Thèse de Doctorat de l’Université Pierre et Marie Curie

présentée par

Maximilien Colange

Symmetry Reduction and Symbolic Data Structures
for Model Checking of Distributed Systems

soutenue le 10 décembre 2013
devant la commission composée de :

A. Bouajjani  
F. Vernadat  
B. Bérard  
M. Heiner  
T. Junttila  
F. Kordon  
S. Baarir  
Y. Thierry-Mieg  

Rapporteur  
Rapporteur  
Examinatrice  
Examinatrice  
Examinateur  
Directeur  
Co-Encadrant  
Co-Encadrant  

Université Paris Diderot  
INSA Toulouse  
Université Pierre et Marie Curie  
University of Technology Cottbus  
Aalto University  
Université Pierre et Marie Curie  
Université Paris Ouest  
Université Pierre et Marie Curie
Context: Formal Verification

**Critical systems**

automatic transportation, robotic surgery, power plants management . . .

**Concurrent systems**

- modern car \(\sim\) 100 computing devices, and growing
- A380 avionics = Ethernet network
- highways with driverless cars . . .

How to ensure safety and reliability of such systems?
Context: Formal Verification

**Critical systems**

automatic transportation, robotic surgery, power plants management . . .

**Concurrent systems**

- modern car $\sim$ 100 computing devices, and growing
- A380 avionics = Ethernet network
- highways with driverless cars . . .

How to ensure safety and reliability of such systems?

- Tests and/or simulation
**Context: Formal Verification**

**Critical systems**

- automatic transportation, robotic surgery, power plants management . . .

**Concurrent systems**

- modern car $\sim 100$ computing devices, and growing
- A380 avionics = Ethernet network
- highways with driverless cars . . .

How to ensure safety and reliability of such systems?

- Tests and/or simulation *cannot be exhaustive*
Context: Formal Verification

**Critical systems**

automatic transportation, robotic surgery, power plants management . . .

**Concurrent systems**

- modern car $\sim$ 100 computing devices, and growing
- A380 avionics = Ethernet network
- highways with driverless cars . . .

How to ensure safety and reliability of such systems?

- Tests and/or simulation *cannot be exhaustive*
- Formal methods *give a guarantee (up to the modelling)*
**Context: Formal Verification**

**Critical systems**

- automatic transportation, robotic surgery, power plants management . . .

**Concurrent systems**

- modern car $\sim 100$ computing devices, and growing
- A380 avionics = Ethernet network
- highways with driverless cars . . .

How to ensure safety and reliability of such systems?

- Tests and/or simulation *cannot be exhaustive*
- Formal methods *give a guarantee (up to the modelling)*
  - assisted mathematical proof
Context: Formal Verification

**Critical systems**

automatic transportation, robotic surgery, power plants management . . .

**Concurrent systems**

- modern car $\sim$ 100 computing devices, and growing
- A380 avionics = Ethernet network
- highways with driverless cars . . .

How to ensure safety and reliability of such systems?

- Tests and/or simulation *cannot be exhaustive*
- Formal methods *give a guarantee* (up to the modelling)
  - assisted mathematical proof
  - model-checking: exploration of all the possible behaviors
Problem

Combinatorial Explosion

number of behaviors grows exponentially with the number of components inherent to concurrent systems

- severely hinders model-checking, that aims to explore behaviors

e.g. $n$ clients, $p$ servers: $p^n$ possible connexions

25 years of Model-Checking $\Rightarrow$ Turing Award (2007)
Problem

### Combinatorial Explosion

number of behaviors grows exponentially with the number of components inherent to concurrent systems
- severely hinders model-checking, that aims to explore behaviors

- e.g. $n$ clients, $p$ servers: $p^n$ possible connexions
- 25 years of Model-Checking $\Rightarrow$ Turing Award (2007)

How to counter the combinatorial explosion?
Problem

Combinatorial Explosion

number of behaviors grows exponentially with the number of components inherent to concurrent systems
- severely hinders model-checking, that aims to explore behaviors

e.g. $n$ clients, $p$ servers: $p^n$ possible connexions
25 years of Model-Checking $\Rightarrow$ Turing Award (2007)

How to counter the combinatorial explosion?

- **Handle** [Bryant, 1986, Burch et al., 1992, Couvreur et al., 2002]
  - Decision Diagrams: use efficient compact data structures
Problem

Combinatorial Explosion

The number of behaviors grows exponentially with the number of components inherent to concurrent systems, severely hindering model-checking, which aims to explore behaviors.

For example, $n$ clients, $p$ servers: $p^n$ possible connections.

25 years of Model-Checking $\Rightarrow$ Turing Award (2007)

How to counter the combinatorial explosion?

- **Handle** [Bryant, 1986, Burch et al., 1992, Couvreur et al., 2002]
  - Decision Diagrams: use efficient compact data structures
- **Fight** [Chiola et al., 1990, Clarke et al., 1996, Junnttila, 2003]
  - Symmetry reduction: avoid exploring similar behaviors
Two main contributions presented today:

1. improve decision diagrams manipulation for model-checking of concurrent systems [CAV 2013]
2. combine symmetry reduction and decision diagrams, in order to stack their respective gains [ACSD 2012]

My thesis features other contributions [ICATPN 2011, Monterey 2012]
1 Context

2 New Efficient Operations for Decision Diagrams [CAV 2013]

3 Combine Symmetry Reduction and Decision Diagrams [ACSD 2012]
Finite Transition Systems

**Definition**

Finite TS $\mathcal{K} = (S, \rightarrow)$

$\rightarrow$ binary relation over $S$: $\rightarrow \subseteq S \times S$

**Hypothesis**

$S \subset \mathbb{N}^k$ fixed-size vectors of integers

each position (address) denoted by a variable: $x_1, \ldots, x_k$
BDD [Bryant, 1986],
MDD [Srinivasan et al., 1990],
DDD [Couvreur et al., 2002]

- a path = a state ∈ \( \mathbb{N}^k \)

\( (2, 3, 1) \)
\( (1, 1, 1) \)
\( (1, 2, 3) \)
Shared Decision Diagrams and Finite Transition Systems

- BDD [Bryant, 1986],
  MDD [Srinivasan et al., 1990],
  DDD [Couvreur et al., 2002]

- a path = a state ∈ \( \mathbb{N}^k \)

- \(|DD| = \# \text{ nodes} \sim \log(|set|)\)

\[
(2, 3, 1) \\
(1, 1, 1) \\
(1, 2, 3)
\]
Shared Decision Diagrams and Finite Transition Systems

- BDD [Bryant, 1986],
  MDD [Srinivasan et al., 1990],
  DDD [Couvreur et al., 2002]

- a path = a state ∈ \( \mathbb{N}^k \)

- \(|DD| = \#\text{ nodes} \sim \log(|set|)\)

- efficient manipulation operations
  - unique tables + caches
  - complexity of operations related to \(|DD|\), not to \(|set|\)
  - comparison in \(O(1)\)
  - union ... in \(O(|DD_1| + |DD_2|)\)
Operations on DD: 2k-levels [Burch et al., 1992]

Encode symbolically a binary relation on states $\Delta \subseteq S \times S = \mathbb{N}^k \times \mathbb{N}^k$?

2k-level

$\Delta$ = subset of $\mathbb{N}^{2k}$
encode it with a DD with 2k variables
$\Delta(S) = \{s'| (s, s') \in \Delta \} \subseteq \mathbb{N}^k$

Problem: pre-computation

- requires a bound
- all potential values
- potential values $\sim \exp(|\text{support}|)$
  - support($x + y$) = \{x, y\}
  - support($u \ast v + w$) = \{u, v, w\}
Operations on DD: 2\(k\)-levels [Burch et al., 1992]

Encode symbolically a binary relation on states \(\Delta \subseteq S \times S = \mathbb{N}^k \times \mathbb{N}^k\)?

2\(k\)-level

\(\Delta = \text{subset of } \mathbb{N}^{2k}\)

encode it with a DD with 2\(k\) variables

\(\Delta(S) = \{s'|(s, s') \in \Delta\} \subseteq \mathbb{N}^k\)

Problem: pre-computation

- requires a bound
- all potential values
- potential values \(\sim \exp(|\text{support}|)\)
  - support(\(x + y\)) = \(\{x, y\}\)
  - support(\(u \ast v + w\)) = \(\{u, v, w\}\)

\[\text{e.g. } z := x + y\]
Homomorphism

Recursive encoding

\[ h : DD \rightarrow DD \]

\[ h(d_1 \cup d_2) = h(d_1) \cup h(d_2) \]
Operations on DD: homomorphisms [Couvreur et al., 2002]

Homomorphism

Recursive encoding

\[ h : DD \mapsto DD \]

\[ h(d_1 \cup d_2) = h(d_1) \cup h(d_2) \]
Operations on DD: homomorphisms [Couvreur et al., 2002]

Homomorphism

Recursive encoding

\[ h : DD \mapsto DD \]

\[ h(d_1 \cup d_2) = h(d_1) \cup h(d_2) \]
Operations on DD: homomorphisms [Couvreur et al., 2002]

Homomorphism

Recursive encoding

\[ h : DD \mapsto DD \]

\[ h(d_1 \cup d_2) = h(d_1) \cup h(d_2) \]

---

Diagram:

- Variables: x, y, z
- Edges for 0 and 1 values
- Initial nodes for x, y, z
- Terminal node

- z := 0
- z := 1

- z
- 2
Homomorphism

Recursive encoding

\[ h : DD \mapsto DD \]

\[ h(d_1 \cup d_2) = h(d_1) \cup h(d_2) \]
Homomorphism

Recursive encoding

\[ h : DD \mapsto DD \]

\[ h(d_1 \cup d_2) = h(d_1) \cup h(d_2) \]

- no pre-computation
- no bound needed
- dynamic support reduction
- what if variables in wrong order?
Towards New Operations on DD

Variables in “wrong” order

\[ w := x + y \]
Towards New Operations on DD

Variables in “wrong” order

\[ w := x + y \]

- equivalence classes w.r.t. the value of \( x + y \)
- \( O(|\text{codomain}|) \) instead of \( O(|\text{set}|) \)
Towards New Operations on DD

Variables in “wrong” order

\[ w := x + y \]

- equivalence classes w.r.t. the value of \( x + y \)
- \( \mathcal{O}(|\text{codomain}|) \) instead of \( \mathcal{O}(|\text{set}|) \)
Towards New Operations on DD

Variables in “wrong” order

\( w := x + y \)

- equivalence classes w.r.t. the value of \( x + y \)
- \( O(|\text{codomain}|) \) instead of \( O(|\text{set}|) \)
Towards New Operations on DD

<table>
<thead>
<tr>
<th>Variables in “wrong” order</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w := x + y$</td>
</tr>
</tbody>
</table>

- equivalence classes w.r.t. the value of $x + y$
- $O(|codomain|)$ instead of $O(|set|)$
- refine
Towards New Operations on DD

Variables in “wrong” order

\[ w := x + y \]

- equivalence classes w.r.t. the value of \( x + y \)
- \( \mathcal{O}(|\text{codomain}|) \) instead of \( \mathcal{O}(|\text{set}|) \)
- refine
Towards New Operations on DD

Variables in “wrong” order

\[ w := x + y \]

- equivalence classes w.r.t. the value of \( x + y \)
- \( O(\text{\textit{codomain}}) \) instead of \( O(\text{\textit{set}}) \)
- refine
Towards New Operations on DD

Variables in “wrong” order

\[ w := x + y \]

- equivalence classes w.r.t. the value of \( x + y \)
- \( O(|\text{codomain}|) \) instead of \( O(|\text{set}|) \)
- refine
- merge
Towards New Operations on DD

Variables in “wrong” order

\[ w := x + y \]

- equivalence classes w.r.t. the value of \( x + y \)
- \( O(\text{|codomain|}) \) instead of \( O(\text{|set|}) \)
- refine
- merge
Towards New Operations on DD

Variables in “wrong” order

\[ w := x + y \]

- equivalence classes w.r.t. the value of \( x + y \)
- \( O(\text{codomain}) \) instead of \( O(\text{set}) \)
- refine
- merge
- constant assignment on each obtained subset
Towards New Operations on DD

Variables in “wrong” order

\[ w := x + y \]

- equivalence classes w.r.t. the value of \( x + y \)
- \( O(|\text{codomain}|) \) instead of \( O(|\text{set}|) \)
- refine
- merge
- constant assignment on each obtained subset
EquivSplit for Complex Operations

Evaluate high-level assignments

\[ \phi := \psi \text{ where } \phi \text{ and } \psi \text{ are arbitrary expressions} \]

Easy case: \( \phi \) is a constant address.
Use EquivSplit to evaluate \( \psi \)
On each subset, assign the value of \( \psi \) to the address \( \phi \)
EquivSplit for Complex Operations

Evaluate high-level assignments

\[ \phi := \psi \text{ where } \phi \text{ and } \psi \text{ are arbitrary expressions} \]

General case: \( \phi \) is not constant (pointer).
Idea: use EquivSplit twice, once for \( \phi \) and \( \psi \), then use constant assignments on each subset
ex: \( t[x+y] := z*x+1 \)
EquivSplit for Complex Operations

Evaluate high-level assignments

$\phi := \psi$ where $\phi$ and $\psi$ are arbitrary expressions

General case: $\phi$ is not constant (pointer).
Idea: use EquivSplit twice, once for $\phi$ and $\psi$, then use constant assignments on each subset
ex: $t[x+y] := z*x+1$

\[
\begin{align*}
\phi &= t[0] \\
\phi &= t[2] \\
\phi &= t[3]
\end{align*}
\]
EquivSplit for Complex Operations

Evaluate high-level assignments

\[ \phi := \psi \] where \( \phi \) and \( \psi \) are arbitrary expressions

General case: \( \phi \) is not constant (pointer).
Idea: use EquivSplit twice, once for \( \phi \) and \( \psi \), then use constant assignments on each subset

ex: \( t[x+y] := z \times x + 1 \)

\[ \phi := \psi \]

\[ \psi = 1 \]

\[ \psi = 2 \]
EquivSplit for Complex Operations

Evaluate high-level assignments

\[ \phi := \psi \] where \( \phi \) and \( \psi \) are arbitrary expressions

General case: \( \phi \) is not constant (pointer).
Idea: use EquivSplit twice, once for \( \phi \) and \( \psi \), then use constant assignments on each subset
ex: \( t[x+y] := z*x+1 \)

\[
t[x + y] := z * x + 1
\]
EquivSplit for Complex Operations

Evaluate high-level assignments

\[ \phi := \psi \text{ where } \phi \text{ and } \psi \text{ are arbitrary expressions} \]

General case: \( \phi \) is not constant (pointer).
Idea: use EquivSplit twice, once for \( \phi \) and \( \psi \), then use constant assignments on each subset
ex: \( t[x+y] := z*x+1 \)

\[
\begin{align*}
\phi & := \psi \\
\phi_1 & := \psi_1 \\
\phi_2 & := \psi_1 \\
\phi_3 & := \psi_1 \\
\phi_1 & := \psi_2 \\
\phi_2 & := \psi_2 \\
\phi_3 & := \psi_2
\end{align*}
\]
Experimental Validation

Benchmark

BEEM benchmark ~ 400 instances

Comparison with

- LTSmin [Blom et al., 2010] explicit/symbolic model-checker
  - state space generation
  - 1 core, 10GB, 1 hour

- super-prove [Berkeley LSV Group, 2012] SAT solver
  - winner of the HWMCC (FMCAD event) since 2010
  - reachability problems
  - 4 cores, 1Gb, 15 min wall-clock-time
  - NB: super-prove multi-thread, but we are not!
Comparison with LTSmin

- state space generation: 1 core, 1 hour, 10 Gb
- below the diagonal = its is better

Comparison in time (s)  Comparison in memory (kb)
Comparison with super_prove

- reachability properties: 4 cores, 900s wall-clock, 1Gb
- there are difficult instances for both tools

<table>
<thead>
<tr>
<th></th>
<th>unsat</th>
<th>sat</th>
</tr>
</thead>
<tbody>
<tr>
<td>instances</td>
<td>456</td>
<td></td>
</tr>
<tr>
<td>its solves</td>
<td>376</td>
<td>192</td>
</tr>
<tr>
<td>sup solves</td>
<td>282</td>
<td>170</td>
</tr>
<tr>
<td>solved by both</td>
<td>258</td>
<td>165</td>
</tr>
<tr>
<td>solved by none</td>
<td>56</td>
<td></td>
</tr>
</tbody>
</table>

Comparison in time (s)
Abstract the Symbolic Engine from the User

My work is integrated in the symbolic model-checker used by the team.
1 Context

2 New Efficient Operations for Decision Diagrams [CAV 2013]

3 Combine Symmetry Reduction and Decision Diagrams [ACSD 2012]
Finite Transition Systems and Symmetries

finite TS $\mathcal{K} = (S, \rightarrow \subseteq S \times S)$
Finite Transition Systems and Symmetries

finite TS $\mathcal{K} = (S, \rightarrow \subseteq S \times S)$

$g : S \mapsto S$ bijective is a symmetry iff:
\[ \forall s, s' \in S, s \rightarrow s' \iff g.s \rightarrow g.s' \]

\[ s \xrightarrow{g} g.s \]
\[ s' \xrightarrow{g} g.s' \]
Finite Transition Systems and Symmetries

**finite** TS $\mathcal{K} = (S, \to \subseteq S \times S)$

$g : S \mapsto S$ bijective is a symmetry iff:

$\forall s, s' \in S, s \to s' \iff g.s \to g.s'$
Finite Transition Systems and Symmetries

finite TS $\mathcal{K} = (S, \rightarrow \subseteq S \times S)$

$g : S \mapsto S$ bijective is a symmetry iff:
$\forall s, s' \in S, s \rightarrow s' \iff g.s \rightarrow g.s'$

$s_1 \equiv_G s_2$ iff $\exists g, g.s_1 = s_2$
$\equiv_G$ equivalence relation
equivalence classes = orbits
Finite Transition Systems and Symmetries

finite TS $\mathcal{K} = (S, \rightarrow \subseteq S \times S)$

$g : S \mapsto S$ bijective is a symmetry iff:
$\forall s, s' \in S, s \rightarrow s' \iff g.s \rightarrow g.s'$

$s_1 \equiv_G s_2$ iff $\exists g, g.s_1 = s_2$
$\equiv_G$ equivalence relation
equivalence classes = orbits

Quotient graph = orbit graph
$\mathcal{K}_{/G} = (S_{/G}, \rightarrow_G \subseteq S_{/G} \times S_{/G})$
Finite Transition and Symmetries

Benefits of the quotient graph:

- $\mathcal{K}/G$ can be exponentially smaller than $\mathcal{K}$
- $\mathcal{K}/G$ preserves CTL* properties with symmetric atomic propositions
  [Haddad et al., 1995, Clarke et al., 1996]

Hypothesis

Without loss of generality

- $S \subset \mathbb{N}^k$
  states = integer vectors of size $k$
- $G \subseteq \mathfrak{S}(k)$
  symmetries permute positions in the vectors

  e.g. $\tau_{1,2}(6, 7, 8) = (7, 6, 8)$
Orbit representation problem

Two ways to represent an orbit

- use a dedicated representation [Chiola et al., 1990]
  - requires to adapt the transition relation
- choose one or several representative states in the orbit [Clarke et al., 1996]
  - the transition relation can be used as is

Finding representatives = canonization

- less representatives
  - = harder canonization
  - = smaller graph
How to represent an orbit symbolically?
How to represent an orbit symbolically?

⇒ choose a representative state per orbit
How to represent an orbit symbolically?

⇒ choose a representative state per orbit
  - for instance, given a total order on $S$, choose the minimum
    - lexicographic order
    - e.g. $s_1 > s_2 > s_3 > s_4 > s_5$
How to represent an orbit symbolically?

Current problems on canonization

- GRAPH ISOMORPHISM
- repeated for each new encountered state (state-by-state algorithms)
  - [Junntila, 2003]
How to represent an orbit symbolically?

[Clarke et al., 1996]

orbit relation maps every potential state to its representative

\[ \Delta_{\text{orbit}} = \{(s, \text{repr}(s))|s \in S\} \]

exponential size

\[ \rightarrow \text{quotient} = \circ \Delta_{\text{orbit}} \]

still a state-by-state algorithm
Our symbolic algorithm for canonization

But the red paths all lead to this minimum
Our symbolic algorithm for canonization

But the red paths all lead to this minimum

Canonization can be done iteratively only through $g_1$ and $g_2$: represent only a subset of $G$
Our symbolic algorithm for canonization

But the red paths all lead to this minimum

Canonization can be done iteratively only through \( g_1 \) and \( g_2 \): represent only a subset of \( G \)

\[
\Delta_{g_1} = \{(s, s)|g_1.s \geq s\} \cup \{(s, g_1.s)|g_1.s < s\}
\]

\[
\Delta_{g_2} = \{(s, s)|g_2.s \geq s\} \cup \{(s, g_2.s)|g_2.s < s\}
\]

\[
\Delta_H = \Delta_{g_1} \circ \Delta_{g_2} \circ \cdots \circ \Delta_{g_n}
\]

canonization algo based on \( \Delta^* \)

M. Colange
Combine Symmetry Reduction and Decision Diagrams
10 décembre 2013 20 / 29
Our symbolic algorithm for canonization

But the red paths all lead to this minimum

Canonization can be done iteratively only through \( g_1 \) and \( g_2 \): represent only a subset of \( G \)

\[
\Delta_{g_1} = \{(s, s) | g_1.s \geq s\} \cup \{(s, g_1.s) | g_1.s < s\}
\]
\[
\Delta_{g_2} = \{(s, s) | g_2.s \geq s\} \cup \{(s, g_2.s) | g_2.s < s\}
\]

\[\ldots\]

\[
\Delta_H = \Delta_{g_1} \circ \Delta_{g_2} \circ \cdots \circ \Delta_{g_n}
\]
canonization algo based on \( \Delta^*_H \)
A Note on Complexity

Any $H$ is correct!

_Whatever the chosen $H$, our algo $\Delta^*_H$ approximates $\Delta_{orbit}$ and chooses (possibly several) representatives per orbit._

- if $H = \{id\}$, $\Delta_H = id$, no canonization
- if $H = G$, $\Delta^*_H = \Delta_H = \Delta_{orbit}$ but $|H| \sim k!$
- larger $H \Rightarrow$ faster fixpoint but harder $\Delta_H$
- number of representatives depends on $H$
Choice of $H$

$\Delta^*_H = \Delta_{\text{orbit}}$ (Guarantees a unique representative)

$H \subseteq G$ is monotonic$_<$ w.r.t. $G$ iff:
$\forall s \in S, (\exists g \in G | g.s < s \Rightarrow \exists h \in H | h.s < s)$

Whenever a state $s$ is not the minimum of its orbit, there is a permutation in $H$ that reduces $s$.

- $H = G$ is always monotonic$_<$, but inefficient
- $|H|$ not polynomially (in $k$) bounded in general
- $H$ of linear (in $k$) size exist for commonly encountered groups
  - if $G = \mathbb{S}(k)$, then $H = \{\tau_i, i+1 | 1 \leq i < k\}$ monotonic$_<$
  - if $G$ is cyclic, $H = G$ is the only monotonic$_<$
  - if $G = \langle H_1, H_2 \rangle$, $H_1 \cup H_2$ not monotonic$_<$, but still good
Benchmarks

<table>
<thead>
<tr>
<th>Tools</th>
<th>symmetry</th>
<th>DD</th>
</tr>
</thead>
<tbody>
<tr>
<td>LoLA</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>its</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>its-sym</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

its-sym extends its → same DD implementation

- Parameterized Symmetric Colored Petri Nets
- state space generation
- confinement 1 hour and 10 GB
Benchmarks

Clients servers model

Combine Symmetry Reduction and Decision Diagrams
Benchmarks

SaleStore model
### Conclusion

#### Operations on DD

- Original fully symbolic algorithm for evaluating arbitrary expressions
  - Based on partitioning and successive refine-merge steps
  - Practical efficiency demonstrated experimentally
  - Expressive, wide scope of applications

#### Symmetries + DD

- First effective fully symbolic algorithm for canonization on DD
  - Based on a subset of the group of symmetries
  - Monotonic\(_<\) criterion to guarantee unique representative
  - Don’t care monotonic\(_<\), it always works!

Perspectives

Symmetry side

- symmetry detection
- temporal logic + symmetry

DD side

- generalize EquivSplit to hierarchical DD
- find new applications: infinite systems?
- provide a DD-free abstraction layer to the user
- compete with SAT/SMT-solvers
Bibliography I

Abc: A System for Sequential Synthesis and Verification, release 12/10/06.
http://www.eecs.berkeley.edu/~alanmi/abc/.

Ltsmin: Distributed and symbolic reachability.

Bryant, R. E. (1986).
Graph-based algorithms for boolean function manipulation.

Symbolic model checking: 10^20 States and beyond.
Information and computation, 98(2):142–170.

On well-formed coloured nets and their symbolic reachability graph.
In 11th International Conference on Application and Theory of Petri Nets.

Exploiting symmetry in temporal logic model checking.


My Papers


