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	*Rapporteur*	Université Paris Diderot
F. Vernadat	*Rapporteur*	INSA Toulouse
B. Bérard	*Examinatrice*	Université Pierre et Marie Curie
M. Heiner	*Examinatrice*	University of Technology Cottbus
T. Junttila	*Examinateur*	Aalto University
F. Kordon	*Directeur*	Université Pierre et Marie Curie
S. Baarir	*Co-Encadrant*	Université Paris Ouest
Y. Thierry-Mieg	*Co-Encadrant*	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

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
- Fight [Chiola et al., 1990, Clarke et al., 1996, Junttila, 2003]
    - Symmetry reduction: avoid exploring *similar* behaviors

Outline

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]

Finite Transition Systems

Definition

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

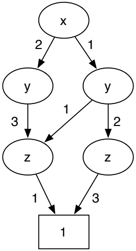
$\rightarrow$ binary relation over S: $\rightarrow \subseteq S \times S$

Hypothesis

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

each position (address) denoted by a variable: $x_1, \ldots x_k$

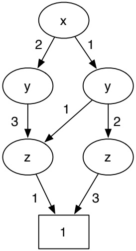
Shared Decision Diagrams and Finite Transition Systems



$(2, 3, 1)$
$(1, 1, 1)$
$(1, 2, 3)$

- BDD [Bryant, 1986],
  MDD [Srinivasan et al., 1990],
  DDD [Couvreur et al., 2002]
- a path = a state $\in \mathbb{N}^k$

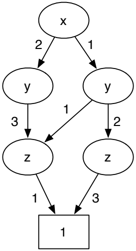
Shared Decision Diagrams and Finite Transition Systems



$$(2, 3, 1)$$
$$(1, 1, 1)$$
$$(1, 2, 3)$$

- BDD [Bryant, 1986], MDD [Srinivasan et al., 1990], DDD [Couvreur et al., 2002]
- a path = a state $\in \mathbb{N}^k$
- $|DD| = \#$ nodes $\sim \log(|set|)$

Shared Decision Diagrams and Finite Transition Systems



$(2, 3, 1)$
$(1, 1, 1)$
$(1, 2, 3)$

- BDD [Bryant, 1986],
  MDD [Srinivasan et al., 1990],
  DDD [Couvreur et al., 2002]
- a path = a state $\in \mathbb{N}^k$
- $|DD| = \#$ nodes $\sim \log(|set|)$
- efficient manipulation operations
  - unique tables + caches
  - complexity of operations related to $|DD|$, not to $|set|$
  - comparison in $\mathcal{O}(1)$
  - union ... in $\mathcal{O}(|DD_1| + |DD_2|)$

Operations on DD: $2k$-levels [Burch et al., 1992]

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

$2k$-level

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

Problem: pre-computation

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

Operations on DD: $2k$-levels [Burch et al., 1992]

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

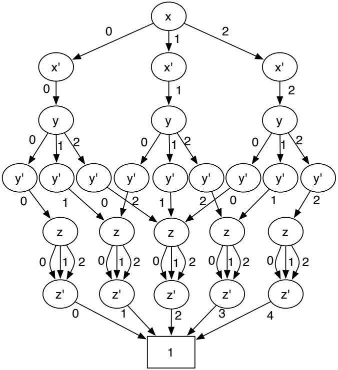
$2k$-level

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

Problem: pre-computation

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

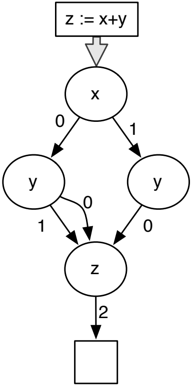
e.g. $z := x + y$



Operations on DD: homomorphisms [Couvreur et al., 2002]

Homomorphism

Recursive encoding
$h : DD \mapsto DD$
$h(d_1 \cup d_2) = h(d_1) \cup h(d_2)$



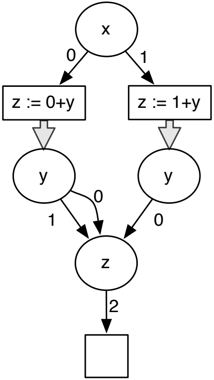
Operations on DD: homomorphisms [Couvreur et al., 2002]

Homomorphism

Recursive encoding

$h : DD \mapsto DD$

$h(d_1 \cup d_2) = h(d_1) \cup h(d_2)$



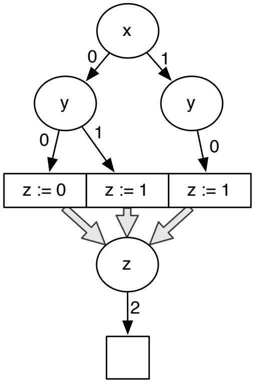
Operations on DD: homomorphisms [Couvreur et al., 2002]

Homomorphism

Recursive encoding

$h : DD \mapsto DD$

$h(d_1 \cup d_2) = h(d_1) \cup h(d_2)$



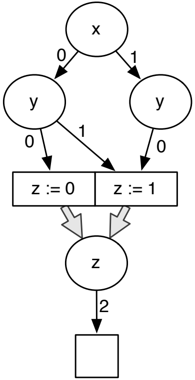
Operations on DD: homomorphisms [Couvreur et al., 2002]

Homomorphism

Recursive encoding

$h : DD \mapsto DD$

$h(d_1 \cup d_2) = h(d_1) \cup h(d_2)$



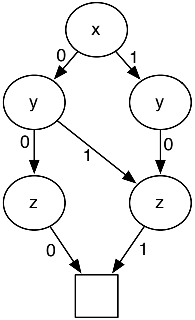
Operations on DD: homomorphisms [Couvreur et al., 2002]

Homomorphism

Recursive encoding

$h : DD \mapsto DD$

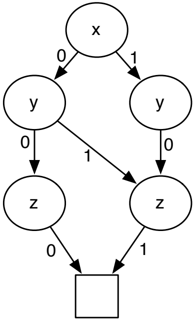
$h(d_1 \cup d_2) = h(d_1) \cup h(d_2)$



Operations on DD: homomorphisms [Couvreur et al., 2002]

Homomorphism

Recursive encoding
$h : DD \mapsto DD$
$h(d_1 \cup d_2) = h(d_1) \cup h(d_2)$

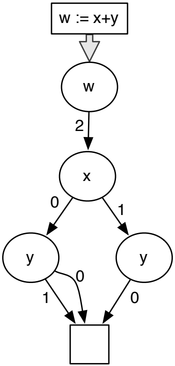


- no pre-computation
- no bound needed
- dynamic support reduction
- what if variables in wrong order?

Towards New Operations on DD

Variables in "wrong" order

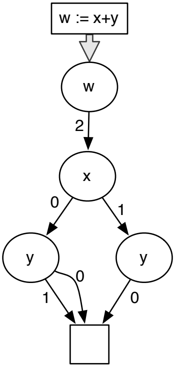
$w := x + y$



Towards New Operations on DD

Variables in "wrong" order

$w := x + y$

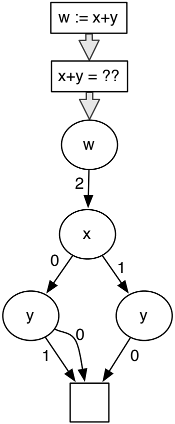


- equivalence classes w.r.t. the value of $x + y$
- $\mathcal{O}(|codomain|)$ instead of $\mathcal{O}(|set|)$

Towards New Operations on DD

Variables in "wrong" order

$w := x + y$

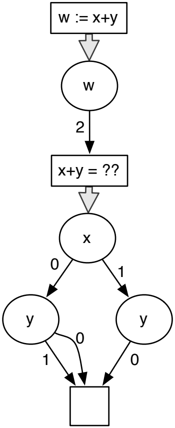


- equivalence classes w.r.t. the value of $x + y$
- $\mathcal{O}(|codomain|)$ instead of $\mathcal{O}(|set|)$

Towards New Operations on DD

Variables in "wrong" order

$w := x + y$

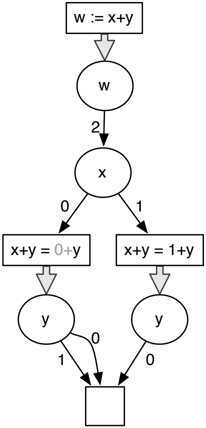


- equivalence classes w.r.t. the value of $x + y$
- $\mathcal{O}(|codomain|)$ instead of $\mathcal{O}(|set|)$

Towards New Operations on DD

Variables in "wrong" order

$w := x + y$

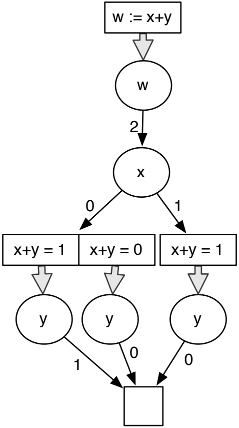


- equivalence classes w.r.t. the value of $x + y$
- $\mathcal{O}(|codomain|)$ instead of $\mathcal{O}(|set|)$
- refine

Towards New Operations on DD

Variables in “wrong” order

$w := x + y$

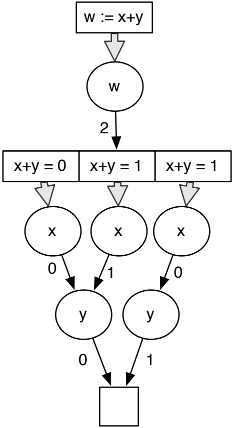


- equivalence classes w.r.t. the value of $x + y$
- $\mathcal{O}(|codomain|)$ instead of $\mathcal{O}(|set|)$
- refine

Towards New Operations on DD

Variables in “wrong” order

$w := x + y$

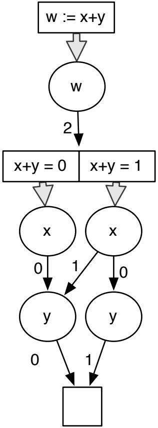


- equivalence classes w.r.t. the value of $x + y$
- $\mathcal{O}(|codomain|)$ instead of $\mathcal{O}(|set|)$
- refine

Towards New Operations on DD

Variables in “wrong” order

$w := x + y$

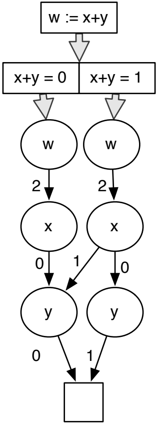


- equivalence classes w.r.t. the value of $x + y$
- $\mathcal{O}(|codomain|)$ instead of $\mathcal{O}(|set|)$
- refine
- merge

Towards New Operations on DD

Variables in “wrong” order

$w := x + y$

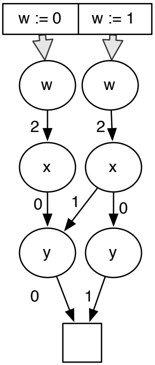


- equivalence classes w.r.t. the value of $x + y$
- $\mathcal{O}(|codomain|)$ instead of $\mathcal{O}(|set|)$
- refine
- merge

Towards New Operations on DD

Variables in "wrong" order

$w := x + y$

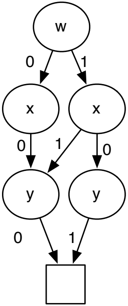


- equivalence classes w.r.t. the value of $x + y$
- $\mathcal{O}(|codomain|)$ instead of $\mathcal{O}(|set|)$
- refine
- merge
- constant assignment on each obtained subset

Towards New Operations on DD

Variables in “wrong” order

$w := x + y$



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

EquivSplit for Complex Operations

Evaluate high-level assignments

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

Easy case: ϕ is a constant address.
Use EquivSplit to evaluate ψ
On each subset, assign the value of ψ to the address ϕ

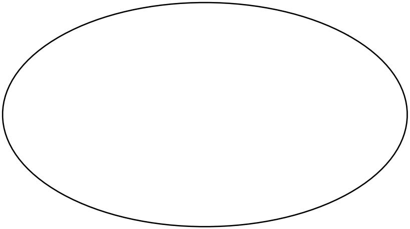
EquivSplit for Complex Operations

Evaluate high-level assignments

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

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

$\phi := \psi$

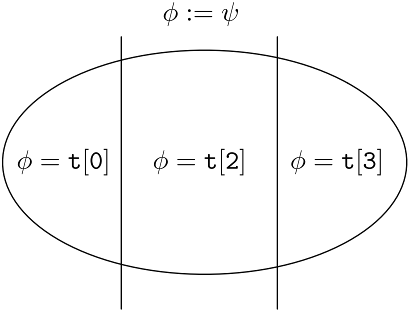


EquivSplit for Complex Operations

Evaluate high-level assignments

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

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



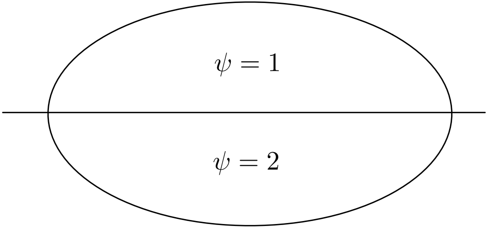
EquivSplit for Complex Operations

Evaluate high-level assignments

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

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

$\phi := \psi$



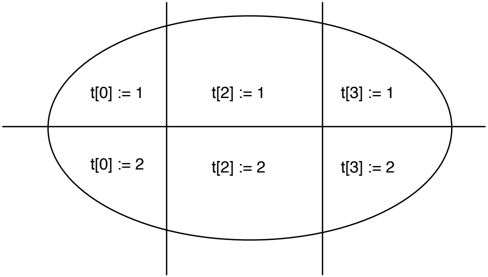
EquivSplit for Complex Operations

Evaluate high-level assignments

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

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

$$\mathtt{t}[\mathtt{x}+\mathtt{y}] := \mathtt{z} * \mathtt{x} + 1$$



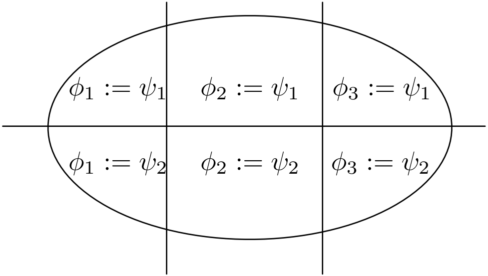
EquivSplit for Complex Operations

Evaluate high-level assignments

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

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

$\phi := \psi$



Experimental Validation

Benchmark

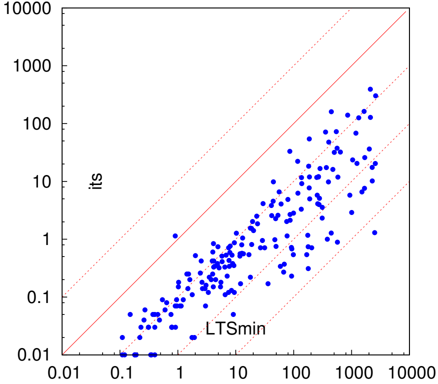
BEEM benchmark $\sim$ 400 instances

Comparison with

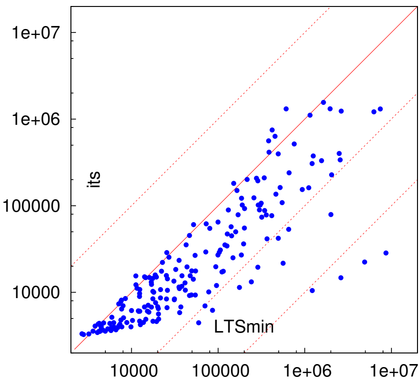
- LTSmin [Blom et al., 2010] explicit/symbolic model-checker
  - state space generation
  - 1 core, 10GB, 1hour
- super_prove [Berkeley LSV Group, 2012] SAT solver
  - winner of the HWMCC (FMCAD event) since 2010
  - reachability problems
  - 4cores, 1Gb, 15min wall-clock-time
  - NB: super_prove multi-thread, 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)



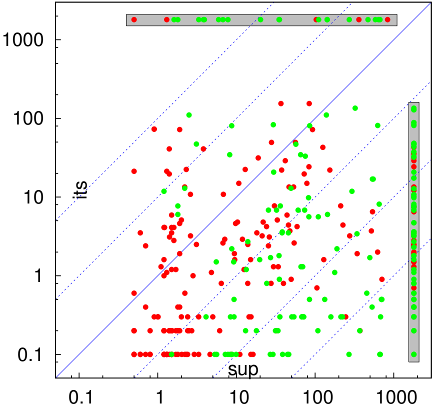
Comparison in memory (kb)

Comparison with super_prove

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

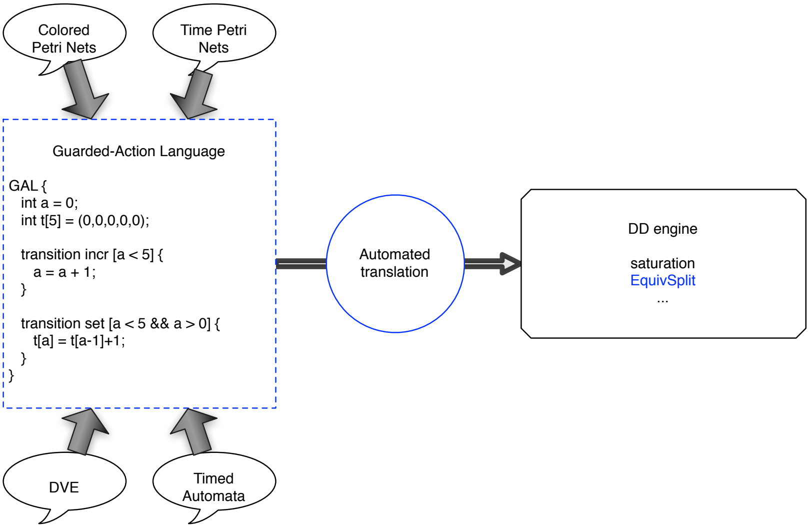
UNSAT
SAT

instances	456		
its solves	376	192	184
sup solves	282	170	112
solved by both	258	165	93
solved by none	56		



Comparison in time (s)

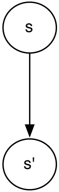
Abstract the Symbolic Engine from the User



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

Finite Transition Systems and Symmetries

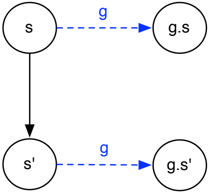
finite TS $\mathcal{K} = (S, \rightarrow \subseteq S \times S)$



Finite Transition Systems and Symmetries

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

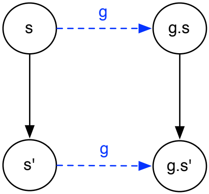
$g : S \mapsto S$ bijective is a symmetry iff:
$\forall s, s' \in S, s \rightarrow s' \iff g.s \rightarrow g.s'$



Finite Transition Systems and Symmetries

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

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

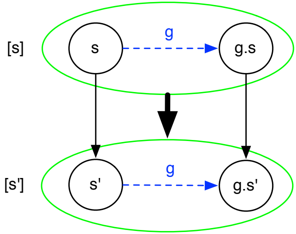


Finite Transition Systems and Symmetries

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



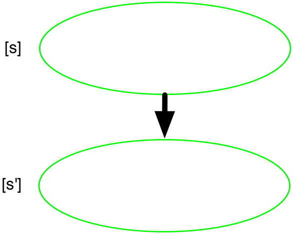
Finite Transition Systems and Symmetries

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

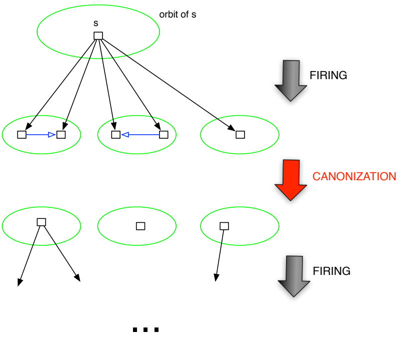
- $S \subsetneq \mathbb{N}^k$
  states = integer vectors of size k
- $G \subseteq \mathfrak{S}(k)$
  symmetries permute positions in the vectors

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

Orbit representation problem

Two ways to represent an orbit

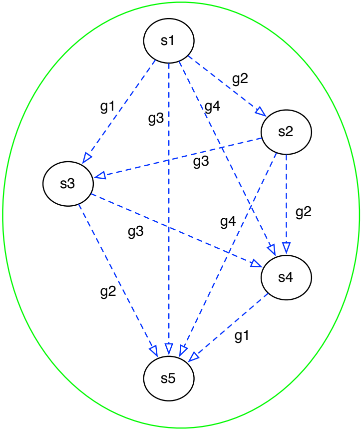
- use a dedicated representation [Chiola et al., 1990]
  - requires to adapt the transition relation
- choose one or several *representative* states in the orbit [Clarke et al., 1996]
  - the transition relation can be used as is



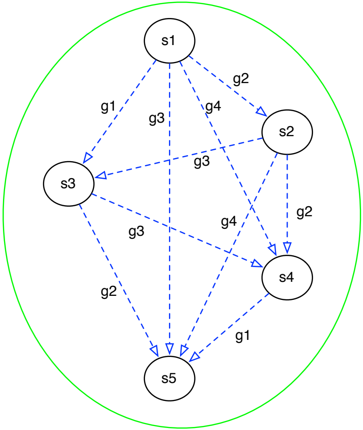
finding representatives = canonization

less representatives
=
harder canonization
=
smaller graph

How to represent an orbit symbolically?

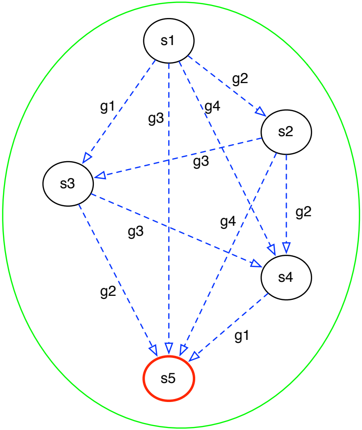


How to represent an orbit symbolically?



$\Rightarrow$ choose a representative state per orbit

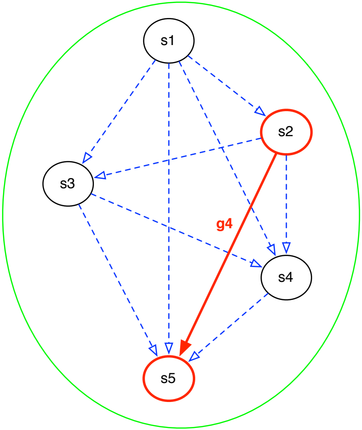
How to represent an orbit symbolically?



$\Rightarrow$ choose a representative state per orbit

- for instance, given a total order on S, choose the minimum
  - lexicographic order
  - e.g. $s1 > s2 > s3 > s4 > s5$

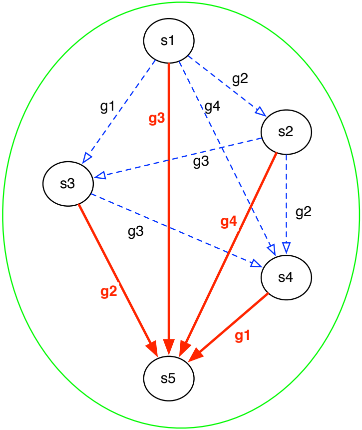
How to represent an orbit symbolically?



Current problems on canonization

- GRAPH ISOMORPHISM
- repeated for each new encountered state (state-by-state algorithms)
  - [Junttila, 2003]

How to represent an orbit symbolically?



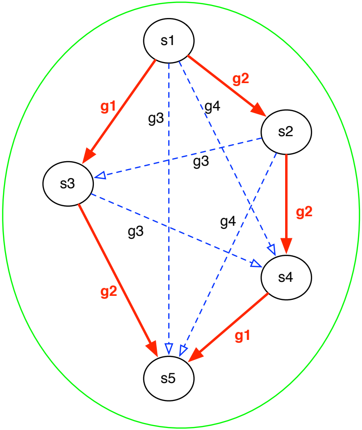
[Clarke et al., 1996]

orbit relation maps every potential state to its representative

$\Delta_{orbit} = \{(s, repr(s)) | s \in S\}$
exponential size
$\rightarrow_{quotient} = \rightarrow \circ \Delta_{orbit}$

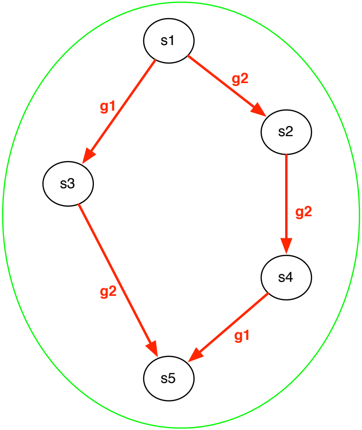
still a state-by-state algorithm

Our symbolic algorithm for canonization



But the red paths all lead to this minimum

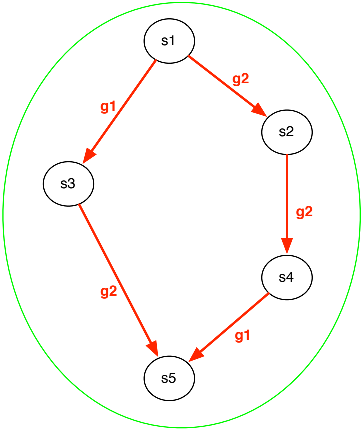
Our symbolic algorithm for canonization



But the red paths all lead to this minimum

Canonization can be done iteratively only through $g1$ and $g2$: represent only a subset of G

Our symbolic algorithm for canonization

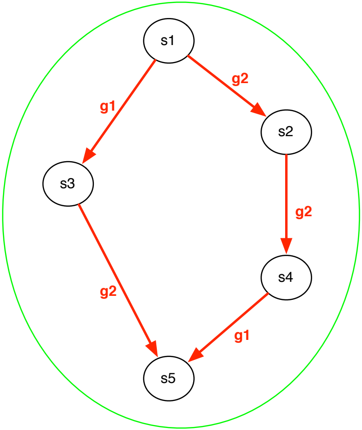


But the red paths all lead to this minimum

Canonization can be done iteratively only through $g1$ and $g2$: represent only a subset of G

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

Our symbolic algorithm for canonization



But the red paths all lead to this minimum

Canonization can be done iteratively only through $g1$ and $g2$: 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 Δ_H^*

A Note on Complexity

Any H is correct!

Whatever the chosen H, our algo Δ_H^* approximates Δ_{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 Δ_H
- number of representatives depends on H

Choice of H

$\Delta^*_H = \Delta_{orbit}$ (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 $G = \mathfrak{S}(k)$, 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

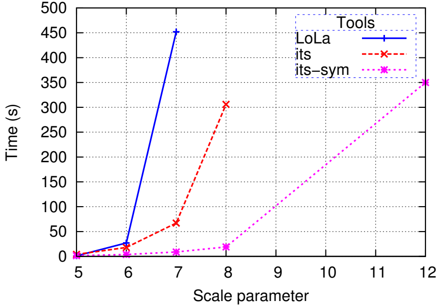
Benchmarks

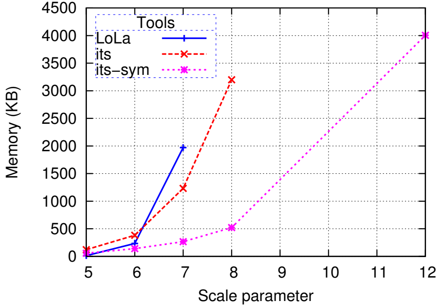
Tools	symmetry	DD
LoLA	✓	
its		✓
its-sym	✓	✓

its-sym extends its $\rightarrow$ same DD implementation

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

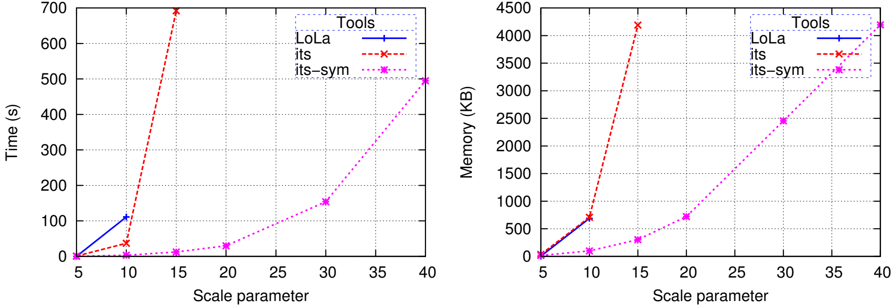
Benchmarks





Clients servers model

Benchmarks



SaleStore model

Conclusion

Operations on DD

- original fully symbolic algorithm for evaluating arbitrary expressions
  - based on partitionning 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!

Implemented! `http://ddd.lip6.fr/`

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

Berkeley LSV Group (2012).
Abc: A System for Sequential Synthesis and Verification, release 12/10/06.
http://www.eecs.berkeley.edu/~alanmi/abc/.

Blom, S., van de Pol, J., and Weber, M. (2010).
Ltsmin: Distributed and symbolic reachability.
In *Computer Aided Verification*, pages 354–359. Springer.

Bryant, R. E. (1986).
Graph-based algorithms for boolean function manipulation.
Computers, IEEE Transactions on, 100(8):677–691.

Burch, J. R., Clarke, E. M., McMillan, K. L., Dill, D. L., and Hwang, L. (1992).
Symbolic model checking: 10^{20} States and beyond.
Information and computation, 98(2):142–170.

Chiola, G., Dutheillet, C., Franceschinis, G., and Haddad, S. (1990).
On well-formed coloured nets and their symbolic reachability graph.
In *11th International Conference on Application and Theory of Petri Nets*.

Clarke, E. M., Enders, R., Filkorn, T., and Jha, S. (1996).
Exploiting symmetry in temporal logic model checking.
Formal Methods in System Design, 9(1):77–104.

Bibliography II

Couvreur, J.-M., Encrenaz, E., Paviot-Adet, E., Poitrenaud, D., and Wacrenier, P. (2002).
Data decision diagrams for petri net analysis.
Application and Theory of Petri Nets 2002, pages 129–158.

Haddad, S., Ilié, J. M., Taghelit, M., and Zouari, B. (1995).
Symbolic reachability graph and partial symmetries.
In *Application and Theory of Petri Nets 1995*, pages 238–257. Springer.

Junttila, T. (2003).
On the symmetry reduction method for Petri Nets and similar formalisms.
PhD thesis, Helsinki University of Technology, Espoo, Finland.

Srinivasan, A., Ham, T., Malik, S., and Brayton, R. K. (1990).
Algorithms for discrete function manipulation.
In *Computer-Aided Design, 1990. ICCAD-90. Digest of Technical Papers., 1990 IEEE International Conference on*, pages 92–95. IEEE.

My Papers

Colange, M., Baarir, S., Kordon, F., and Thierry-Mieg, Y. (2011).
Crocodile: a symbolic/symbolic tool for the analysis of symmetric nets with bag.
Applications and Theory of Petri Nets, pages 338–347.

Colange, M., Baarir, S., Kordon, F., and Thierry-Mieg, Y. (2013).
Towards Distributed Software Model-Checking using Decision Diagrams.
In *25th International Conference on Computer Aided Verification (CAV)*, volume 8044 of *Lecture Notes in Computer Science*, pages 830–845. Springer Verlag.

Colange, M., Hillah, L. M., Kordon, F., and Parutto, P. (2012a).
Extreme Symmetries in Complex Distributed Systems: the Bag-Oriented Approach.
In *Development, Operation and Management of Large-Scale Complex IT Systems, 17th Monterey Workshop, Revised Selected Papers*, volume 7539 of *LNCS*, pages 330–352. Springer.

Colange, M., Kordon, F., Thierry-Mieg, Y., and Baarir, S. (2012b).
State Space Analysis using Symmetries on Decision Diagrams.
In *12th International Conference on Application of Concurrency to System Design (ACSD'2012)*, pages 164–172, Hamburg, Germany. IEEE Computer Society.