Generating representative executions

Hendrik Maarand, Tarmo Uustalu

Institute of Cybernetics at Tallinn University of Technology

October 15, 2016

Introductory example

This is an excerpt from Dekker's mutual exclusion algorithm

Init: `x = 0; y = 0;`	
P_1	P_2
(a) `[x] := 1`	(c) `[y] := 1`
(b) `r1 := [y]`	(d) `r2 := [x]`
Observed? `r1 = 0; r2 = 0;`	

Introductory example

This is an excerpt from Dekker's mutual exclusion algorithm

Init: `x = 0; y = 0;`	
P_1	P_2
(a) `[x] := 1`	(c) `[y] := 1`
(b) `r1 := [y]`	(d) `r2 := [x]`
Observed? `r1 = 0; r2 = 0;`	

There are six possible interleavings:

abcd acbd cdab
cabd
acdb
cadb

Introductory example

This is an excerpt from Dekker's mutual exclusion algorithm

Init: `x = 0; y = 0;`	
P_1	P_2
(a) `[x] := 1`	(c) `[y] := 1`
(b) `r1 := [y]`	(d) `r2 := [x]`
Observed? `r1 = 0; r2 = 0;`	

There are six possible interleavings:

abcd acbd cdab

cabd

acdb

cadb

Mazurkiewicz traces

An independency $I \subseteq \Sigma \times \Sigma$ is an irreflexive and symmetric binary relation

$s, t \in \Sigma^*$ are Mazurkiewicz equivalent, $s \equiv_I t$, iff s can be transformed to t by a finite number of exchanges of adjacent, independent actions

A Mazurkiewicz trace $\sigma = [s]_I$ is the Mazurkiewicz equivalence class of a string

Let $\Sigma = \{a, b, c\}$ with $a \; I \; b$ and $b \; I \; c$

$$[ab]_I = \{ab, ba\}$$
$$[aba]_I = \{aba, baa, aab\}$$
$$[abc]_I = \{abc, bac, acb\}$$
$$[abca]_I = \{abca, baca, acba, acab\}$$

Foata normal form

A step is a subset $F \subseteq \Sigma$ of pairwise independent letters

The Foata normal form is a sequence of steps $F_1 \dots F_k$ such that $F_1, \dots, F_k$ are chosen from left to right with maximal cardinality

Since each step requires only one parallel execution step, the Foata normal form encodes a maximal parallel execution

Let $\Sigma = \{a, b, c, d\}$ and $a \; I \; c$ and $b \; I \; d$

The Foata normal form of $acbd$ is $(ac)(bd)$

Normality checking

We assume a total order on Σ

We say that a string is in Foata normal form if it can be split into steps such that the sequence of steps is the Foata normal form

A string is in Foata normal form if:

- For every step F_i, the letters in F_i are pairwise independent
- For every step F_i, the letters in F_i are in increasing order
- For all $a \in F_i$, there is a $b \in F_{i-1}$ such that $\neg(a\ I\ b)$

Independency

Init: `x = 0; y = 0;`	
P_1	P_2
(a) `[x] := 1`	(c) `[y] := 1`
(b) `r1 := [y]`	(d) `r2 := [x]`
Observed? `r1 = 0; r2 = 0;`	

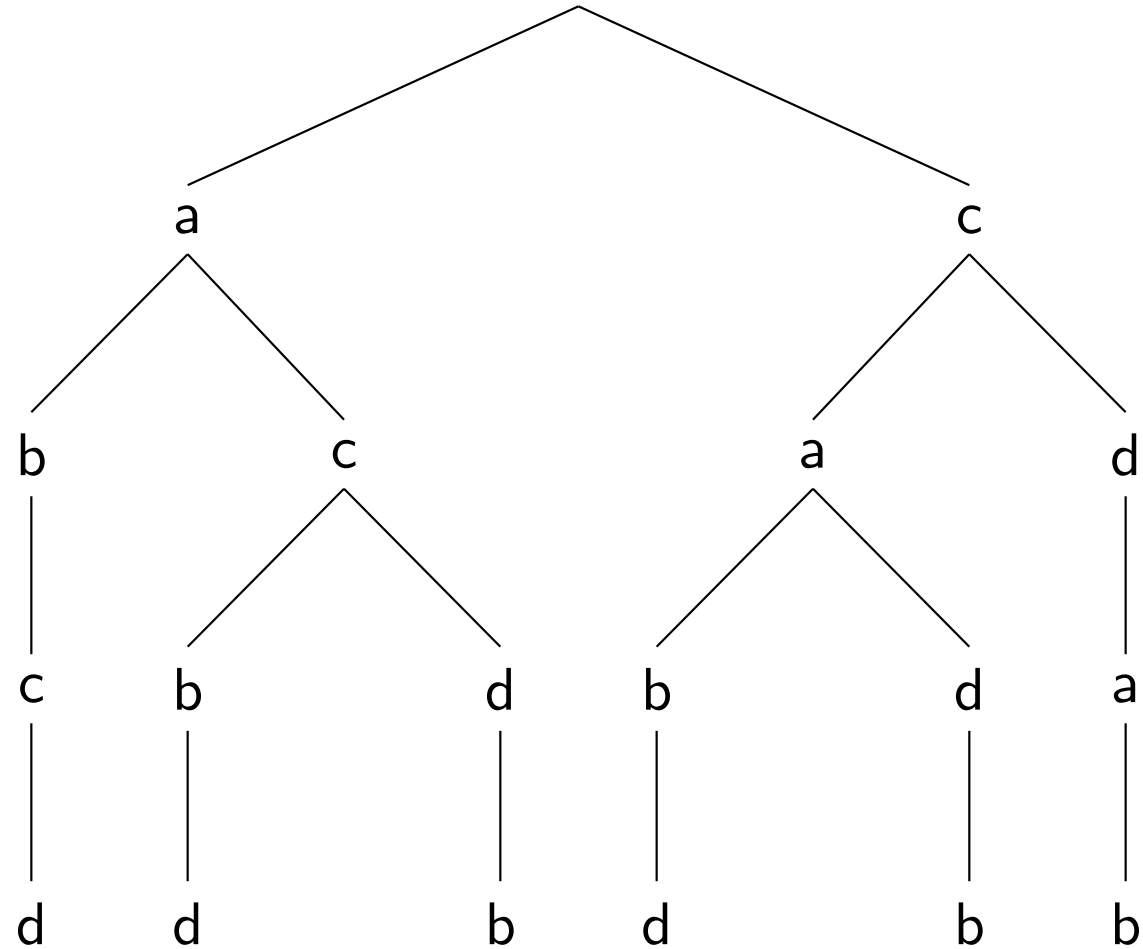
Independency is the complement of dependency

```
dep (p1, a1) (p2, a2)
  | p1 == p2  = True
  | otherwise = (isWrite a1 || isWrite a2)
                && var a1 == var a2
```

$I = \{(a, c), (b, d), (c, a), (d, b)\}$

Execution trees

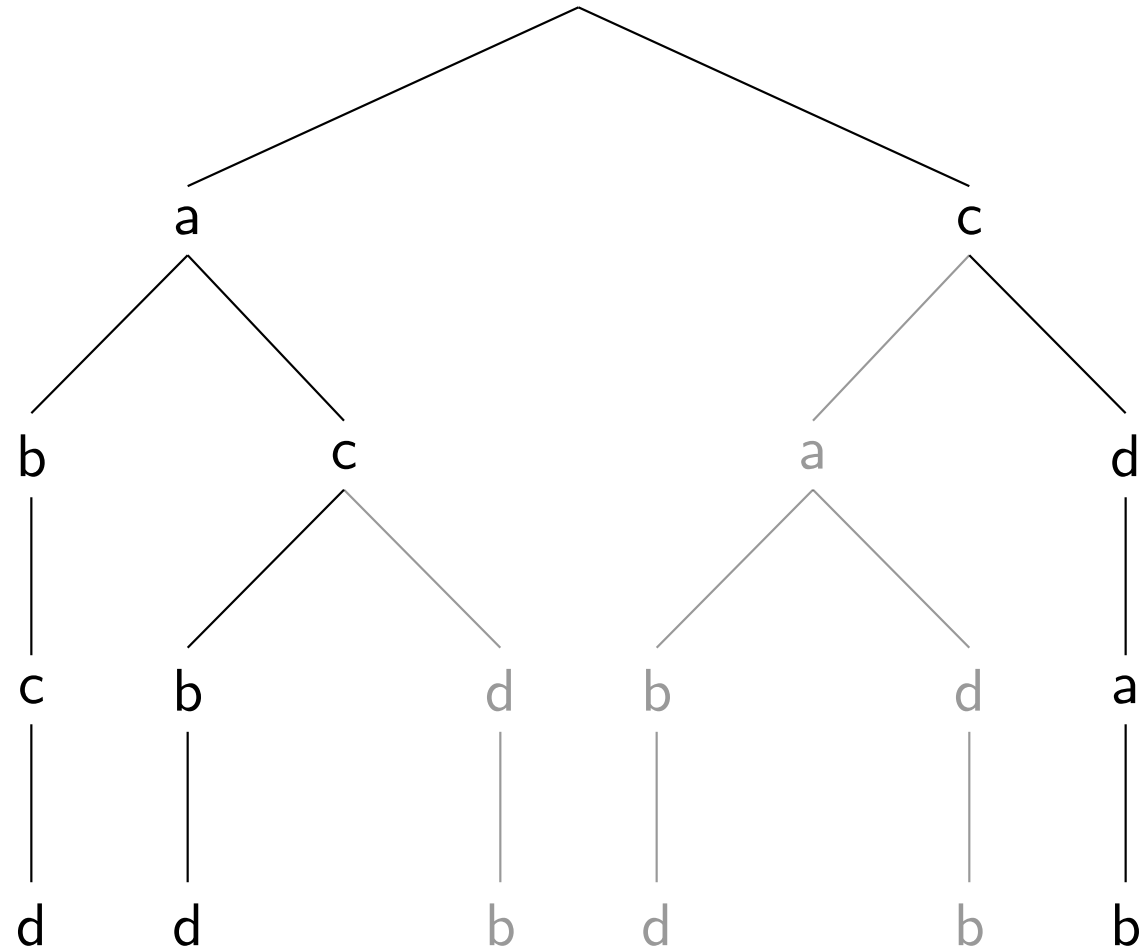
$I = \{(a, c), (b, d), (c, a), (d, b)\}$



$(a)(b)(c)(d)$ $(ac)(bd)$ $(c)(d)(a)(b)$

Execution trees

$I = \{(a, c), (b, d), (c, a), (d, b)\}$



$(a)(b)(c)(d)$ $(ac)(bd)$ $(c)(d)(a)(b)$

Other architectures

So far we have looked at sequentially consistent behaviour

Modern multiprocessors have more involved execution models; x86, for example, has a Total Store Order (TSO) model

In TSO, loads can be reordered with older stores

In Partial Store Order (PSO), stores can be reordered with older stores to different locations

In Relaxed Memory Order (RMO), unrelated load operations can also be reordered

Buffered model

Each thread has a buffer

If an instruction causes a shadow action then the shadow is added to the buffer

An action in the buffer which is independent of all of the older actions in the buffer can be scheduled

A read instruction reads its value from the latest corresponding write action in the buffer or from memory

An architecture specifies which instructions have which shadow actions and what are the independency and ordering relations

Total Store Order

```
shadows a
  | isWrite a = [shadow a]
  | otherwise = []

ord (p1, a1) (p2, a2)
  | p1 < p2   = True
  | p1 == p2  = label a1 < label a2
  | otherwise = False

dep (p1, a1) (p2, a2)
  | p1 == p2  = isShadow a1 == isShadow a2
                || label a1 == label a2
  | otherwise = isGlobal a1 && isGlobal a2
                && (isWrite a1 || isWrite a2)
                && var a1 == var a2
```

Our example with TSO

Init: `x = 0; y = 0;`	
P_1	P_2
(a) `[x] := 1`	(c) `[y] := 1`
(b) `r1 := [y]`	(d) `r2 := [x]`
Observed? `r1 = 0; r2 = 0;`	

Instructions a and c generate shadow actions a' and c'

$D = \{(a, a'), (a, b), (c, c'), (c, d), (b, c'), (a', d)\}$

The Foata normal forms are the following:

$$(ac)(bd)(a'c') \quad (ac)(a'b)(c'd)$$
$$(ac)(a'c')(bd) \quad (ac)(c'd)(a'b)$$

Our example with TSO

Init: `x = 0; y = 0;`	
P_1	P_2
(a) `[x] := 1`	(c) `[y] := 1`
(b) `r1 := [y]`	(d) `r2 := [x]`
Observed? `r1 = 0; r2 = 0;`	

Instructions a and c generate shadow actions a' and c'

$D = \{(a, a'), (a, b), (c, c'), (c, d), (b, c'), (a', d)\}$

The Foata normal forms are the following:

$$(ac)(bd)(a'c') \quad (ac)(a'b)(c'd)$$
$$(ac)(a'c')(bd) \quad (ac)(c'd)(a'b)$$

Conclusion

With tools from trace theory we can generate only the representative excutions of a program

This may reduce the amount of work required for verification

The approach can also be applied to some relaxed memory architectures such as TSO, PSO and RMO

Thank you!

This work was supported by the IT Academy PhD scholarship