Fully-Implicit Relational Coarsest Partitioning for Faster Bisimulation

(As Preparation for Fully-Implicit Lumping)

Department of Computer Science
University of California at Riverside

Project Presentation for MS CS Oral Examination

Outline

Abstract

The present work applies interleaved MDD partition representation to the bisimulation problem. We have implemented these techniques in the context of the SMART verification tool. We compare the execution time and memory consumption of our fully-implicit methods (using interleaved MDDs) with the execution time and memory consumption of partially-implicit methods, as applied to the same bisimulation problems. We found that the fully implicit method had surprisingly poor speed performance, especially for models with few variables with many values. The performance of the fully implicit method was reasonable for models with many variables having few values, and there are hints that the fully implicit method will be an improvement with much larger models.

Outline

Outline

Definition of Bisimulation

R is a bisimulation of colored, labeled FSA: $\langle S, C, T \rangle \mid$
$C \in S \to \mathit{color} \wedge T \subseteq S \times \mathit{label} \times S$, iff:
$R \subseteq S \times S \wedge \forall \langle s_1, s_2 \rangle \in R : [\ C(s_1) = C(s_2) \wedge (\forall \langle s, l, s_1' \rangle \in T :$
$s = s_1 \implies \exists s_2' \in S : T(s_2, l, s_2') \wedge R(s_1', s_2')) \wedge (\forall \langle s, l, s_2' \rangle \in T :$
$s = s_2 \implies \exists s_1' \in S : T(s_1, l, s_1') \wedge R(s_1', s_2'))\ ]$

Original Definition of Bisimulation (Milner 1989)

4.2 Strong bisimulation

The above discussion leads us to consider an equivalence relation with the following property:

> P and Q are equivalent iff, for every action α, every α-derivative of P is equivalent to some α-derivative of Q, and conversely.

...

Definition 1 A binary relation $\mathcal{S} \subseteq \mathcal{P} \times \mathcal{P}$ over agents is a *strong bisimulation* if $(P, Q) \in \mathcal{S}$ implies, for all $\alpha \in Act$,

(i) Whenever $P \xrightarrow{\alpha} P'$ then, for some Q', $Q \xrightarrow{\alpha} Q'$ and $(P', Q') \in \mathcal{S}$
(ii) Whenever $Q \xrightarrow{\alpha} Q'$ then, for some P', $P \xrightarrow{\alpha} P'$ and $(P', Q') \in \mathcal{S}$ ■

Why Bisimulation?

Bisimulation is . . .

- A special case of Lumping
  (A minimization problem for Markov systems) to simplify subsequent numeric computations
- An extensional notion of equivalence of states (FSA)

Notation:

- $R \subseteq S \times S$ — A relation between states
- $R(s_1, s_2)$ or $\langle s_1, s_2 \rangle \in R$ — "s_1 and s_2 are bisimilar"
- "$\sim$" — The Largest Bisimulation

A Bisimulation is ...

Definition

- (Given a colored, labeled transition system,(st,col,tran) $\langle S, C, T\rangle \mid C \in S \rightarrow \mathit{color} \wedge T \subseteq S \times \mathit{label} \times S$,
- A Bisimulation R is a 2-ary relation on S where: $R \subseteq S \times S \wedge$
- Each pair in R has the same color, $\forall \langle s_1, s_2\rangle \in R : C(s_1) = C(s_2) \wedge$
- And has matching transitions to pairs in R $\forall \langle s, l, s_1'\rangle \in T : s = s_1 \implies \exists s_2' \in S : T(s_2, l, s_2') \wedge R(s_1', s_2')$

$$\wedge$$

$$\forall \langle s, l, s_2'\rangle \in T : s = s_2 \implies \exists s_1' \in S : T(s_1, l, s_1') \wedge R(s_1', s_2')$$

A Bisimulation is ...

Definition

- (Given a colored, labeled transition system,(st,col,tran) $\langle S, C, T\rangle \mid C \in S \rightarrow color \wedge T \subseteq S \times label \times S$,
- A Bisimulation R is a 2-ary relation on S where: $R \subseteq S \times S \wedge$
- Each pair in R has the same color, $\forall \langle s_1, s_2\rangle \in R : C(s_1) = C(s_2) \wedge$
- And has matching transitions to pairs in R $\forall \langle s, l, s'_1\rangle \in T : s = s_1 \implies \exists s'_2 \in S : T(s_2, l, s'_2) \wedge R(s'_1, s'_2)$

$$\wedge$$

$$\forall \langle s, l, s'_2\rangle \in T : s = s_2 \implies \exists s'_1 \in S : T(s_1, l, s'_1) \wedge R(s'_1, s'_2)$$

A Bisimulation is ...

Definition

- (Given a colored, labeled transition system,(st,col,tran) $\langle S, C, T\rangle \mid C \in S \rightarrow color \wedge T \subseteq S \times label \times S$,
- A Bisimulation R is a 2-ary relation on S where: $R \subseteq S \times S \wedge$
- Each pair in R has the same color, $\forall\langle s_1, s_2\rangle \in R : C(s_1) = C(s_2) \wedge$
- And has matching transitions to pairs in R $\forall\langle s, l, s_1'\rangle \in T : s = s_1 \implies \exists s_2' \in S : T(s_2, l, s_2') \wedge R(s_1', s_2')$

$$\wedge$$

$$\forall\langle s, l, s_2'\rangle \in T : s = s_2 \implies \exists s_1' \in S : T(s_1, l, s_1') \wedge R(s_1', s_2')$$

A Bisimulation is ...

Definition

- (Given a colored, labeled transition system,(st,col,tran) $\langle S, C, T\rangle \mid C \in S \rightarrow \mathit{color} \wedge T \subseteq S \times \mathit{label} \times S$,
- A Bisimulation R is a 2-ary relation on S where: $R \subseteq S \times S \wedge$
- Each pair in R has the same color, $\forall\langle s_1, s_2\rangle \in R : C(s_1) = C(s_2) \wedge$
- And has matching transitions to pairs in R $\forall\langle s, l, s_1'\rangle \in T : s = s_1 \implies \exists s_2' \in S : T(s_2, l, s_2') \wedge R(s_1', s_2')$

$$\wedge$$

$$\forall\langle s, l, s_2'\rangle \in T : s = s_2 \implies \exists s_1' \in S : T(s_1, l, s_1') \wedge R(s_1', s_2')$$

A Bisimulation is ...

Definition

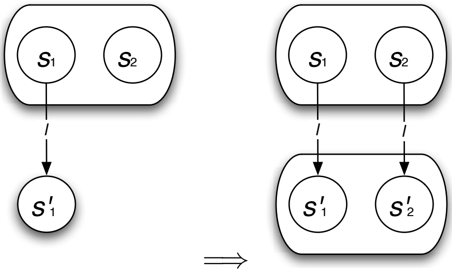
- (Given a colored, labeled transition system,(st,col,tran) $\langle S, C, T\rangle \mid C \in S \rightarrow color \wedge T \subseteq S \times label \times S$,
- A Bisimulation R is a 2-ary relation on S where: $R \subseteq S \times S \wedge$
- Each pair in R has the same color, $\forall\langle s_1, s_2\rangle \in R : C(s_1) = C(s_2) \wedge$
- And has matching transitions to pairs in R $\forall\langle s, l, s_1'\rangle \in T : s = s_1 \implies \exists s_2' \in S : T(s_2, l, s_2') \wedge R(s_1', s_2')$

$$\wedge$$

$$\forall\langle s, l, s_2'\rangle \in T : s = s_2 \implies \exists s_1' \in S : T(s_1, l, s_1') \wedge R(s_1', s_2')$$

Matching Transitions to Pairs in R.

$$\forall \langle s_1, s_2 \rangle \in R :$$
$$\forall \langle s, l, s_1' \rangle \in T : s = s_1 \implies \exists s_2' \in S : T(s_2, l, s_2') \wedge R(s_1', s_2')$$



The (Largest) Bisimulation is ...

Definition

The Largest Bisimulation, "$\sim$" is the union of all bisimulations R

And is an equivalence relation.

Original Definition of "$\sim$" (Milner 1989)

Definition 2 P and Q are *strongly equivalent* or *strongly bisimilar*, written $P \sim Q$, if $(P, Q) \in \mathcal{S}$ for some strong bisimulation $\mathcal{S}$. This may be equivalently expressed as follows:

$$\sim \;=\; \bigcup\{\mathcal{S} \;:\; \mathcal{S} \text{ is a strong bisimulation}\} \qquad \blacksquare$$

Proposition 2

(1) $\sim$ is the largest strong bisimulation.
(2) $\sim$ is an equivalence relation.

Relational Coarsest Partition = Largest Bisimulation.

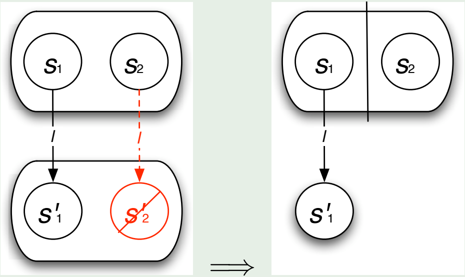
Generic iterative *splitting* algorithm:

- Iterative update of some equivalence relation variable R.
- Start with $R =$ coarsest partition of state space S, $S \times S$ $(\sim \subseteq R)$
- Initially *split* R based on state color
- Iteratively remove implausible members from R when required by definition of Bisimulation, by splitting R into smaller blocks B_*.
  $\forall \langle s, l, s_1' \rangle \in T : s = s_1 \implies \exists s_2' \in S : T(s_2, l, s_2') \wedge R(s_1', s_2')$
- Iteration continues until all blocks have been used as splitters (inherited stability, block unions).
- May iterate over transition labels. Algorithm cores are often described without reference to labeling.

Splitting.

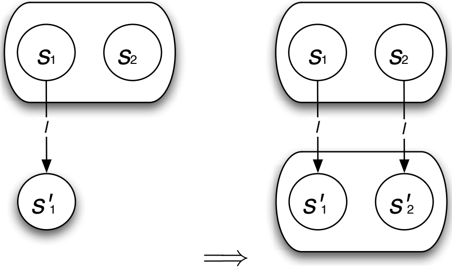
$$\forall \langle s_1, s_2 \rangle \in R :$$
$$\forall \langle s, l, s_1' \rangle \in T : s = s_1 \implies \exists s_2' \in S : T(s_2, l, s_2') \wedge R(s_1', s_2')$$

Example



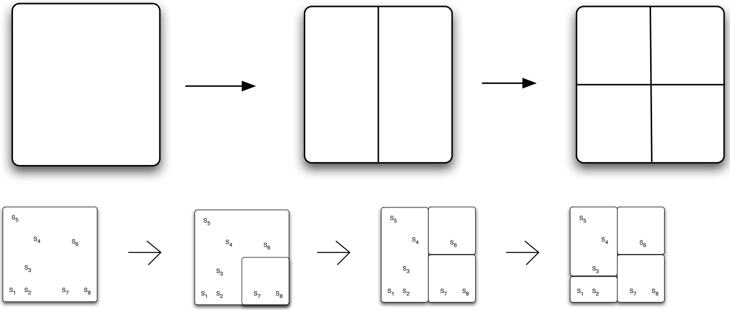
Matching Transitions to Pairs in R.

$$\forall \langle s_1, s_2 \rangle \in R :$$
$$\forall \langle s, l, s_1' \rangle \in T : s = s_1 \implies \exists s_2' \in S : T(s_2, l, s_2') \wedge R(s_1', s_2')$$



Outline

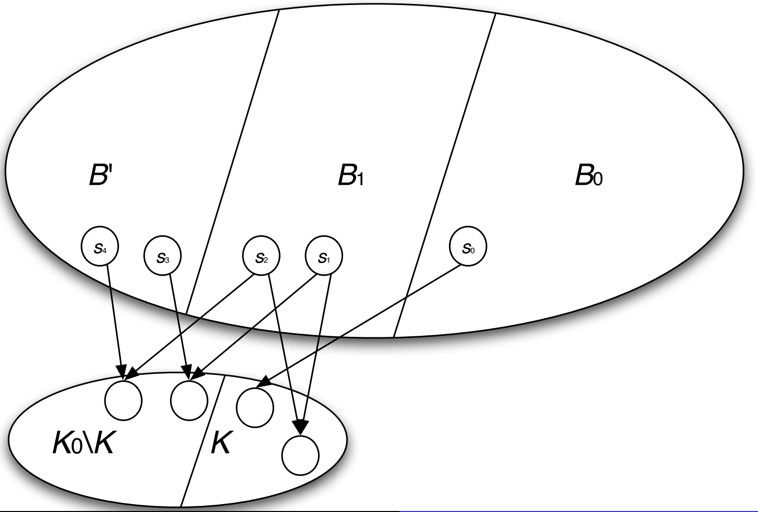
Splitting produces hierarchy of partition blocks



"Process The Smaller Half." $O(m \log n)$

- Start with $R =$ coarsest partition of state space S, $S \times S$
- First split uses S as splitter. Separates states with no transitions.
- Remember hierarchy of split blocks for use as splitters
- Use 2 splitters K and $K_0 \setminus K$, where K_0 was already a splitter.
- Iteratively split blocks B into smaller blocks B_0, B_1, and B'
- Maintain reverse adjacency lists
- Maintain counts of edges from states to states in splitter blocks

"Process The Smaller Half." $O(m \log n)$

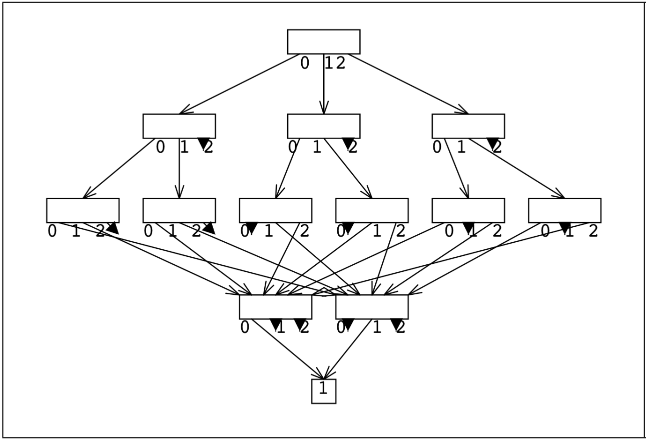


"Process The Smaller Half." $O(m \log n)$

- Uses edge counts to distinguish between members of B_0 and B_1.
- Avoids processing members of B' and $K_0 \setminus K$ (by reusing structures).
- Update edge counts.
- Each state s occurs in at most $\log n$ splitters.
- Each edge participates in at most $O(\log n)$ splitting operations
- $T = O(m \log n)$

Outline

"Symbolic Methods" $\neq$ Mathematica ®$_{(\mathit{WolframResearch})}$

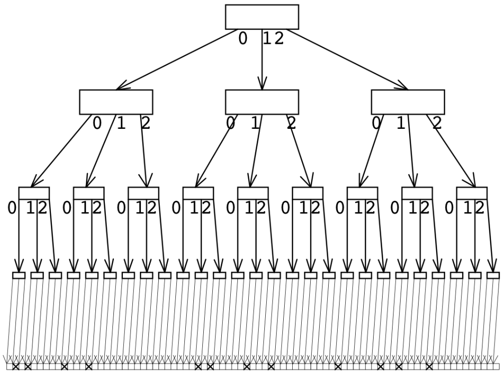


"0001"
"0010"
"0100"
"0111"
"1011"
"1020"
"1110"
"1121"
"2000"
"2021"
"2101"
"2120"

Multi-Way Decision Diagrams Represent Relations

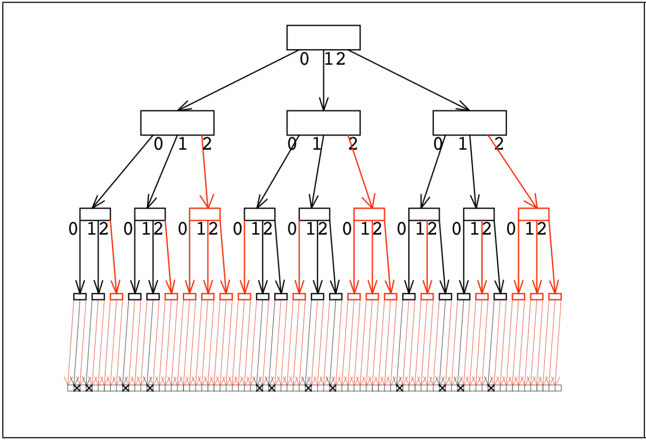
- Each path in MDD (graph) corresponds to tuple in relation.
- Canonical: sharing $\leftrightarrow$ compression, comparison, *unique* table, non-mutable.
- Efficient memoized recursive algorithms for set operations: ($\in$ (not memoized), $|()|$, $\cup$, $\cap$, $\setminus$, $\subseteq$).
- Efficient memoized recursive algorithms for functional operations: ($\circ$, $\exists$, $\forall$).
- Set operations implemented in SMART MDD library.
- SMART Saturation algorithm for transitive closure (state space exploration).
- "Quasi-reduced", with "NULL" edges
- Variable ordering matters.

Set = Boolean Table ($\hat{S} = [1,3]^4$)



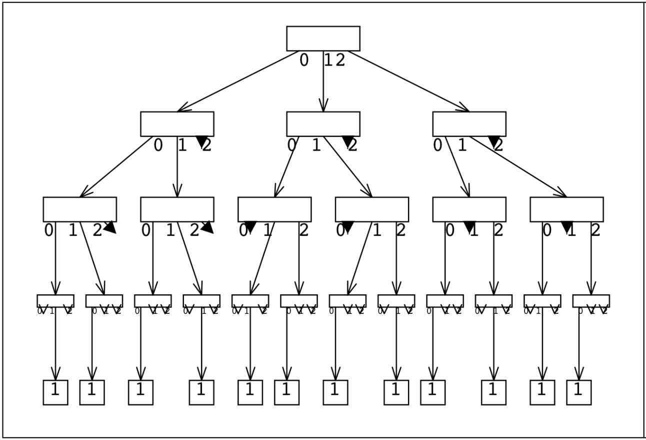
"0001"
"0010"
"0100"
"0111"
"1011"
"1020"
"1110"
"1121"
"2000"
"2021"
"2101"
"2120"

Empty Subsets



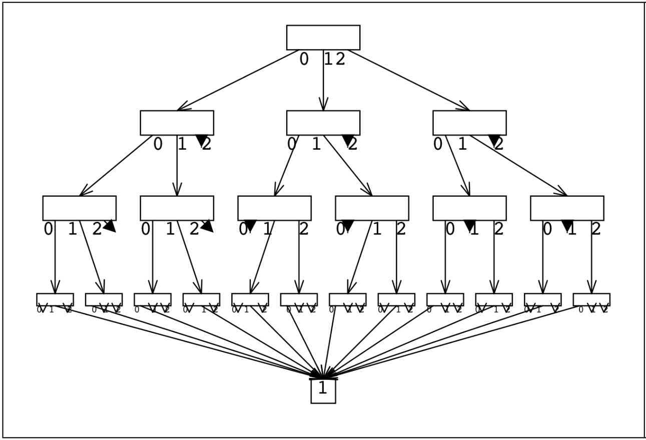
“0001”
“0010”
“0100”
“0111”
“1011”
“1020”
“1110”
“1121”
“2000”
“2021”
“2101”
“2120”

Replace with “NULL” Edges



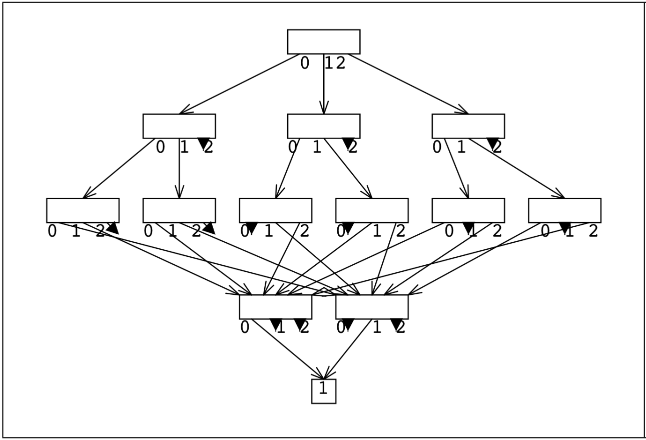
“0001”
“0010”
“0100”
“0111”
“1011”
“1020”
“1110”
“1121”
“2000”
“2021”
“2101”
“2120”

Quasi-Reduce at Leaf Level



“0001”
“0010”
“0100”
“0111”
“1011”
“1020”
“1110”
“1121”
“2000”
“2021”
“2101”
“2120”

Quasi-Reduced MDD



"0001"
"0010"
"0100"
"0111"
"1011"
"1020"
"1110"
"1121"
"2000"
"2021"
"2101"
"2120"

Memoized Recursive Algorithm for Set Difference ("\")

Algorithm $\mathcal{R} \leftarrow \mathcal{X} \setminus \mathcal{Y}$

1. Handle a few special cases before checking cache:
   1. If $\mathcal{X} = \emptyset$ then return with $\mathcal{R} \leftarrow \emptyset$
   2. If $\mathcal{Y} = \emptyset$ then return with $\mathcal{R} \leftarrow \mathcal{X}$
   3. If $\mathcal{X} = \mathcal{Y}$ then return with $\mathcal{R} \leftarrow \emptyset$
2. If the cache has $\mathcal{X} \setminus \mathcal{Y}$ then return with $\mathcal{R} \leftarrow$ cached value
3. Construct new MDD node $\mathcal{R}$ as follows:
4. Recursively call: $\mathcal{R}_i \leftarrow \mathcal{X}_i \setminus \mathcal{Y}_i$, for each variable value i
5. If $\forall i : \mathcal{R}_i = \emptyset$ then $\mathcal{R} \leftarrow \emptyset$
6. Make $\mathcal{R}$ canonical: $\mathcal{R} \leftarrow \mathit{unique}(\mathcal{R})$
7. Put $\mathcal{R} = \mathcal{X} \setminus \mathcal{Y}$ into the cache
8. Return $\mathcal{R}$

Memoized Recursive Algorithm for Set Difference ("\")

Algorithm $\mathcal{R} \leftarrow \mathcal{X} \setminus \mathcal{Y}$

1. Handle a few special cases before checking cache:
    1. If $\mathcal{X} = \emptyset$ then return with $\mathcal{R} \leftarrow \emptyset$
    2. If $\mathcal{Y} = \emptyset$ then return with $\mathcal{R} \leftarrow \mathcal{X}$
    3. If $\mathcal{X} = \mathcal{Y}$ then return with $\mathcal{R} \leftarrow \emptyset$
2. If the cache has $\mathcal{X} \setminus \mathcal{Y}$ then return with $\mathcal{R} \leftarrow$ cached value
3. Construct new MDD node $\mathcal{R}$ as follows:
4. Recursively call: $\mathcal{R}_i \leftarrow \mathcal{X}_i \setminus \mathcal{Y}_i$, for each variable value i
5. If $\forall i : \mathcal{R}_i = \emptyset$ then $\mathcal{R} \leftarrow \emptyset$
6. Make $\mathcal{R}$ canonical: $\mathcal{R} \leftarrow \mathit{unique}(\mathcal{R})$
7. Put $\mathcal{R} = \mathcal{X} \setminus \mathcal{Y}$ into the cache
8. Return $\mathcal{R}$

Memoized Recursive Algorithm for Set Difference ("\")

Algorithm $\mathcal{R} \leftarrow \mathcal{X} \setminus \mathcal{Y}$

1. Handle a few special cases before checking cache:
   1. If $\mathcal{X} = \emptyset$ then return with $\mathcal{R} \leftarrow \emptyset$
   2. If $\mathcal{Y} = \emptyset$ then return with $\mathcal{R} \leftarrow \mathcal{X}$
   3. If $\mathcal{X} = \mathcal{Y}$ then return with $\mathcal{R} \leftarrow \emptyset$
2. If the cache has $\mathcal{X} \setminus \mathcal{Y}$ then return with $\mathcal{R} \leftarrow$ cached value
3. Construct new MDD node $\mathcal{R}$ as follows:
4. Recursively call: $\mathcal{R}_i \leftarrow \mathcal{X}_i \setminus \mathcal{Y}_i$, for each variable value i
5. If $\forall i : \mathcal{R}_i = \emptyset$ then $\mathcal{R} \leftarrow \emptyset$
6. Make $\mathcal{R}$ canonical: $\mathcal{R} \leftarrow \mathit{unique}(\mathcal{R})$
7. Put $\mathcal{R} = \mathcal{X} \setminus \mathcal{Y}$ into the cache
8. Return $\mathcal{R}$

Memoized Recursive Algorithm for Set Difference ("\")

Algorithm $\mathcal{R} \leftarrow \mathcal{X} \setminus \mathcal{Y}$

1. Handle a few special cases before checking cache:
   1. If $\mathcal{X} = \emptyset$ then return with $\mathcal{R} \leftarrow \emptyset$
   2. If $\mathcal{Y} = \emptyset$ then return with $\mathcal{R} \leftarrow \mathcal{X}$
   3. If $\mathcal{X} = \mathcal{Y}$ then return with $\mathcal{R} \leftarrow \emptyset$
2. If the cache has $\mathcal{X} \setminus \mathcal{Y}$ then return with $\mathcal{R} \leftarrow$ cached value
3. Construct new MDD node $\mathcal{R}$ as follows:
4. Recursively call: $\mathcal{R}_i \leftarrow \mathcal{X}_i \setminus \mathcal{Y}_i$, for each variable value i
5. If $\forall i : \mathcal{R}_i = \emptyset$ then $\mathcal{R} \leftarrow \emptyset$
6. Make $\mathcal{R}$ canonical: $\mathcal{R} \leftarrow \mathit{unique}(\mathcal{R})$
7. Put $\mathcal{R} = \mathcal{X} \setminus \mathcal{Y}$ into the cache
8. Return $\mathcal{R}$

Memoized Recursive Algorithm for Set Difference ("\")

Algorithm $\mathcal{R} \leftarrow \mathcal{X} \setminus \mathcal{Y}$

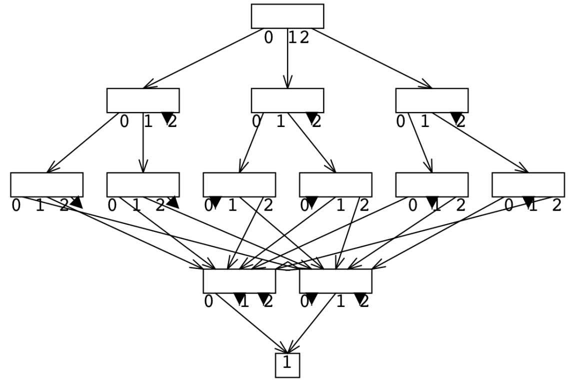
1. Handle a few special cases before checking cache:
   1. If $\mathcal{X} = \emptyset$ then return with $\mathcal{R} \leftarrow \emptyset$
   2. If $\mathcal{Y} = \emptyset$ then return with $\mathcal{R} \leftarrow \mathcal{X}$
   3. If $\mathcal{X} = \mathcal{Y}$ then return with $\mathcal{R} \leftarrow \emptyset$
2. If the cache has $\mathcal{X} \setminus \mathcal{Y}$ then return with $\mathcal{R} \leftarrow$ cached value
3. Construct new MDD node $\mathcal{R}$ as follows:
4. Recursively call: $\mathcal{R}_i \leftarrow \mathcal{X}_i \setminus \mathcal{Y}_i$, for each variable value i
5. If $\forall i : \mathcal{R}_i = \emptyset$ then $\mathcal{R} \leftarrow \emptyset$
6. Make $\mathcal{R}$ canonical: $\mathcal{R} \leftarrow \mathit{unique}(\mathcal{R})$
7. Put $\mathcal{R} = \mathcal{X} \setminus \mathcal{Y}$ into the cache
8. Return $\mathcal{R}$

Memoized Recursive Algorithm for Set Difference ("\")

Algorithm $\mathcal{R} \leftarrow \mathcal{X} \setminus \mathcal{Y}$

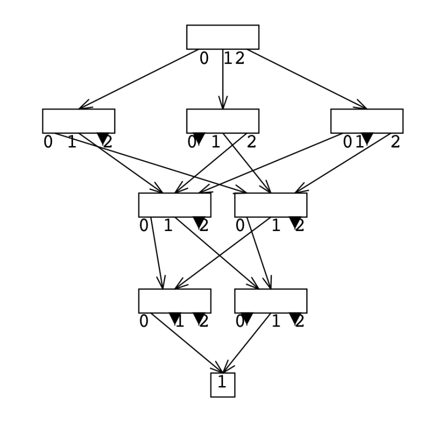
1. Handle a few special cases before checking cache:
   1. If $\mathcal{X} = \emptyset$ then return with $\mathcal{R} \leftarrow \emptyset$
   2. If $\mathcal{Y} = \emptyset$ then return with $\mathcal{R} \leftarrow \mathcal{X}$
   3. If $\mathcal{X} = \mathcal{Y}$ then return with $\mathcal{R} \leftarrow \emptyset$
2. If the cache has $\mathcal{X} \setminus \mathcal{Y}$ then return with $\mathcal{R} \leftarrow$ cached value
3. Construct new MDD node $\mathcal{R}$ as follows:
4. Recursively call: $\mathcal{R}_i \leftarrow \mathcal{X}_i \setminus \mathcal{Y}_i$, for each variable value i
5. If $\forall i : \mathcal{R}_i = \emptyset$ then $\mathcal{R} \leftarrow \emptyset$
6. Make $\mathcal{R}$ canonical: $\mathcal{R} \leftarrow \mathit{unique}(\mathcal{R})$
7. Put $\mathcal{R} = \mathcal{X} \setminus \mathcal{Y}$ into the cache
8. Return $\mathcal{R}$

Variable Ordering Matters (1)



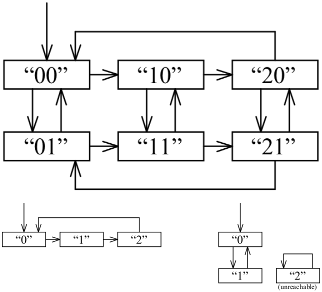
"0001"
"0010"
"0100"
"0111"
"1011"
"1020"
"1110"
"1121"
"2000"
"2021"
"2101"
"2120"

Variable Ordering Matters (2)



"0001"	"0001"
"0100"	"0010"
"0010"	"0100"
"0111"	"0111"
"1101"	"1011"
"1200"	"1020"
"1110"	"1110"
"1211"	"1121"
"2000"	"2000"
"2201"	"2021"
"2011"	"2101"
"2210"	"2120"

Represent FSAs as Relations (and MDDs)



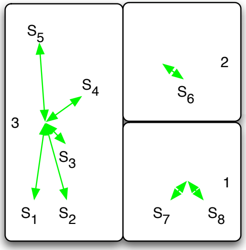
⟨ "00" → "01" ⟩	"0001"
⟨ "00" → "10" ⟩	"0010"
⟨ "01" → "00" ⟩	"0100"
⟨ "01" → "11" ⟩	"0111"
⟨ "10" → "11" ⟩	"1011"
⟨ "10" → "20" ⟩	"1020"
⟨ "11" → "10" ⟩	"1110"
⟨ "11" → "21" ⟩	"1121"
⟨ "20" → "00" ⟩	"2000"
⟨ "20" → "21" ⟩	"2021"
⟨ "21" → "01" ⟩	"2101"
⟨ "21" → "20" ⟩	"2120"

Represent FSAs as Relations (and MDDs)

- Each state variable corresponds to a (set of) variables in tuple.
- Each transition in FSA corresponds to tuple in transition relation.
- Interleaved ordering of variables of source and destination states of transition relation usually yields relatively compact MDDs.
- SMART$_2$ produces MDDs of transition relations in interleaved form.

Alternate Ways to Represent Partitions as Relations

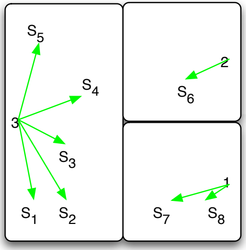
1. Equivalence Relation:
   $\langle s_1, s_2 \rangle \mid s_1, s_2 \in S$
2. List of Partition Blocks
   $B_1, B_2, B_3, B_4, \ldots \mid B_* \subseteq S$
3. Block Numbering
   $\langle s, n \rangle \mid s \in S, n \in \mathbb{N}$



1

Alternate Ways to Represent Partitions as Relations

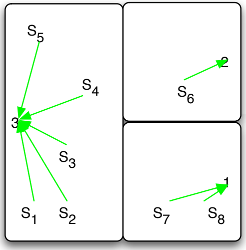
1. Equivalence Relation: $\langle s_1, s_2 \rangle \mid s_1, s_2 \in S$
2. List of Partition Blocks $B_1, B_2, B_3, B_4, \ldots \mid B_* \subseteq S$
3. Block Numbering $\langle s, n \rangle \mid s \in S, n \in \mathbb{N}$



2

Alternate Ways to Represent Partitions as Relations

1. Equivalence Relation: $\langle s_1, s_2 \rangle \mid s_1, s_2 \in S$
2. List of Partition Blocks $B_1, B_2, B_3, B_4, \ldots \mid B_* \subseteq S$
3. Block Numbering $\langle s, n \rangle \mid s \in S, n \in \mathbb{N}$



3

Ways to Represent Partitions as MDDs

1. Equivalence Relation (Non-Interleaved) (link)
   - $\langle s_1, s_2 \rangle \mid s_1, s_2 \in S$
   - Variable ordering: $x_1, x_2, x_3, \ldots, y_1, y_2, y_3, \ldots$
2. Equivalence Relation (Interleaved) (link)
   - $\langle s_1, s_2 \rangle \mid s_1, s_2 \in S$
   - Variable ordering: $x_1, y_1, x_2, y_2, x_3, y_3, \ldots$
3. Lists of Partition Blocks (link)
   - $B_1, B_2, B_3, B_4, \ldots \mid B_* \subseteq S$
   - Variable ordering: $x_1, x_2, x_3, \ldots$
4. Block Numbering/function of state (link)
   - $\langle s, n \rangle \mid s \in S, n \in \mathbb{N}$
   - Variable ordering: $x_1, x_2, x_3, \ldots k_1, k_2, k_3, \ldots$

Ways to Represent Partitions as MDDs

1. Equivalence Relation (Non-Interleaved) (link)
   - $\langle s_1, s_2 \rangle \mid s_1, s_2 \in S$
   - Variable ordering: $x_1, x_2, x_3, \ldots, y_1, y_2, y_3, \ldots$
2. Equivalence Relation (Interleaved) (link)
   - $\langle s_1, s_2 \rangle \mid s_1, s_2 \in S$
   - Variable ordering: $x_1, y_1, x_2, y_2, x_3, y_3, \ldots$
3. Lists of Partition Blocks (link)
   - $B_1, B_2, B_3, B_4, \ldots \mid B_* \subseteq S$
   - Variable ordering: $x_1, x_2, x_3, \ldots$
4. Block Numbering/function of state (link)
   - $\langle s, n \rangle \mid s \in S, n \in \mathbb{N}$
   - Variable ordering: $x_1, x_2, x_3, \ldots k_1, k_2, k_3, \ldots$

Ways to Represent Partitions as MDDs

1. Equivalence Relation (Non-Interleaved) (link)
   - $\langle s_1, s_2 \rangle \mid s_1, s_2 \in S$
   - Variable ordering: $x_1, x_2, x_3, \ldots, y_1, y_2, y_3, \ldots$
2. Equivalence Relation (Interleaved) (link)
   - $\langle s_1, s_2 \rangle \mid s_1, s_2 \in S$
   - Variable ordering: $x_1, y_1, x_2, y_2, x_3, y_3, \ldots$
3. Lists of Partition Blocks (link)
   - $B_1, B_2, B_3, B_4, \ldots \mid B_* \subseteq S$
   - Variable ordering: $x_1, x_2, x_3, \ldots$
4. Block Numbering/function of state (link)
   - $\langle s, n \rangle \mid s \in S, n \in \mathbb{N}$
   - Variable ordering: $x_1, x_2, x_3, \ldots k_1, k_2, k_3, \ldots$

Ways to Represent Partitions as MDDs

1. Equivalence Relation (Non-Interleaved) (link)
   - $\langle s_1, s_2 \rangle \mid s_1, s_2 \in S$
   - Variable ordering: $x_1, x_2, x_3, \ldots, y_1, y_2, y_3, \ldots$
2. Equivalence Relation (Interleaved) (link)
   - $\langle s_1, s_2 \rangle \mid s_1, s_2 \in S$
   - Variable ordering: $x_1, y_1, x_2, y_2, x_3, y_3, \ldots$
3. Lists of Partition Blocks (link)
   - $B_1, B_2, B_3, B_4, \ldots \mid B_* \subseteq S$
   - Variable ordering: $x_1, x_2, x_3, \ldots$
4. Block Numbering/function of state (link)
   - $\langle s, n \rangle \mid s \in S, n \in \mathbb{N}$
   - Variable ordering: $x_1, x_2, x_3, \ldots k_1, k_2, k_3, \ldots$

Outline

Generic Signature-Based Splitting Algorithm

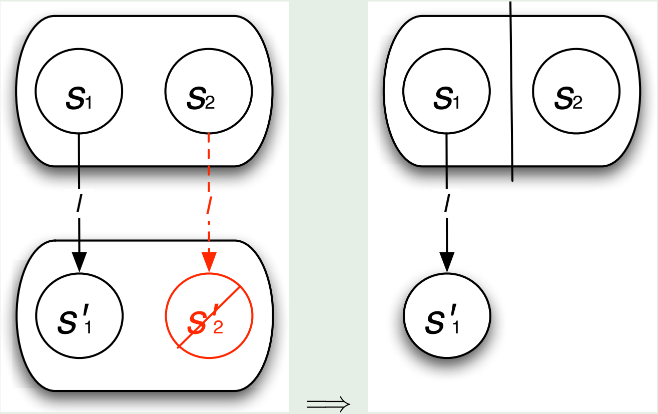
- Split each partition block using all blocks as splitters.
- State Space: S, Partition: $P \in S \rightarrow Block$, Transition: $Q \subseteq S \times S$, Signature: T
- Signature of a state s includes set of partition blocks to which s has transitions.
- Signature includes current partition block where s resides.
- Signature often described without edge labeling.
- Define new partition of S, with a block for each signature.

Algorithm: Signature-Based Splitting

1. Signature: $T(s) = \langle$ P(s), $\{P(s')|\langle s, s'\rangle \in Q\}$ $\rangle$.
2. New Partition: $P'(s) = f(T(s))$ (for some bijection f)
3. Repeat 1;2;$P \leftarrow P'$ until $P = P'$

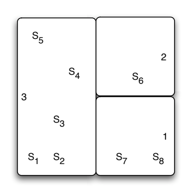
Splitting.

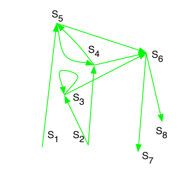
Example

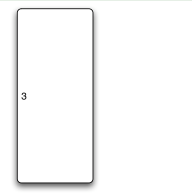


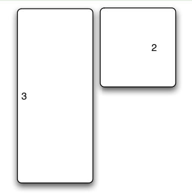
Generic Signature-Based Splitting Algorithm

Example

partition (P): 

transitions (Q): 

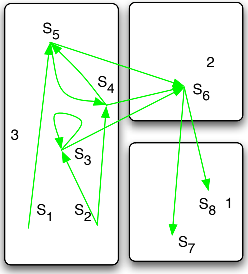
$T(\ s_4\ ) = \langle$ 

$, \{$ 

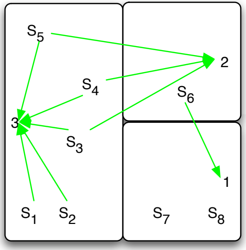
$\}\rangle$

Generic Signature-Based Splitting Algorithm

P, Q:

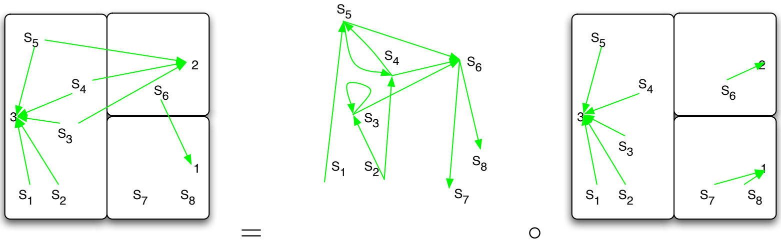


, T:

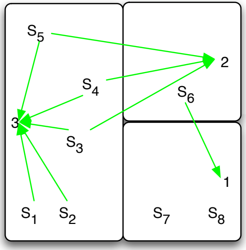


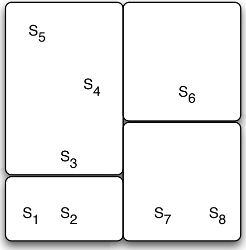
Generic Signature-Based Splitting Algorithm

$$T = Q \circ P$$



Generic Signature-Based Splitting Algorithm

T: 

, P': 

Generic Signature-Based Splitting Algorithm

- Split each partition block using all blocks as splitters.
- State Space: S, Partition: $P \in S \rightarrow Block$, Transition: $Q \subseteq S \times S$, Signature: T
- Signature of a state s includes set of partition blocks to which s has transitions.
- Signature includes current partition block where s resides.
- Signature often described without edge labeling.
- Define new partition of S, with a block for each signature.

Algorithm: Signature-Based Splitting

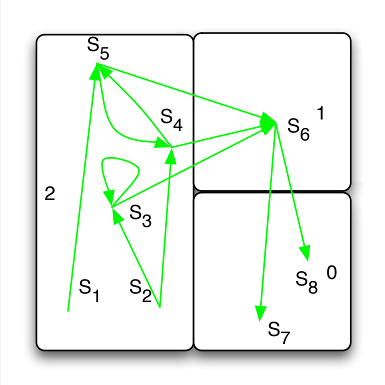
1. Signature: $T(s) = \langle$ P(s), $\{P(s')|\langle s, s'\rangle \in Q\}$ $\rangle$.
2. New Partition: $P'(s) = f(T(s))$ (for some bijection f)
3. Repeat 1;2;$P \leftarrow P'$ until $P = P'$

Algorithm: Rank-Based Initial Partition

- Agostino Dovier, Carla Piazza, and Alberto Policriti (2004).
- Linear symbolic steps.
- Produces *rank*-based partition
- Partition representation: lists of partition blocks
- Needs other block splitting algorithm to finish.
- Apply other algorithm to blocks in rank order.
- Strongly connected components cause problems.
- Extract rank-1 elements: $R_1 \leftarrow S \setminus preimage(S)$

Algorithm: Rank-Based Initial Partition

Example



Algorithm: Forwarding, Splitting, Ordering

- Ralf Wimmer, Marc Herbstritt, and Bernd Becker (2007).
- Partition representation: lists of blocks AND numbering function
- Algorithm maintains signature and partition.
- Forwarding: Immediately update partition numbering function.
- Split-drive refinement: Only attempt splitting on blocks that might be split.
- Block ordering: Split blocks that might propagate splitting most.

History

1. Review lumping algorithms.
2. Contrive new ideas.
   - Interleaved partition representation (Ciardo)
   - Depth-based initial partition (Mumme)
3. Limit scope to bisimulation instead of lumping.
4. Devise new algorithms
   - Saturation for distance calculation (Ciardo)
   - Relational operations for interleaved partition refinement (Mumme)
5. Implement interleaved partition refinement for bisimulation.
6. Review bisimulation algorithms.
7. Amar Bouali and Robert De Simone (1992).
8. Implement hybrid algorithm to compare partition representations.

Outline

Symbolic Bisimulation Minimization

- Amar Bouali and Robert De Simone (1992).
- Partition representation: Equivalence relation (interleaved or non-interleaved)
- Transition representation: Relation (interleaved or non-interleaved (respectively))
- Similar to generic signature-based splitting algorithm.

Our Implementation of Bouali and De Simone's Algorithm

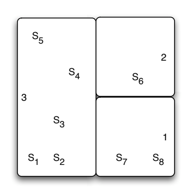
- Partition representation: Equivalence relation (interleaved)
- Transition representation: Relation (interleaved)
- Similar to generic signature-based splitting algorithm, except:
- Equivalence relation allows signature without current partition number.

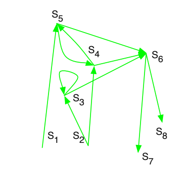
Algorithm 1 Signature Formula

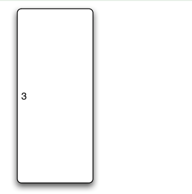
- S — State space
- $E \subseteq S \times S$ — Equivalence relation
- $Q_{(t)} \subseteq \hat{S} \times \hat{S}$ — Transition relation (for transition t)
- $T \subseteq S \times S = Q \circ E$ — Signatures
- $T(s_1, s_3)$ iff $\exists s_2 \in S : Q(s_1, s_2) \wedge E(s_2, s_3) \wedge S(s_1)$

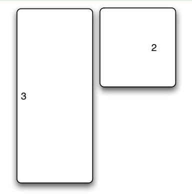
Generic Signature-Based Splitting Algorithm

Example

partition (P): 

transitions (Q): 

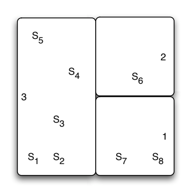
$T(\ s_4\ ) = \langle$ 

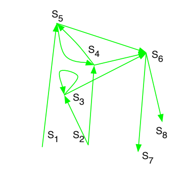
$, \{$ 

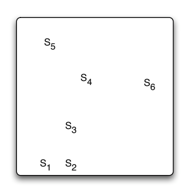
$\}\rangle$

Algorithm 1 Signature

Example

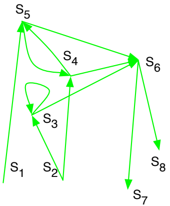
partition (P): 

transitions (Q): 

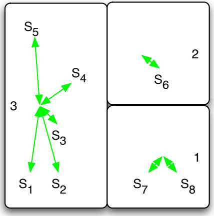
$T(\ s_4\ ) =$ 

Algorithm 1 Signature

$T = Q \circ P$



$=$ $\circ$



Algorithm 1 Signature Calculation

- State space MDD: $\mathcal{S}$
- Interleaved equivalence relation MDD: $\mathcal{E} \subseteq S \times S$
- Interleaved transition relation MDD: $\mathcal{Q} \subseteq \hat{S} \times \hat{S}$
- Signatures MDD: $\mathcal{T} \leftarrow proj_{\vee 3}((DC_2(\mathcal{Q}, \mathcal{S})) \cap (DC_1(\mathcal{E}, \mathcal{S})))$
- $T(s_1, s_3)$ iff $\exists s_2 \in S : Q(s_1, s_2) \wedge E(s_2, s_3) \wedge S(s_1)$

Definitions for Extra Operators

- $DC_1(\mathcal{E},\mathcal{S}) \triangleq \underline{\mathcal{E}}$, where $\underline{\mathcal{E}}(x,y,z) = \mathcal{E}(y,z) \wedge \mathcal{S}(x)$
- $DC_2(\mathcal{Q},\mathcal{S}) \triangleq \underline{\mathcal{Q}}$, where $\underline{\mathcal{Q}}(x,y,z) = \mathcal{Q}(x,z) \wedge \mathcal{S}(y)$
- $proj_{\vee 3}(\mathcal{F}) \triangleq \mathcal{F}'$, where $\mathcal{F}'(x,y) = \bigvee c : \mathcal{F}(x,y,c)$
- Signatures MDD: $\mathcal{T} \leftarrow proj_{\vee 3}((DC_2(\mathcal{Q},\mathcal{S})) \cap (DC_1(\mathcal{E},\mathcal{S})))$
- $\mathcal{T}(x,y) \leftarrow \bigvee z : (\underline{\mathcal{Q}}(x,(y),z) \wedge \underline{\mathcal{E}}((x),y,z))$
- $\mathcal{T}(x,y) \leftarrow \bigvee z : (\mathcal{Q}(x,z) \wedge \mathcal{S}(y) \wedge \mathcal{E}(y,z) \wedge \mathcal{S}(x))$
- $\mathcal{T}(s_1,s_3) \leftarrow \bigvee s_2 : (\mathcal{Q}(s_1,s_2) \wedge \mathcal{S}(s_3) \wedge \mathcal{E}(s_3,s_2) \wedge \mathcal{S}(s_1))$
- $\mathcal{T}(s_1,s_3) \leftarrow \bigvee s_2 : (\mathcal{Q}(s_1,s_2) \wedge \mathcal{E}(s_3,s_2) \wedge \mathcal{S}(s_1))$
- $T(s_1,s_3)$ iff $\exists s_2 \in S : Q(s_1,s_2) \wedge E(s_2,s_3) \wedge S(s_1)$

Algorithm 1 Equivalence Relation Formula

- S State space
- $T \subseteq S \times S = Q \circ E$ Signatures
- $\Delta E \subseteq S \times S$ Equivalence relation update
- $\Delta E(s_1, s_3)$ iff $\forall s_2 \in S : T(s_1, s_2) = T(s_3, s_2)$
- $E' \leftarrow E \wedge \Delta E$

Algorithm 1 Equivalence Relation Calculation

- State space MDD: $\mathcal{S}$
- Signatures MDD: $\mathcal{T}$
- $\Delta\mathcal{E} \leftarrow proj_{\wedge 3}(DC_2(\mathcal{T}, \mathcal{S}) \equiv DC_1(\mathcal{T}, \mathcal{S}))$
- $\mathcal{E}' \leftarrow \mathcal{E} \wedge \Delta\mathcal{E}$
- $\Delta E(s_1, s_3)$ iff $\forall s_2 \in S : T(s_1, s_2) = T(s_3, s_2)$

Algorithm 1 Equivalence Relation Calculation

- $\Delta\mathcal{E} \;\leftarrow\; proj_{\wedge 3}(DC_2(\mathcal{T},\mathcal{S}) \equiv DC_1(\mathcal{T},\mathcal{S}))$
- $\mathcal{E}' \leftarrow \mathcal{E} \wedge \Delta\mathcal{E}$
- $\overline{\Delta\mathcal{E}} \;\leftarrow\; proj_{\vee 3}(DC_2(\mathcal{T},\mathcal{S}) \dot\cup DC_1(\mathcal{T},\mathcal{S}))$
- where $x \dot\cup y \triangleq (x \setminus y) \cup (y \setminus x)$
- $\mathcal{E}' \leftarrow \mathcal{E} \setminus \overline{\Delta\mathcal{E}}$

Algorithm 1

Given: Initial partition in variable $\mathcal{E}$, transition relation in $\mathcal{Q}$, state space in $\mathcal{S}$.
Returns final partition in $\mathcal{E}$.

Algorithm: refinement of equivalence relation using signature relation

Repeat:

- $\mathcal{E}_{old} \leftarrow \mathcal{E}$
- $\mathcal{T} \leftarrow proj_{\vee 3}((DC_2(\mathcal{Q}, \mathcal{S})) \cap (DC_1(\mathcal{E}, \mathcal{S})))$
- $\overline{\Delta\mathcal{E}} \leftarrow proj_{\vee 3}(DC_2(\mathcal{T}, \mathcal{S}) \cup DC_1(\mathcal{T}, \mathcal{S}))$
- $\mathcal{E} \leftarrow \mathcal{E} \setminus \overline{\Delta\mathcal{E}}$

Until $\mathcal{E} = \mathcal{E}_{old}$

Algorithm 1 with Transition Labeling

Given: Initial partition in variable $\mathcal{E}$, transition relation in $\mathcal{Q}$, state space in $\mathcal{S}$.
Returns final partition in $\mathcal{E}$.

Algorithm: refinement of equivalence relation using signature relation

Repeat:

- $\mathcal{E}_{old} \leftarrow \mathcal{E}$
- For each $t \in label$ loop:
  - $\mathcal{T} \leftarrow proj_{\vee 3}((DC_2(\mathcal{Q}_t, \mathcal{S})) \cap (DC_1(\mathcal{E}, \mathcal{S})))$
  - $\overline{\Delta\mathcal{E}} \leftarrow proj_{\vee 3}(DC_2(\mathcal{T}, \mathcal{S}) \dot\cup DC_1(\mathcal{T}, \mathcal{S}))$
  - $\mathcal{E} \leftarrow \mathcal{E} \setminus \overline{\Delta\mathcal{E}}$

Until $\mathcal{E} = \mathcal{E}_{old}$

Hybrid Algorithm (for Comparison)

- Partition representation: Block numbering function (non-interleaved)
- Transition representation: Relation (interleaved)
- Similar to generic signature-based splitting algorithm.

Hybrid Algorithm Signature Formula (First Try)

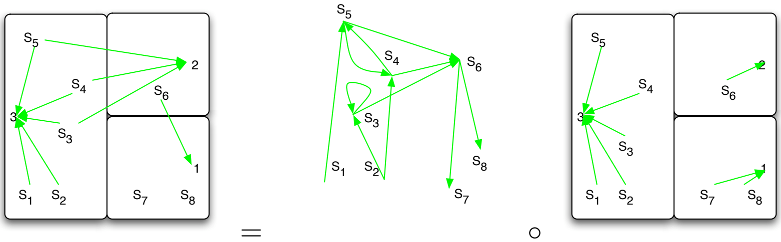
- S — State Space
- $P \subseteq S \times \mathbb{N}^+$ — Partition block number function of state
- $Q \subseteq \hat{S} \times \hat{S}$ — Transition relation
- $T \subseteq S \times \mathbb{N}^+ \times \mathbb{N}^+$ — Signature map state to pairs of blocks
- $T(s) = \bigcup_{s' \in S} \{\langle P(s), P(s')\rangle | \langle s, s'\rangle \in Q\}$. (wrong)
- $T(s, b, b')$ iff $\exists s' \in S : (Q(s, s') \wedge P(s, b) \wedge P(s', b'))$.

Hybrid Algorithm Signature Formula

- S State Space
- $P \subseteq S \times [1, |S|]$ Partition block number function of state
- $Q \subseteq \hat{S} \times \hat{S}$ Transition relation
- $T \subseteq S \times [1, |S|] \times [0, |S|]$ Signature map to pairs of blocks
- $T(s) = \{\langle P(s), 0\rangle\} \cup \bigcup_{s' \in S} \{\langle P(s), P(s')\rangle | \langle s, s'\rangle \in Q\}$.
- $T(s, b, b')$ iff $(P(s, b) \wedge b' = 0) \vee \exists s' \in S :$ $(Q(s, s') \wedge P(s, b) \wedge P(s', b'))$.

Hybrid Algorithm Signature

$$T = Q \circ P$$



Hybrid Algorithm signature Calculation

- State space MDD: $\mathcal{S}$
- Partition block number function MDD: $\mathcal{P} \subseteq S \times [1, |S|]$
- interleaved transition relation MDD: $\mathcal{Q} \subseteq \hat{S} \times \hat{S}$
- Signatures MDD: $\mathcal{T} \leftarrow \mathcal{W} \cup \mathcal{T}_{partial}$, where:
  - $\mathcal{W} = DC_3(\mathcal{P}, \{0\})$
  - $\mathcal{I} = [0, |\mathcal{S}|]$
  - $\mathcal{T}_{partial} = proj_{\vee 2}($ $DC_4(DC_3(\mathcal{Q},\mathcal{I}),\mathcal{I}) \cap DC_4(DC_2(\mathcal{P},\mathcal{S}),\mathcal{I}) \cap DC_1(DC_2(\mathcal{P},\mathcal{I}),\mathcal{S})$ $)$
- $T(s, b, b')$ iff $(P(s, b) \wedge b' = 0) \vee \exists s' \in S :$ $(Q(s, s') \wedge P(s, b) \wedge P(s', b'))$.

Definitions for Extra Operators

- $DC_2(\mathcal{P},\mathcal{S}) \triangleq \underline{\mathcal{P}}$, where $\underline{\mathcal{P}}(x,y,z) = \mathcal{P}(x,z) \wedge \mathcal{S}(y)$
- $DC_3(\mathcal{Q},\mathcal{I}) \triangleq \underline{\mathcal{Q}}$, where $\underline{\mathcal{Q}}(x,y,z) = \mathcal{Q}(x,y) \wedge \mathcal{I}(z)$
- $DC_1(\mathcal{R},\mathcal{S}) \triangleq \underline{\mathcal{R}}$, where $\underline{\mathcal{R}}(x,y,z,h) = \mathcal{R}(y,z,h) \wedge \mathcal{S}(x)$
- $DC_4(\mathcal{R},\mathcal{I}) \triangleq \underline{\mathcal{R}}$, where $\underline{\mathcal{R}}(x,y,z,h) = \mathcal{R}(x,y,z) \wedge \mathcal{I}(h)$
- $proj_{\vee 2}(\mathcal{F}) \triangleq \mathcal{F}'$, where $\mathcal{F}'(x,y,z) = \bigvee c : \mathcal{F}(x,c,y,z)$
- Signatures MDD: $\mathcal{T} \leftarrow \mathcal{W} \cup \mathcal{T}_{partial}$, where:
  - $\mathcal{W} = DC_3(\mathcal{P},\{0\})$
  - $\mathcal{I} = [0, |\mathcal{S}|]$
  - $\mathcal{T}_{partial} = proj_{\vee 2}($ $DC_4(DC_3(\mathcal{Q},\mathcal{I}),\mathcal{I}) \cap DC_4(DC_2(\mathcal{P},\mathcal{S}),\mathcal{I}) \cap DC_1(DC_2(\mathcal{P},\mathcal{I}),\mathcal{S})$ $)$
- $\mathcal{W}(s,b,b')$ iff $\mathcal{P}(s,b) \wedge b' \in \{0\}$
- $T(s,b,b')$ iff $(P(s,b) \wedge b' = 0) \vee \exists s' \in S : (Q(s,s') \wedge P(s,b) \wedge P(s',b'))$.

Substituting Extra Operators into $\mathcal{T}_{partial}$

- $DC_2(\mathcal{P},\mathcal{S}) \triangleq \underline{\mathcal{P}}$, where $\underline{\mathcal{P}}(x,y,z) = \mathcal{P}(x,z) \wedge \mathcal{S}(y)$
- $DC_3(\mathcal{Q},\mathcal{I}) \triangleq \underline{\mathcal{Q}}$, where $\underline{\mathcal{Q}}(x,y,z) = \mathcal{Q}(x,y) \wedge \mathcal{I}(z)$
- $DC_1(\mathcal{R},\mathcal{S}) \triangleq \underline{\mathcal{R}}$, where $\underline{\mathcal{R}}(x,y,z,h) = \mathcal{R}(y,z,h) \wedge \mathcal{S}(x)$
- $DC_4(\mathcal{R},\mathcal{I}) \triangleq \underline{\mathcal{R}}$, where $\underline{\mathcal{R}}(x,y,z,h) = \mathcal{R}(x,y,z) \wedge \mathcal{I}(h)$
- $proj_{\vee 2}(\mathcal{F}) \triangleq \mathcal{F}'$, where $\mathcal{F}'(x,y,z) = \bigvee c : \mathcal{F}(x,c,y,z)$
- $\mathcal{T}_{partial} = proj_{\vee 2}$
  - $DC_4(DC_3(\mathcal{Q},\mathcal{I}),\mathcal{I}) \cap DC_4(DC_2(\mathcal{P},\mathcal{S}),\mathcal{I}) \cap DC_1(DC_2(\mathcal{P},\mathcal{I}),\mathcal{S})$
- $\mathcal{T}_{partial}(s,b,b')$ iff $\bigvee s'$
  - $DC_4(DC_3(\mathcal{Q},\mathcal{I}),\mathcal{I})(s,s',b,b') \wedge$
    $DC_4(DC_2(\mathcal{P},\mathcal{S}),\mathcal{I})(s,s',b,b') \wedge$
    $DC_1(DC_2(\mathcal{P},\mathcal{I}),\mathcal{S})(s,s',b,b')$
- $T(s,b,b')$ iff $(P(s,b) \wedge b' = 0) \vee \exists s' \in S :$
  $(Q(s,s') \wedge P(s,b) \wedge P(s',b'))$.

Substituting Extra Operators into $\mathcal{T}_{partial}$

- $DC_2(\mathcal{P}, \mathcal{S}) \triangleq \underline{\mathcal{P}}$, where $\underline{\mathcal{P}}(x, y, z) = \mathcal{P}(x, z) \wedge \mathcal{S}(y)$
- $DC_3(\mathcal{Q}, \mathcal{I}) \triangleq \underline{\mathcal{Q}}$, where $\underline{\mathcal{Q}}(x, y, z) = \mathcal{Q}(x, y) \wedge \mathcal{I}(z)$
- $DC_1(\mathcal{R}, \mathcal{S}) \triangleq \underline{\mathcal{R}}$, where $\underline{\mathcal{R}}(x, y, z, h) = \mathcal{R}(y, z, h) \wedge \mathcal{S}(x)$
- $DC_4(\mathcal{R}, \mathcal{I}) \triangleq \underline{\mathcal{R}}$, where $\underline{\mathcal{R}}(x, y, z, h) = \mathcal{R}(x, y, z) \wedge \mathcal{I}(h)$
- $proj_{\vee 2}(\mathcal{F}) \triangleq \mathcal{F}'$, where $\mathcal{F}'(x, y, z) = \bigvee c : \mathcal{F}(x, c, y, z)$
- $\mathcal{T}_{partial} = proj_{\vee 2}$
  - $DC_4(DC_3(\mathcal{Q}, \mathcal{I}), \mathcal{I}) \cap DC_4(DC_2(\mathcal{P}, \mathcal{S}), \mathcal{I}) \cap DC_1(DC_2(\mathcal{P}, \mathcal{I}), \mathcal{S})$
- $\mathcal{T}_{partial}(s, b, b')$ iff $\bigvee s'$
  - $[\mathcal{Q}(s, s') \wedge \mathcal{I}(b) \wedge \mathcal{I}(b')] \wedge [\mathcal{P}(s, b) \wedge \mathcal{S}(s') \wedge \mathcal{I}(b')] \wedge [\mathcal{P}(s', b') \wedge \mathcal{I}(b) \wedge \mathcal{S}(s)]$
- $T(s, b, b')$ iff $(P(s, b) \wedge b' = 0) \vee \exists s' \in S :$ $(Q(s, s') \wedge P(s, b) \wedge P(s', b'))$.

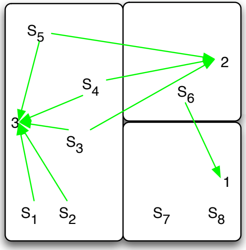
Substituting Extra Operators into $\mathcal{T}_{partial}$

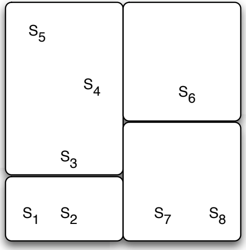
- $DC_2(\mathcal{P}, \mathcal{S}) \triangleq \underline{\mathcal{P}}$, where $\underline{\mathcal{P}}(x, y, z) = \mathcal{P}(x, z) \wedge \mathcal{S}(y)$
- $DC_3(\mathcal{Q}, \mathcal{I}) \triangleq \underline{\mathcal{Q}}$, where $\underline{\mathcal{Q}}(x, y, z) = \mathcal{Q}(x, y) \wedge \mathcal{I}(z)$
- $DC_1(\mathcal{R}, \mathcal{S}) \triangleq \underline{\mathcal{R}}$, where $\underline{\mathcal{R}}(x, y, z, h) = \mathcal{R}(y, z, h) \wedge \mathcal{S}(x)$
- $DC_4(\mathcal{R}, \mathcal{I}) \triangleq \underline{\mathcal{R}}$, where $\underline{\mathcal{R}}(x, y, z, h) = \mathcal{R}(x, y, z) \wedge \mathcal{I}(h)$
- $proj_{\vee 2}(\mathcal{F}) \triangleq \mathcal{F}'$, where $\mathcal{F}'(x, y, z) = \bigvee c : \mathcal{F}(x, c, y, z)$
- $\mathcal{T}_{partial} = proj_{\vee 2}$
  - $DC_4(DC_3(\mathcal{Q}, \mathcal{I}), \mathcal{I}) \cap DC_4(DC_2(\mathcal{P}, \mathcal{S}), \mathcal{I}) \cap DC_1(DC_2(\mathcal{P}, \mathcal{I}), \mathcal{S})$
- $\mathcal{T}_{partial}(s, b, b')$ iff $\bigvee s'$
  - $[\mathcal{Q}(s, s')] \wedge [\mathcal{P}(s, b)] \wedge [\mathcal{P}(s', b')] \wedge \mathcal{S}(s) \wedge \mathcal{S}(s') \wedge \mathcal{I}(b) \wedge \mathcal{I}(b')$
- $T(s, b, b')$ iff $(P(s, b) \wedge b' = 0) \vee \exists s' \in S : (Q(s, s') \wedge P(s, b) \wedge P(s', b'))$.

Hybrid Algorithm Block Splitting/Numbering

- S — State Space
- $T \subseteq S \times [1, |S|] \times [0, |S|]$ — Signature map to pairs of blocks
- $P' \subseteq S \times [1, |S|]$ — Partition block number function of state
- New partition blocks for each different signature.
- Block number for each state according to its signature.
- $\exists f \in [1, |S|] \times [1, |S|] \times [0, |S|] : \forall s \in S : \forall b \in [1, |S|] :$
  $P'(s, b)$ iff $\{\langle b_1, b_2\rangle | f(b, b_1, b_2)\} = \{\langle b_1, b_2\rangle | T(s, b_1, b_2)\}$.

Hybrid Algorithm Block Splitting

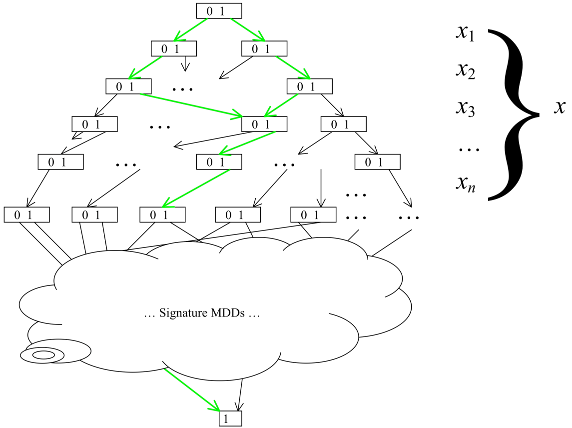
T: 

, P': 

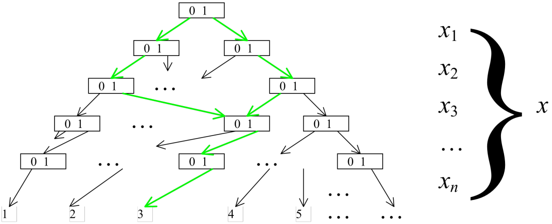
Hybrid Algorithm Block Renumbering Calculation

- Utilize canonicity of MDD
- Utilize fact that MDD is non-interleaved with state toward root
- Recursively DFS signature MDD $\mathcal{T}$
- Assign new partition number upon finding new signature.

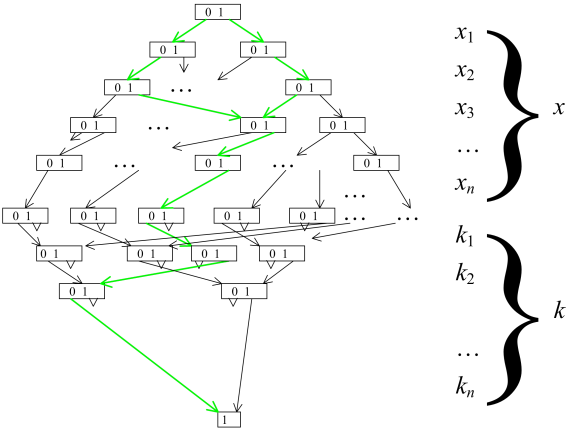
Hybrid Algorithm: Signatures MDD



Hybrid Algorithm: Block Renumbering $S \to \mathbb{N}$



Hybrid Algorithm: Block Renumbering $S \rightarrow \mathbb{N}$



Hybrid Algorithm Block Renumbering Algorithm

Assign new block number, corresponding to signature, to each state.

Algorithm: SigRenum(MDD $\mathcal{T}$)

Return SigRenum(MDD $\mathcal{T}$) from cache if possible.
If $\mathcal{T}$ is above signature level then

- let $\mathcal{R}$ = new MDD with each child $\mathcal{R}_i$ = SigRenum($\mathcal{T}_i$)

else

- let $\mathcal{R}$ = BDD for value of counter
- increment counter

Put $\mathcal{R}$ = SigRenum(MDD $\mathcal{T}$) into cache.
Return $\mathcal{R}$

Hybrid Algorithm

Given: Initial partition block numbering in variable $\mathcal{P}$, transition relation in $\mathcal{Q}$, state space in $\mathcal{S}$.
Returns final partition block numbering in $\mathcal{P}$.

Algorithm: refinement of block numbering using signature

Repeat:

- $\mathcal{P}_{old} \leftarrow \mathcal{P}$
- $\mathcal{T} \leftarrow \mathcal{W} \cup \mathcal{T}_{partial}$, where:
  - let: $\mathcal{W} \leftarrow DC_3(\mathcal{P}, \{0\})$, and: $\mathcal{I} \leftarrow [0, |\mathcal{S}|]$
  - $\mathcal{T}_{partial} \leftarrow proj_{\vee 2}$ $DC_4(DC_3(\mathcal{Q},\mathcal{I}),\mathcal{I}) \cap DC_4(DC_2(\mathcal{P},\mathcal{S}),\mathcal{I}) \cap DC_1(DC_2(\mathcal{P},\mathcal{I}),\mathcal{S})$
- $\mathcal{P} \leftarrow$ SigRenum($\mathcal{T}$)

Until $\mathcal{P} = \mathcal{P}_{old}$

Hybrid Algorithm with Transition Labeling

Given: Initial partition block numbering in variable $\mathcal{P}$, transition relation in $\mathcal{Q}$, state space in $\mathcal{S}$.
Returns final partition block numbering in $\mathcal{P}$.

Algorithm: refinement of block numbering using signature

Repeat:

- $\mathcal{P}_{old} \leftarrow \mathcal{P}$
- For each $t \in label$ loop:
  - $\mathcal{T} \leftarrow \mathcal{W} \cup \mathcal{T}_{partial}$, where:
    - let: $\mathcal{W} \leftarrow DC_3(\mathcal{P}, \{0\})$, and: $\mathcal{I} \leftarrow [0, |\mathcal{S}|]$
    - $\mathcal{T}_{partial} \leftarrow proj_{\vee 2}$ $DC_4(DC_3(\mathcal{Q}_t, \mathcal{I}), \mathcal{I}) \cap DC_4(DC_2(\mathcal{P}, \mathcal{S}), \mathcal{I}) \cap DC_1(DC_2(\mathcal{P}, \mathcal{I}), \mathcal{S})$
  - $\mathcal{P} \leftarrow$ SigRenum($\mathcal{T}$)

Until $\mathcal{P} = \mathcal{P}_{old}$

Outline

Example from Algorithm 1 Signature MDD

- Signatures MDD: $\mathcal{T} \leftarrow proj_{\vee 3}((DC_2(\mathcal{Q},\mathcal{S})) \cap (DC_1(\mathcal{E},\mathcal{S})))$
- Calculate: $((DC_2(\mathcal{Q},\mathcal{S})) \cap (DC_1(\mathcal{E},\mathcal{S})))$ using single recursive function.
- Avoid construction of intermediates: $DC_2(\mathcal{Q},\mathcal{S})$ and $DC_1(\mathcal{E},\mathcal{S})$.
- Recursive function will have 3 MDD parameters: $\mathcal{Q},\mathcal{E},\mathcal{S}$.
- Given $\mathcal{E} = \mathcal{E}^{-1}$ and $\mathcal{E} \subseteq S \times S$.
- Each recursive call level corresponds to level of output MDD.

Algorithm 6: Unprojected Relational Composition

Calculate: $\mathcal{R} = ((DC_2(\mathcal{Q}, \mathcal{S})) \cap (DC_1(\mathcal{E}, \mathcal{S}))),$
so that $\mathcal{R}(a, b, c)$ iff $\mathcal{Q}(a, c) \wedge \mathcal{E}(b, c) \wedge \mathcal{S}(a)$

Algorithm: UcompL(MDD $\mathcal{Q}, \mathcal{E}, \mathcal{S}$) (memoized)

- Leaf level: Return $\mathcal{Q} \cap \mathcal{E}$
- "a" level
  - Return new MDD $\mathcal{R}$ where child $\mathcal{R}_i =$ UcompL($\mathcal{Q}_i, \mathcal{E}, \mathcal{S}_i$)
- "b" level
  - Return new MDD $\mathcal{R}$ where child $\mathcal{R}_i =$ UcompL($\mathcal{Q}, \mathcal{E}_i, \mathcal{S}$)
- "c" level
  - Return new MDD $\mathcal{R}$ where child $\mathcal{R}_i =$ UcompL($\mathcal{Q}_i, \mathcal{E}_i, \mathcal{S}$)

Improvements Applied to Both Algorithms

- Improvement implemented as a single highly parameterized recursive function: GenericComposeQQ.
- Applied to: $\mathcal{T} \leftarrow proj_{\vee 3}((DC_2(\mathcal{Q},\mathcal{S})) \cap (DC_1(\mathcal{E},\mathcal{S})))$, (signature for Algorithm 1).
- Applied to: $\mathcal{T}_{partial} = proj_{\vee 2}($ $DC_4(DC_3(\mathcal{Q},\mathcal{I}),\mathcal{I}) \cap DC_4(DC_2(\mathcal{P},\mathcal{S}),\mathcal{I}) \cap$ $DC_1(DC_2(\mathcal{P},\mathcal{I}),\mathcal{S})\ )$, (signature for Hybrid Algorithm).
- Not applied to: $\overline{\Delta\mathcal{E}} \leftarrow proj_{\vee 3}(DC_2(\mathcal{T},\mathcal{S}) \dot{\cup} DC_1(\mathcal{T},\mathcal{S}))$, ($\mathcal{E}$ update for Algorithm 1).
- where $x \dot{\cup} y \triangleq (x \setminus y) \cup (y \setminus x)$
- Could have been (avoid calculating $(x \setminus y)$ and $(y \setminus x)$).

SMART Integration

- All code implemented in a single unit: “ms_lumping”.
- Invoked from SMART by a single C++ function call: “bigint ComputeNumEQClass(state_model *mdl);”
- Calculates largest bisimulation and returns number of equivalence classes.
- Invocation caused by “num_eqclass” function in model.
- Uses multiple caches supplied by SMART MDD library (Thanks, Min!).
- Uses operations: $\cup$, $\cap$, $\setminus$, new MDD, $||$, etc. from SMART MDD library.
- Implements operations for interleaved MDDs: $proj_\vee$, $\circ$, DC_*, $|classes|$
- Implements SigRenum

Summary of Our Bisimulation Algorithms

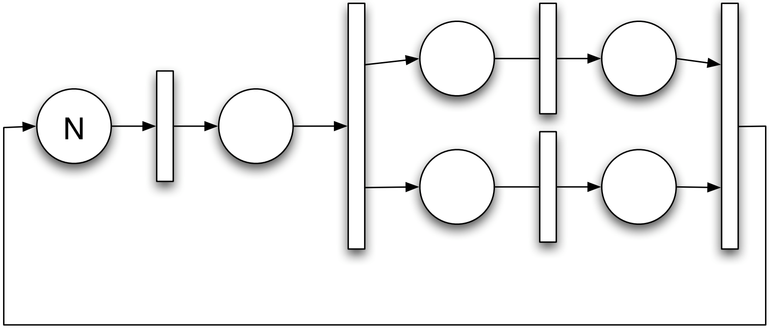
Two Algorithms:

Fully Implicit	Hybrid
Transition relation: Interleaved MDD	Transition relation: Interleaved MDD
Partition: Equivalence, Interleaved MDD	Partition: Block number function MDD

Outline

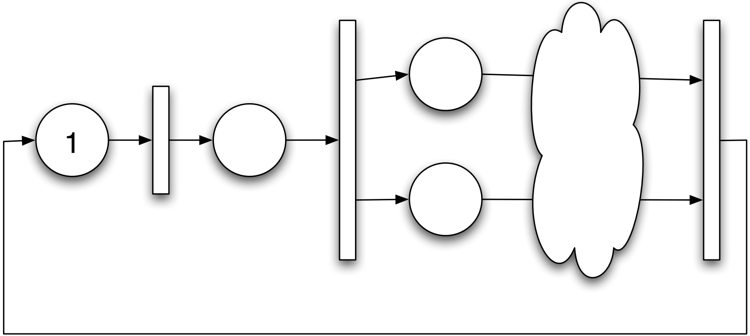
“Short” Simple Fork-JoinModel

Simple Fork-Join Petri net, parameterized in number of tokens N, in place 1 Has 6 places, 5 transitions, and initially N tokens.



“Tall” Simple Fork-JoinModel

Simple Fork-Join Petri net, parameterized in number of levels N, in parallel chain. Has 2N+2 places and 2N+1 transitions.



Summary of Our Models

Two simple fork-join models:

"Short" model
Fixed # places (variables)
Fixed # transitions
Growing # tokens = many values
Fixed depth, growing fanout

"Tall" model
Growing # places (variables)
Growing # transitions
Fixed # tokens = few values
Growing depth, fixed fanout

Model Statistics

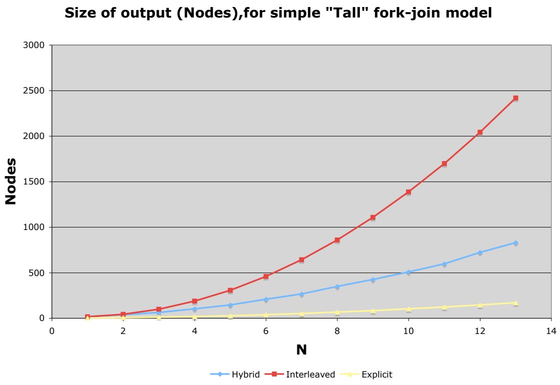
“Short” model

N	states	classes	iterations
1	6	5	3
2	20	15	3
3	50	35	4
4	105	70	4
5	196	126	5
6	336	210	6
7	540	330	7
8	825	495	8
9	1210	715	9
10	1716	1001	10
11	2366	1365	11
12	3185	1820	12
13	4200	2380	13

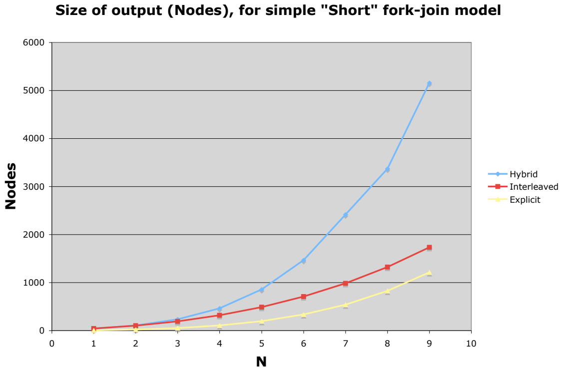
“Tall” model

N	states	classes	iterations
1	3	3	2
2	6	5	2
3	11	7	4
4	18	9	6
5	27	11	8
6	38	13	10
7	51	15	12
8	66	17	14
9	83	19	16
10	102	21	18
11	123	23	20
12	146	25	22
13	171	27	24

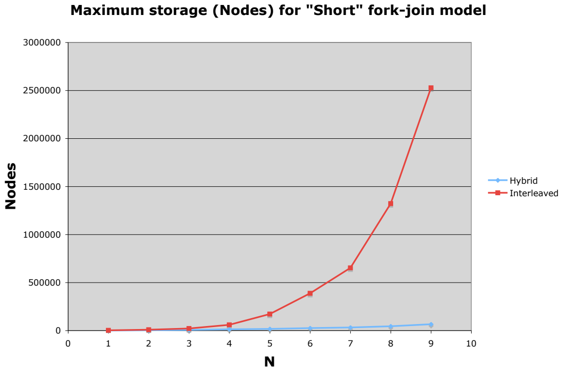
"Tall" MDD Size Results



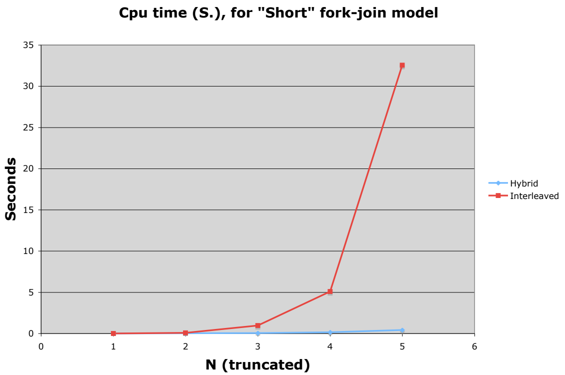
“Short” MDD Size Results (The Bright Side ...)



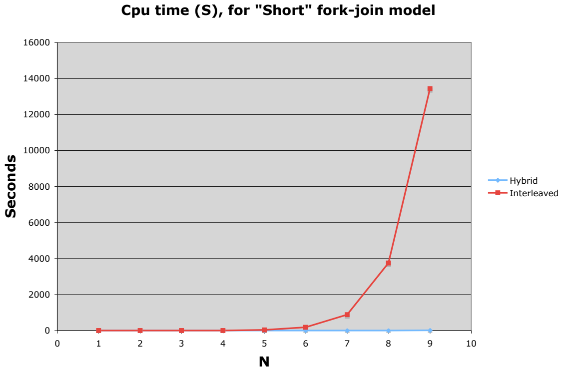
“Short” Maximum Storage



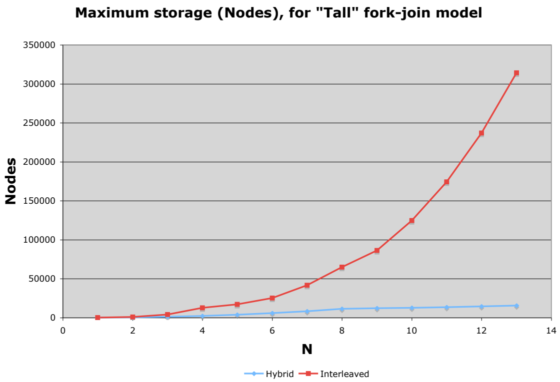
"Short" CPU Time



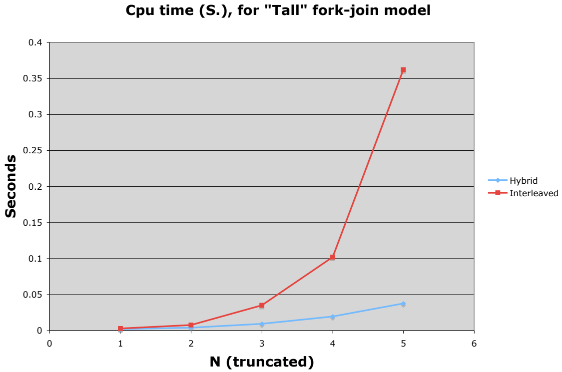
"Short" CPU Time



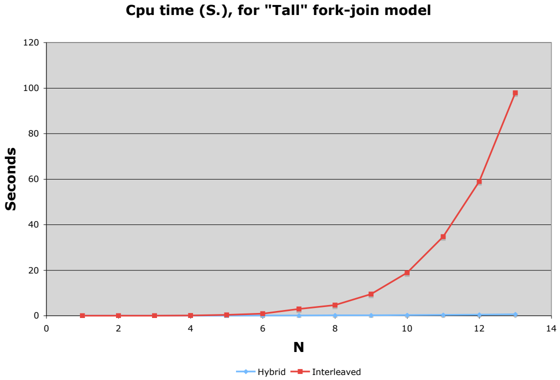
"Tall" Maximum Storage



“Tall” CPU Time



"Tall" CPU Time



Discussion of Results

Qualitative evaluation of Quantitative results:

- In nearly all cases, pure interleaved MDD algorithm performed poorly.
- Hybrid algorithm was surprisingly reasonable considering it is partly explicit.
- Output size of interleaved MDD is better in one case.

Additional Thoughts:

- Perhaps models are too small.
- "Short" model output results show trend for interleaved output size growing slower than $|S|$.
- Optimization should have been done for Algorithm 1 partition update.
- Intermediate results in symmetric difference may be large.

Future Work

Improvements to current work:

- Algorithm 1 partition update.
- Quantification/Projection Improvements.
- Additional models.

Other related work:

- If reasonable, apply to lumping problem.
- Affine decision diagrams.
- Apply distance algorithm for initial partition.
- Saturation for direct exploration of $\eqsim$.

Summary

- Implementation of two bisimulation algorithms in SMART
- Comparison using two Petri net models.
- Initially, partition representation by block numbering function wins.

- Future:
    - Interleaved representation of equivalence relation may actually be reasonable.
    - Saturation should also be applied to this problem.
    - Lumping.

The End

fin

After The End

(Click here for a reference.)

Ways to Represent Partitions

1. Equivalence Relation (Non-Interleaved) (link)
   - $\langle s_1, s_2 \rangle \mid s_1, s_2 \in S$
   - Variable ordering: $x_1, x_2, x_3, \ldots, y_1, y_2, y_3, \ldots$
2. Equivalence Relation (Interleaved) (link)
   - $\langle s_1, s_2 \rangle \mid s_1, s_2 \in S$
   - Variable ordering: $x_1, y_1, x_2, y_2, x_3, y_3, \ldots$
3. Lists of Partition Blocks (link)
   - $B_1, B_2, B_3, B_4, \ldots \mid B_* \subseteq S$
   - Variable ordering: $x_1, x_2, x_3, \ldots$
4. Block Numbering/function of state (link)
   - $\langle s, n \rangle \mid s \in S, n \in \mathbb{N}$
   - Variable ordering: $x_1, x_2, x_3, \ldots k_1, k_2, k_3, \ldots$

Ways to Represent Partitions

1. Equivalence Relation (Non-Interleaved) (link)
   - $\langle s_1, s_2 \rangle \mid s_1, s_2 \in S$
   - Variable ordering: $x_1, x_2, x_3, \ldots, y_1, y_2, y_3, \ldots$
2. Equivalence Relation (Interleaved) (link)
   - $\langle s_1, s_2 \rangle \mid s_1, s_2 \in S$
   - Variable ordering: $x_1, y_1, x_2, y_2, x_3, y_3, \ldots$
3. Lists of Partition Blocks (link)
   - $B_1, B_2, B_3, B_4, \ldots \mid B_* \subseteq S$
   - Variable ordering: $x_1, x_2, x_3, \ldots$
4. Block Numbering/function of state (link)
   - $\langle s, n \rangle \mid s \in S, n \in \mathbb{N}$
   - Variable ordering: $x_1, x_2, x_3, \ldots k_1, k_2, k_3, \ldots$

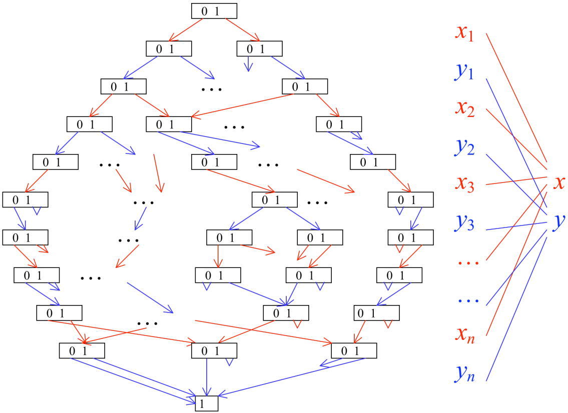
Ways to Represent Partitions

1. Equivalence Relation (Non-Interleaved) (link)
   - $\langle s_1, s_2 \rangle \mid s_1, s_2 \in S$
   - Variable ordering: $x_1, x_2, x_3, \ldots, y_1, y_2, y_3, \ldots$
2. Equivalence Relation (Interleaved) (link)
   - $\langle s_1, s_2 \rangle \mid s_1, s_2 \in S$
   - Variable ordering: $x_1, y_1, x_2, y_2, x_3, y_3, \ldots$
3. Lists of Partition Blocks (link)
   - $B_1, B_2, B_3, B_4, \ldots \mid B_* \subseteq S$
   - Variable ordering: $x_1, x_2, x_3, \ldots$
4. Block Numbering/function of state (link)
   - $\langle s, n \rangle \mid s \in S, n \in \mathbb{N}$
   - Variable ordering: $x_1, x_2, x_3, \ldots k_1, k_2, k_3, \ldots$

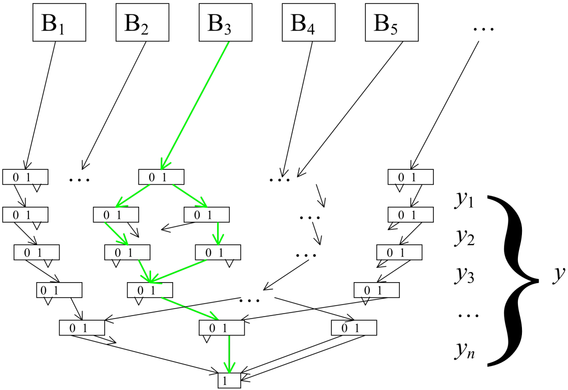
Ways to Represent Partitions

1. Equivalence Relation (Non-Interleaved) (link)
   - $\langle s_1, s_2 \rangle \mid s_1, s_2 \in S$
   - Variable ordering: $x_1, x_2, x_3, \ldots, y_1, y_2, y_3, \ldots$
2. Equivalence Relation (Interleaved) (link)
   - $\langle s_1, s_2 \rangle \mid s_1, s_2 \in S$
   - Variable ordering: $x_1, y_1, x_2, y_2, x_3, y_3, \ldots$
3. Lists of Partition Blocks (link)
   - $B_1, B_2, B_3, B_4, \ldots \mid B_* \subseteq S$
   - Variable ordering: $x_1, x_2, x_3, \ldots$
4. Block Numbering/function of state (link)
   - $\langle s, n \rangle \mid s \in S, n \in \mathbb{N}$
   - Variable ordering: $x_1, x_2, x_3, \ldots k_1, k_2, k_3, \ldots$

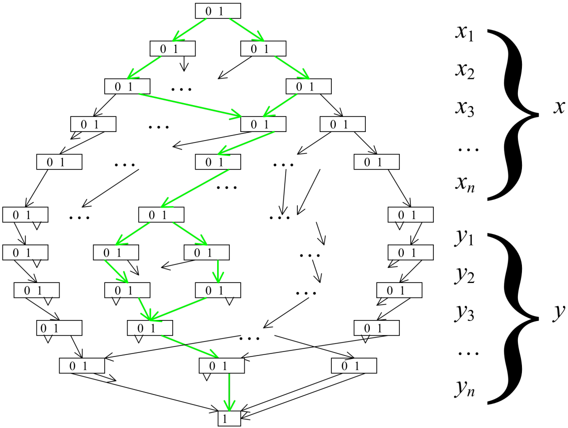
Partition Representation: Equivalence Relation (Interleaved) $\{\langle x, y\rangle | E(x, y)\}$



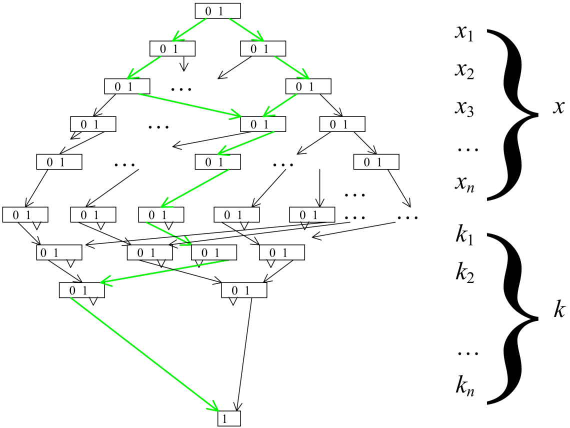
Partition Representation: Lists of Partition Blocks (or Array etc) $\mathbb{N} \rightarrow S$



Partition Representation: Equivalence Relation (Non-Interleaved) $\{\langle x, y\rangle | E(x, y)\}$



Partition Representation: Block Numbering/function of state $S \rightarrow \mathbb{N}$



Bibliography I

R. Milner.
Communication and Concurrency.
Prentice Hall, 1989.