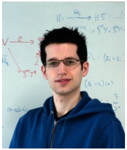
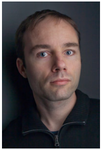
Dynamic Reductions for Model Checking Concurrent Software

Alfons Laarman
alfons@laarman.com

Henning Günther, Ana Sokolova and Georg Weissenbacher





Formal Methods in Systems Engineering
Vienna University of Technology

March 21, 2017

Reductions

Model Checking of Concurrent Software

1. Explosion of interleavings
2. Partial-order reduction vs Lipton reduction
3. Symbolic is a challenge
4. *Global* commutativity is needed, but a severe bottleneck

Reductions

Model Checking of Concurrent Software

1. Explosion of interleavings
2. Partial-order reduction vs Lipton reduction
3. Symbolic is a challenge
4. *Global* commutativity is needed, but a severe bottleneck

```
a = 1;        ||        x  = 1;
b = 2;                  x += 2;
                        x += 3;
```

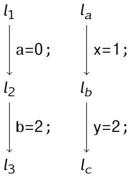
Reductions

Model Checking of Concurrent Software

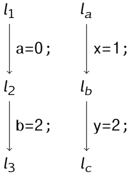
1. Explosion of interleavings
2. Partial-order reduction vs Lipton reduction
3. Symbolic is a challenge
4. *Global* commutativity is needed, but a severe bottleneck

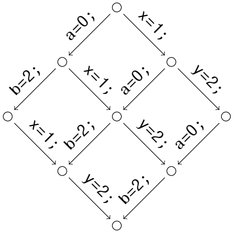
```
x = 1;            ||      x  = 1;
a = 1;                    x += 2;
b = 2;                    x += 3;
```

Lipton vs Partial-Order Reduction (POR)

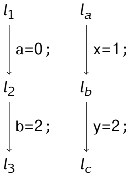


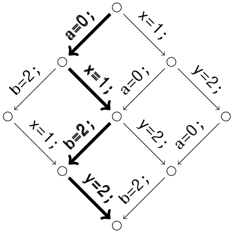
Lipton vs Partial-Order Reduction (POR)



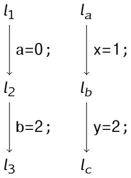


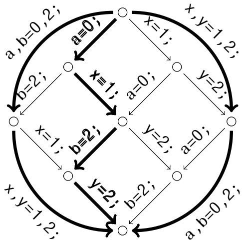
Lipton vs Partial-Order Reduction (POR)





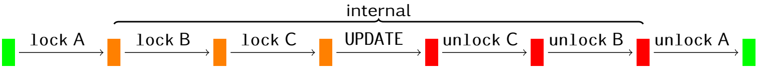
Lipton vs Partial-Order Reduction (POR)



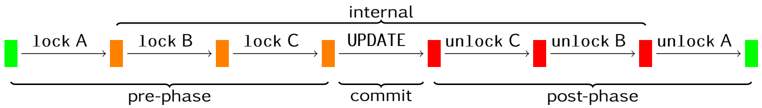


Transactions in databases

Transactions in databases



Transactions in databases



Commutativity

Action α right commutes with β, iff α can always be delayed after β:

$$\xrightarrow{\alpha}\xrightarrow{\beta} \quad \rightsquigarrow \quad \xrightarrow{\beta}\xrightarrow{\alpha}$$

Commutativity

Action α right commutes with β, iff α can always be delayed after β:

$$\xrightarrow{\alpha}\xrightarrow{\beta} \quad \rightsquigarrow \quad \xrightarrow{\beta}\xrightarrow{\alpha}$$

Definition (Right commutativity ($\overrightarrow{\bowtie}$))

$$\alpha \mathrel{\overrightarrow{\bowtie}} \beta \quad \text{iff} \quad \forall \sigma_1,\sigma_2,\sigma_3 : \begin{array}{ccc} \sigma_1 & & \\ \downarrow{\scriptstyle\alpha} & & \\ \sigma_2 & \xrightarrow{\beta} & \sigma_3 \end{array} \quad \Rightarrow \exists \sigma_4 : \begin{array}{ccc} \sigma_1 & \xrightarrow{\beta} & \sigma_4 \\ \downarrow{\scriptstyle\alpha} & & \downarrow{\scriptstyle\alpha} \\ \sigma_2 & \xrightarrow{\beta} & \sigma_3 \end{array}$$

Commutativity

Action α right commutes with β, iff α can always be delayed after β:

$$\xrightarrow{\alpha}\xrightarrow{\beta} \leadsto \xrightarrow{\beta}\xrightarrow{\alpha}$$

Definition (Right commutativity ($\overrightarrow{\bowtie}$))

$$\alpha \overrightarrow{\bowtie} \beta \quad \text{iff} \quad \forall \sigma_1,\sigma_2,\sigma_3 : \begin{array}{ccc} \sigma_1 & & \\ \downarrow\alpha & & \\ \sigma_2 & \xrightarrow{\beta} & \sigma_3 \end{array} \Rightarrow \exists \sigma_4 : \begin{array}{ccc} \sigma_1 & \xrightarrow{\beta} & \sigma_4 \\ \downarrow\alpha & & \downarrow\alpha \\ \sigma_2 & \xrightarrow{\beta} & \sigma_3 \end{array}$$

Example

- An action both-commutes with all actions that access a disjoint set of variables.
- A lock(/unlock) right(/right)-commutes with other lock and unlock operations.

Commutativity

Action α right commutes with β, iff α can always be delayed after β:

$$\xrightarrow{\alpha}\xrightarrow{\beta} \;\rightsquigarrow\; \xrightarrow{\beta}\xrightarrow{\alpha}$$

Definition (Right commutativity ($\overrightarrow{\bowtie}$))

$$\alpha \overrightarrow{\bowtie} \beta \quad \text{iff} \quad \forall \sigma_1, \sigma_2, \sigma_3 : \begin{array}{ccc} \sigma_1 & & \\ \downarrow\scriptstyle\alpha & & \\ \sigma_2 & \xrightarrow{\beta} & \sigma_3 \end{array} \quad \Rightarrow \exists \sigma_4 : \begin{array}{ccc} \sigma_1 & \xrightarrow{\beta} & \sigma_4 \\ \downarrow\scriptstyle\alpha & & \downarrow\scriptstyle\alpha \\ \sigma_2 & \xrightarrow{\beta} & \sigma_3 \end{array}$$

Example

- An action both-commutes with all actions that access a disjoint set of variables.
- A lock(/unlock) right(/right)-commutes with other lock and unlock operations.

Definition (Right-Movability)

The action α of thread i is a right-mover, iff for all $j \neq i$: $\xrightarrow{\alpha}_i \overrightarrow{\bowtie} \rightarrow_j$

Lipton Reduction

[Lipton '77, Lamport et al. '89]

Example (A statement sequence, where x is the only global variable)

```
a = 1; x = 2; b = 3; c = 4;
```

Lipton Reduction

[Lipton '77, Lamport et al. '89]

Example (A statement sequence, where x is the only global variable)

```
a = 1; x = 2; b = 3; c = 4;    ⇝    a, x, b, c = 1, 2, 3, 4;
```

Lipton Reduction

[LIPTON '77, LAMPORT ET AL. '89]

Lipton Reduction

A statement $\alpha_1;\ldots;\alpha_n$ of thread i can be reduced to $\alpha_1 \circ \cdots \circ \alpha_n$, if for some $1 \leq k < n$, and all $j \neq i$:

1. $\alpha_1,\ldots,\alpha_{k-1} \overset{\rightarrow}{\bowtie} \rightarrow_j$ (*pre-phase* statements (before α_k) are *right movers*)
2. $\alpha_{k+1},\ldots,\alpha_n \overset{\leftarrow}{\bowtie} \rightarrow_j$ (*post-phase* statements (after α_k) are *left movers*)
3. $\forall \sigma \exists \sigma' \colon \sigma \overset{\alpha_2}{\rightarrow}_i \circ \cdots \circ \overset{\alpha_n}{\rightarrow}_i \sigma'$ (statements after α_1 do not block)

Example (A statement sequence, where x is the only global variable)

`a = 1; x = 2; b = 3; c = 4;` $\rightsquigarrow$ `a, x, b, c = 1, 2, 3, 4;`

Lipton Reduction

[Lipton '77, Lamport et al. '89]

Lipton Reduction

A statement $\alpha_1; \ldots ; \alpha_n$ of thread i can be reduced to $\alpha_1 \circ \cdots \circ \alpha_n$, if for some $1 \leq k < n$, and all $j \neq i$:

1. $\alpha_1, \ldots, \alpha_{k-1} \overset{\rightarrow}{\bowtie} \rightarrow_j$ (*pre-phase* statements (before α_k) are *right movers*)
2. $\alpha_{k+1}, \ldots, \alpha_n \overset{\leftarrow}{\bowtie} \rightarrow_j$ (*post-phase* statements (after α_k) are *left movers*)
3. $\forall \sigma \exists \sigma' \colon \sigma \overset{\alpha_2}{\rightarrow}_i \circ \cdots \circ \overset{\alpha_n}{\rightarrow}_i \sigma'$ (statements after α_1 do not block)

Example (A statement sequence, where x is the only global variable)

```
a = 1; x = 2; b = 3; c = 4;   ⇝   a, x, b, c = 1, 2, 3, 4;
```

$\xrightarrow{x\ =\ 6}$ $\xrightarrow{a\ =\ 1}$ $\xrightarrow{x\ =\ 7}$ $\xrightarrow{x\ =\ 8}$ $\xrightarrow{x\ =\ 2}$ $\xrightarrow{b\ =\ 3}$ $\xrightarrow{x\ =\ 9}$ $\xrightarrow{c\ =\ 4}$

Lipton Reduction

[Lipton '77, Lamport et al. '89]

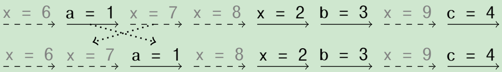
Lipton Reduction

A statement $\alpha_1;\ldots;\alpha_n$ of thread i can be reduced to $\alpha_1 \circ \cdots \circ \alpha_n$, if for some $1 \leq k < n$, and all $j \neq i$:

1. $\alpha_1,\ldots,\alpha_{k-1} \overset{\rightarrow}{\bowtie} \rightarrow_j$ (*pre-phase* statements (before α_k) are *right movers*)
2. $\alpha_{k+1},\ldots,\alpha_n \overset{\leftarrow}{\bowtie} \rightarrow_j$ (*post-phase* statements (after α_k) are *left movers*)
3. $\forall \sigma \exists \sigma' \colon \sigma \overset{\alpha_2}{\rightarrow}_i \circ \cdots \circ \overset{\alpha_n}{\rightarrow}_i \sigma'$ (statements after α_1 do not block)

Example (A statement sequence, where x is the only global variable)

```
a = 1; x = 2; b = 3; c = 4;   ⤳   a, x, b, c = 1, 2, 3, 4;
```



Lipton Reduction

[LIPTON '77, LAMPORT ET AL. '89]

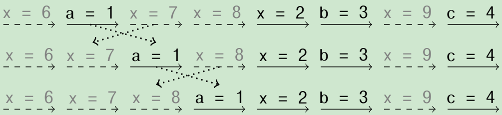
Lipton Reduction

A statement $\alpha_1;\ldots;\alpha_n$ of thread i can be reduced to $\alpha_1 \circ \cdots \circ \alpha_n$, if for some $1 \leq k < n$, and all $j \neq i$:

1. $\alpha_1,\ldots,\alpha_{k-1} \overrightarrow{\bowtie}\!\!\rightarrow_j$ (*pre-phase* statements (before α_k) are *right movers*)
2. $\alpha_{k+1},\ldots,\alpha_n \overleftarrow{\bowtie}\!\!\rightarrow_j$ (*post-phase* statements (after α_k) are *left movers*)
3. $\forall\sigma\,\exists\sigma' \colon \sigma \xrightarrow{\alpha_2}_i \circ \cdots \circ \xrightarrow{\alpha_n}_i \sigma'$ (statements after α_1 do not block)

Example (A statement sequence, where x is the only global variable)

`a = 1; x = 2; b = 3; c = 4;` $\rightsquigarrow$ `a, x, b, c = 1, 2, 3, 4;`



Lipton Reduction

[LIPTON '77, LAMPORT ET AL. '89]

Lipton Reduction

A statement $\alpha_1; \ldots; \alpha_n$ of thread i can be reduced to $\alpha_1 \circ \cdots \circ \alpha_n$, if for some $1 \le k < n$, and all $j \neq i$:

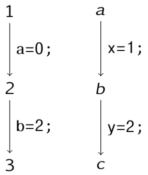
1. $\alpha_1, \ldots, \alpha_{k-1} \overset{\rightarrow}{\bowtie} \rightarrow_j$ (*pre-phase* statements (before α_k) are *right movers*)
2. $\alpha_{k+1}, \ldots, \alpha_n \overset{\leftarrow}{\bowtie} \rightarrow_j$ (*post-phase* statements (after α_k) are *left movers*)
3. $\forall \sigma \exists \sigma' \colon \sigma \xrightarrow{\alpha_2}_i \circ \cdots \circ \xrightarrow{\alpha_n}_i \sigma'$ (statements after α_1 do not block)

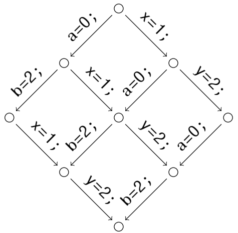
Example (A statement sequence, where x is the only global variable)

`a = 1; x = 2; b = 3; c = 4;` $\rightsquigarrow$ `a, x, b, c = 1, 2, 3, 4;`

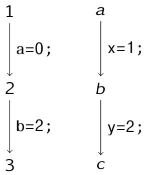
```
x = 6  a = 1  x = 7  x = 8  x = 2  b = 3  x = 9  c = 4
x = 6  x = 7  a = 1  x = 8  x = 2  b = 3  x = 9  c = 4
x = 6  x = 7  x = 8  a = 1  x = 2  b = 3  x = 9  c = 4
x = 6  x = 7  x = 8  a = 1  x = 2  b = 3  c = 4  x = 9
```

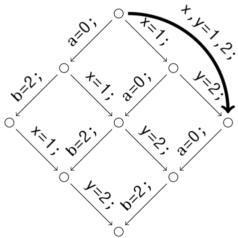
Movability is too strong a condition



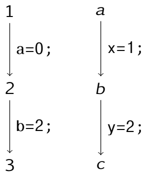


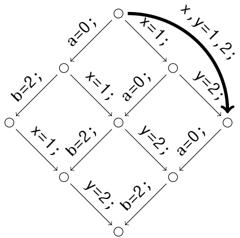
Movability is too strong a condition





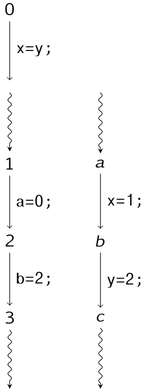
Movability is too strong a condition

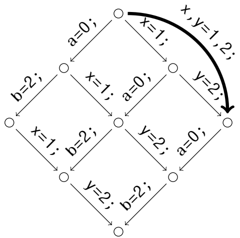




1. $\alpha_1, \ldots, \alpha_{k-1} \overset{\rightarrow}{\bowtie}\!\rightarrow_j$ (*pre-phase* statements (before α_k) are *right movers*)
2. $\alpha_{k+1}, \ldots, \alpha_n \overset{\leftarrow}{\bowtie}\!\rightarrow_j$ (*post-phase* statements (after α_k) are *left movers*)

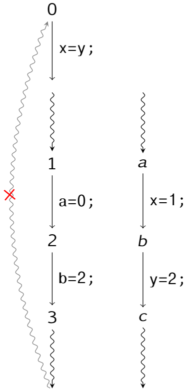
Movability is too strong a condition

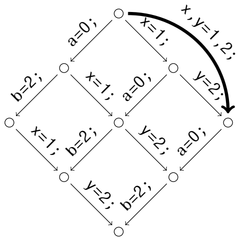




1. $\alpha_1, \ldots, \alpha_{k-1} \overset{\rightarrow}{\bowtie}\!\!\rightarrow_j$ (*pre-phase* statements (before α_k) are *right movers*)
2. $\alpha_{k+1}, \ldots, \alpha_n \overset{\leftarrow}{\bowtie}\!\!\rightarrow_j$ (*post-phase* statements (after α_k) are *left movers*)

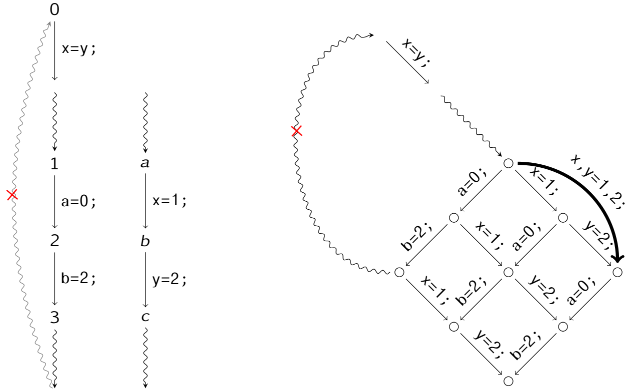
Movability is too strong a condition





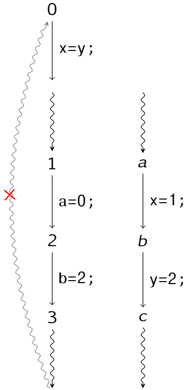
1. $\alpha_1, \ldots, \alpha_{k-1} \overrightarrow{\bowtie}\!\rightarrow_j$ (*pre-phase* statements (before α_k) are *right movers*)
2. $\alpha_{k+1}, \ldots, \alpha_n \overleftarrow{\bowtie}\!\rightarrow_j$ (*post-phase* statements (after α_k) are *left movers*)

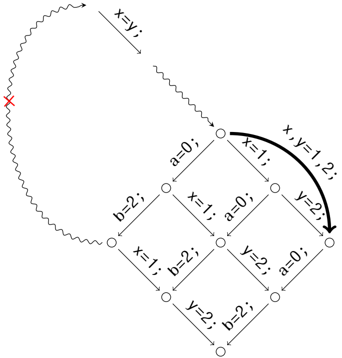
Movability is too strong a condition



1. $\alpha_1, \ldots, \alpha_{k-1} \overrightarrow{\bowtie}\!\!\rightarrow_j$ (*pre-phase* statements (before α_k) are *right movers*)
2. $\alpha_{k+1}, \ldots, \alpha_n \overleftarrow{\bowtie}\!\!\rightarrow_j$ (*post-phase* statements (after α_k) are *left movers*)

Movability is too strong a condition





1. $\alpha_1,\ldots,\alpha_{k-1} \overrightarrow{\bowtie}\!\rightarrow_j$ (*pre-phase* statements (before α_k) are *right movers*)
2. $\alpha_{k+1},\ldots,\alpha_n \overleftarrow{\bowtie}\!\rightarrow_j$ (*post-phase* statements (after α_k) are *left movers*)

Monotonicity is key!

Example

```
   int *data = NULL;
   void worker_thread(int tid) {
     if (data == NULL) {
       int *tmp = read_from_disk(1024);
W:     if (!CAS(&data, NULL, tmp)) free(tmp);
     }
     for (int i = 0; i < 512; i++)
R:     process(data[i + tid * 512]);
   }
   int main () {
     pthread_create(worker_thread, 0); // T1
     pthread_create(worker_thread, 1); // T2
   }
```

Example

```
int *data = NULL;
void worker_thread(int tid) {
  if (data == NULL) {
    int *tmp = read_from_disk(1024);
W:    if (!CAS(&data, NULL, tmp)) free(tmp);
```

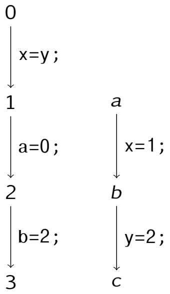
Example

Other dynamic conditions

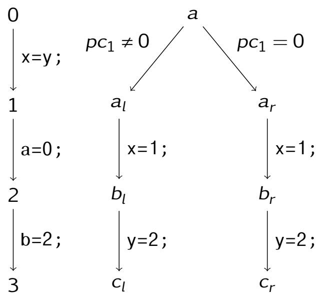
- Pointers / array indices that don't change value.
- Atomic Compare-And-Swap (CAS) operations to permanently grab resources.

```
  pthread_create(worker_thread, 0); // T1
  pthread_create(worker_thread, 1); // T2
}
```

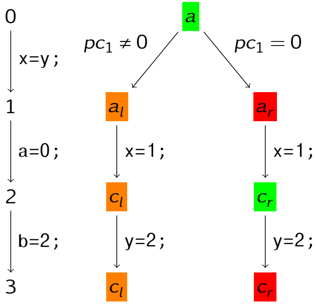
Instrumentation with dynamic reduction



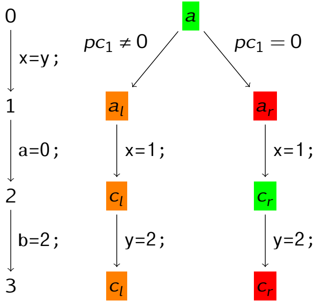
Instrumentation with dynamic reduction

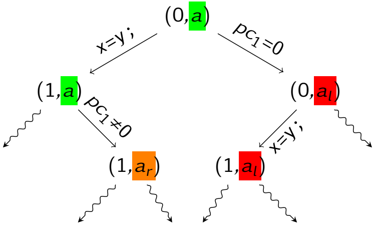


Instrumentation with dynamic reduction

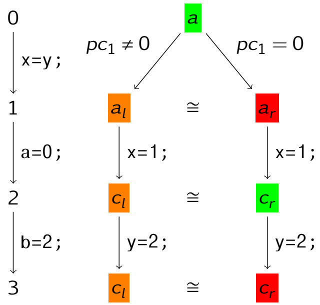


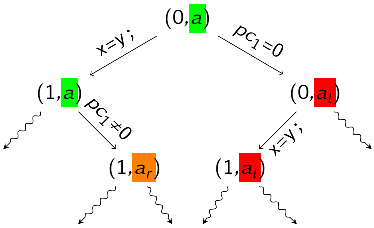
Instrumentation with dynamic reduction



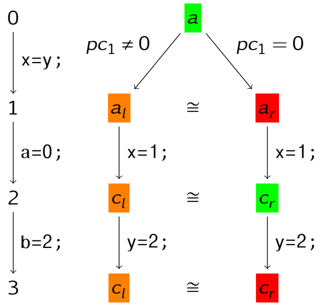


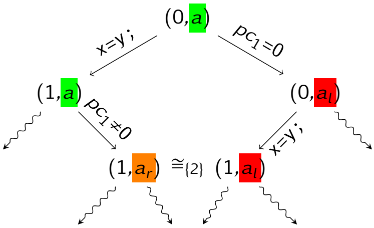
Instrumentation with dynamic reduction



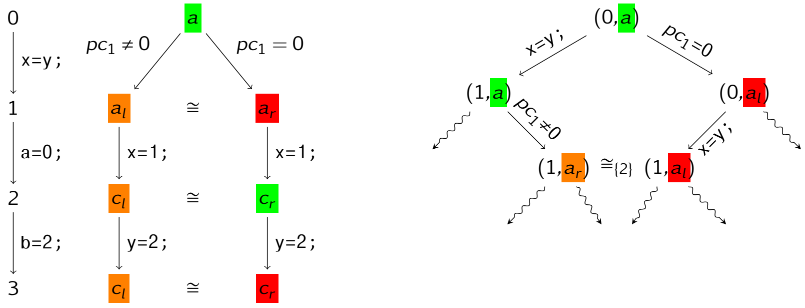


Instrumentation with dynamic reduction





Instrumentation with dynamic reduction



Definition (Dynamic *both*-moving conditions)

A state predicate c_α is a dynamic both-moving condition for an action α, if for all $j \neq i$:

1. $(c_\alpha \,/\!/\!\xrightarrow{\alpha}_i) \bowtie (c_\alpha \,/\!/\!\rightarrow_j)$
2. c_α is never disabled

Instrumentation with dynamic reduction

Example (Heuristics for other dynamic conditions)

```
int T[10] = {E,E,22,35,46,25,E,E,91,E};

int find-or-put(int v) {
   int hash = v / 10;
   for (int i = 0; i < 10; i++) {
      int index = (i + hash) % 10;
      if (CAS(&T[index], E, v)) {
         return INSERTED;
      } else if (T[index] == v)
         return FOUND;
   }
   return TABLE_FULL;
}
int main() {
   pthread_create(find-or-put, 25);
   pthread_create(find-or-put, 42);
}
```

Instrumentation with dynamic reduction

Example (Heuristics for other dynamic conditions)

```
int T[10] = {E,E,22,35,46,25,E,E,91,E};

int find-or-put(int v) {
   int hash = v / 10;
   for (int i = 0; i < 10; i++) {
      int index = (i + hash) % 10;
      if (CAS(&T[index], E, v)) {
         return INSERTED;
      } else if (T[index] == v)
         return FOUND;
   }
   return TABLE_FULL;
}
int main() {
   pthread_create(find-or-put, 25);
   pthread_create(find-or-put, 42);
}
```

c_α := `T[index] != E`

Movability up to bisimulation

Definition (Right-commutativity up to bisimulation ($\overrightarrow{\bowtie}_X$))

The transition relation $\xrightarrow{\alpha}_i$ right-commutes with $\xrightarrow{\beta}_j$ up to $\cong_X$, notation $\xrightarrow{\alpha}_i \overrightarrow{\bowtie}_X \xrightarrow{\beta}_j$, iff:

$$\forall \sigma_1, \sigma_2, \sigma_3 : \begin{array}{ccc} \sigma_1 & & \\ \downarrow \alpha & & \\ \sigma_2 & \xrightarrow{\beta}_j & \sigma_3 \end{array} \quad \Rightarrow \exists \sigma_3', \sigma_4 : \begin{array}{ccccc} \sigma_1 & \xrightarrow{\beta}_j & \sigma_4 & & \\ \downarrow \alpha & & & \searrow \alpha & \\ \sigma_2 & \xrightarrow{\beta}_j & \sigma_3 & \cong_X & \sigma_3' \end{array}$$

Definition (Right-movability up to bisimulation)

The action α of thread i is a right-mover up to $\cong_X$, iff for all $j \neq i$: $\xrightarrow{\alpha}_i \overrightarrow{\bowtie}_X \rightarrow_j$

Transaction Reduction

[Flanagan, Qadeer SoftMC'03]

Theorem (Reduction)

Let $(\rightarrow, \mathcal{S})$ be a transition system. For all $i, j \neq i$, let $\mathcal{S} = R_i \uplus L_i \uplus N_i$ such that

0. $L_i \,/\!\!/\!\rightarrow_i\!\setminus\!\!\setminus R_i = \emptyset$ — *post does not reach pre*
1. $\rightarrow_i\!\setminus\!\!\setminus R_i \overrightarrow{\bowtie} \rightarrow_j$ — *$\rightarrow_i$ into pre right commutes with $\rightarrow_j$*
2. $L_i \,/\!\!/\!\rightarrow_i \overleftarrow{\bowtie} \rightarrow_j$ — *$\rightarrow_i$ from post left commutes with $\rightarrow_j$*
3. $\forall \sigma \in L_i : \exists \sigma' \in N_i : \sigma \rightarrow_i^* \sigma'$ — *post phases terminate*

Let $\hookrightarrow_i \triangleq \bigcup_{j \neq i} N_j \,/\!\!/\!\rightarrow_i$ (i can only transition when all j are in an external state).

Suppose $\sigma \rightarrow^* \sigma'$, $\sigma \in N$ and $\sigma' \in N$, then there is an $\sigma'' \in N$ s.t. $\sigma \hookrightarrow^* \sigma''$.

Dynamic Transaction Reduction

[Günther, Laarman, Sokolova, Weissenbacher VMCAI'17]

Theorem (Reduction)

Let $(\rightarrow, \mathcal{S})$ be a transition system. For all $i, j \neq i$, let $\mathcal{S} = R_i \uplus L_i \uplus N_i$ such that
For each thread i, there exists a thread bisimulation relation $\cong_i$, and:

0. $L_i /\!\!/\!\rightarrow_i \!\backslash\!\backslash\, R_i = \emptyset$ — *post does not reach pre*
1. $\rightarrow_i \!\backslash\!\backslash\, R_i \overrightarrow{\bowtie}_{\{j\}} \rightarrow_j$ — *$\rightarrow_i$ into pre right commutes with $\rightarrow_j$*
2. $L_i /\!\!/\!\rightarrow_i \overleftarrow{\bowtie}_{\{i,j\}} \rightarrow_j$ — *$\rightarrow_i$ from post left commutes with $\rightarrow_j$*
3. $\forall \sigma \in L_i : \exists \sigma' \in N_i : \sigma \rightarrow_i^* \sigma'$ — *post phases terminate*
4. $\cong_i \subseteq L_j^2 \cup R_j^2 \cup N_j^2$ — *($\cong_i$ entails j-phase-equality)*

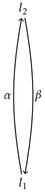
Let $\hookrightarrow_i \triangleq \bigcup_{j \neq i} N_j /\!\!/\!\rightarrow_i$ (i can only transition when all j are in an external state).
Let $\rightsquigarrow_i \triangleq N_i /\!\!/ (\hookrightarrow_i \!\backslash\!\backslash\, \overline{N_i})^* \hookrightarrow_i \!\backslash\!\backslash\, N_i$ (block steps skip internal states)
Suppose $\sigma \rightarrow^* \sigma'$, $\sigma \in N$ and $\sigma' \in N$, then there is an $\sigma'' \in N$ s.t. $\sigma \rightsquigarrow^* \sigma''$.

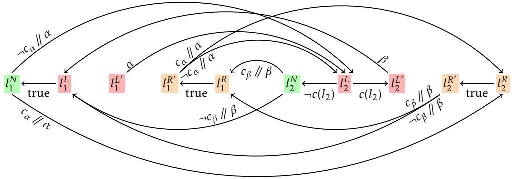
Complete instrumentation

$G_i \triangleq (V_i, \delta_i)$	V_i', δ' in G_i' (pictured)
$\forall (l_a, \alpha, l_b) \in \delta_i$:	$l_a^N \xrightarrow{c_\alpha /\!/ \alpha} l_b^R$, $l_a^N \xrightarrow{\neg c_\alpha /\!/ \alpha} l_b^L$
$\forall (l_a, \alpha, l_b) \in \delta_i$:	$l_a^{R'} \xrightarrow{c_\alpha /\!/ \alpha} l_b^R$, $l_a^{R'} \xrightarrow{\neg c_\alpha /\!/ \alpha} l_b^L$
$\forall l_a \in V_i$:	$l_a^R \xrightarrow{\text{true}} l_a^{R'}$
$\forall l_a \in V_i \setminus LFS_i$:	$l_a^L \xrightarrow{c(l_a)} l_a^{L'}$, $l_a^L \xrightarrow{\neg c(l_a)} l_a^N$ with $c(l_a) \triangleq \bigwedge_{(l_a,\alpha,l_b)\in\delta_i} c_\alpha$
$\forall (l_a, \alpha, l_b) \in \delta_i, l_a \in V_i \setminus LFS_i$:	$l_a^{L'} \xrightarrow{\alpha} l_b^L$
$\forall l_a \in LFS_i$:	$l_a^L \xrightarrow{\text{true}} l_a^N$

Complete instrumentation

$G_i \triangleq (V_i, \delta_i)$	V_i', δ' in G_i' (pictured)
$\forall (l_a, \alpha, l_b) \in \delta_i :$	$l_a^N \xrightarrow{c_\alpha \,/\!/\, \alpha} l_b^R$, $l_a^N \xrightarrow{\neg c_\alpha \,/\!/\, \alpha} l_b^L$
$\forall (l_a, \alpha, l_b) \in \delta_i :$	$l_a^{R'} \xrightarrow{c_\alpha \,/\!/\, \alpha} l_b^R$, $l_a^{R'} \xrightarrow{\neg c_\alpha \,/\!/\, \alpha} l_b^L$
$\forall l_a \in V_i :$	$l_a^R \xrightarrow{\text{true}} l_a^{R'}$
$\forall l_a \in V_i \setminus LFS_i :$	$l_a^L \xrightarrow{c(l_a)} l_a^{L'}$, $l_a^L \xrightarrow{\neg c(l_a)} l_a^N$ with $c(l_a) \triangleq \bigwedge_{(l_a, \alpha, l_b) \in \delta_i} c_\alpha$
$\forall (l_a, \alpha, l_b) \in \delta_i, l_a \in V_i \setminus LFS_i :$	$l_a^{L'} \xrightarrow{\alpha} l_b^L$
$\forall l_a \in LFS_i :$	$l_a^L \xrightarrow{\text{true}} l_a^N$





VVT

Vienna Verification Tool [Günther, Laarman, Weissenbacher SVCOMP'16]

http://vvt.forsyte.at/ (open source)

VVT

Vienna Verification Tool [Günther, Laarman, Weissenbacher SVCOMP'16]

http://vvt.forsyte.at/ (open source)

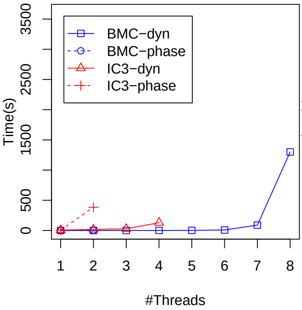
- BMC with all dynamic reductions (*BMC-dyn* in the graphs);
- BMC with only static reductions (*BMC-phase*);
- IC3 with all dynamic reductions (*IC3-dyn*); and
- IC3 with only static reductions (*IC3-phase*).

VVT

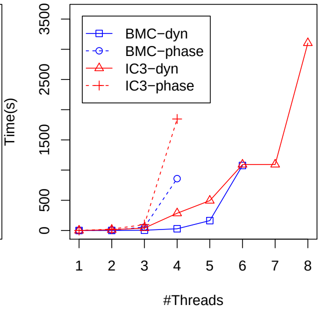
Vienna Verification Tool [Günther, Laarman, Weissenbacher SVCOMP'16]

`http://vvt.forsyte.at/` (open source)

- BMC with all dynamic reductions (*BMC-dyn* in the graphs);
- BMC with only static reductions (*BMC-phase*);
- IC3 with all dynamic reductions (*IC3-dyn*); and
- IC3 with only static reductions (*IC3-phase*).



Lazy initialization



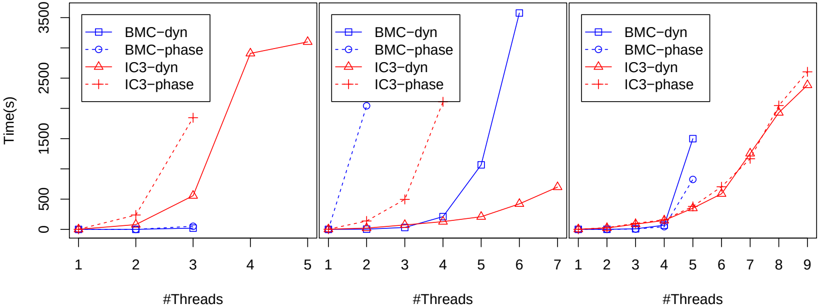
Dynamic locking

VVT

Vienna Verification Tool [Günther, Laarman, Weissenbacher SVCOMP'16]

`http://vvt.forsyte.at/` (open source)

- BMC with all dynamic reductions (*BMC-dyn* in the graphs);
- BMC with only static reductions (*BMC-phase*);
- IC3 with all dynamic reductions (*IC3-dyn*); and
- IC3 with only static reductions (*IC3-phase*).



Hash table reads/writes   Hash table reads   Hash table writes

Conclusions

Contributions

- Dynamic movers as in stubborn set POR
- "Straight-forward" instrumentation and encoding
- Dynamic mover conditions are suitable for symbolic model checking

Conclusions

Contributions

- Dynamic movers as in stubborn set POR
- "Straight-forward" instrumentation and encoding
- Dynamic mover conditions are suitable for symbolic model checking

Open Questions

- Is a weaker reduction theorem possible?
- How do transactions compare to POR?