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

→ minimal variable selection: {a, d}