Symbolic Model Checking of High-Level Petri Nets

Didier Buchs Steve Hostettler Matteo Risoldi

Université de Genève

2 décembre 2015

Outline

Outline

Outline

Outline

- PART I : Model Checking
- PART II : Symbolic Model Checking
- PART III : AlPiNA : Basics
- PART IV : AlPiNA : Advanced

Outline

Outline

- PART I : Model Checking
- PART II : Symbolic Model Checking
- PART III : AlPiNA : Basics
- PART IV : AlPiNA : Advanced

Outline

Outline

- PART I : Model Checking
- PART II : Symbolic Model Checking
- PART III : AlPiNA : Basics
- PART IV : AlPiNA : Advanced

Outline

Outline

- PART I : Model Checking
- PART II : Symbolic Model Checking
- PART III : AlPiNA : Basics
- PART IV : AlPiNA : Advanced

- Ask questions during Q&A.
- Bibliography at the end of the slides.

Outline

Outline

- PART I : Model Checking
- PART II : Symbolic Model Checking
- PART III : AlPiNA : Basics
- PART IV : AlPiNA : Advanced

- Ask questions during Q&A.
- Bibliography at the end of the slides.

PART I : Model Checking

Model Checking

Model checking is a way to automatically prove that :

Definition

$$\mathbf{M}_{\{\mathbf{s_0}\}} \models \mathbf{\Phi} \text{ where :}$$

- $\mathbf{M}_{\{s_0\}}$ is Kripke structure with s_0 as initial state
- $\mathbf{\Phi}$ is a property expressed in a temporal logic

- Does the Kripke structure $\mathbf{M}$ model the specification $\mathbf{\Phi}$?
- Original idea by Clarke & Emerson [13] and Queille & Sifakis [27]

Model Checking

Model checking is a way to automatically prove that :

Definition

$$\mathbf{M}_{\{\mathbf{s_0}\}} \models \mathbf{\Phi} \text{ where :}$$

- $\mathbf{M}_{\{\mathbf{s_0}\}}$ is Kripke structure with s_0 as initial state
- $\mathbf{\Phi}$ is a property expressed in a temporal logic

- Does the Kripke structure $\mathbf{M}$ model the specification $\mathbf{\Phi}$?
- Original idea by Clarke & Emerson [13] and Queille & Sifakis [27]

Model Checking

Model checking is a way to automatically prove that :

Definition

$$\mathbf{M}_{\{s_0\}} \models \mathbf{\Phi} \text{ where :}$$

- $\mathbf{M}_{\{s_0\}}$ is Kripke structure with s_0 as initial state
- $\mathbf{\Phi}$ is a property expressed in a temporal logic

- Does the Kripke structure $\mathbf{M}$ model the specification $\mathbf{\Phi}$?
- Original idea by Clarke & Emerson [13] and Queille & Sifakis [27]

Model Checking

Model checking is a way to automatically prove that :

Definition

$$\mathbf{M}_{\{\mathbf{s_0}\}} \models \mathbf{\Phi} \text{ where :}$$

- $\mathbf{M}_{\{\mathbf{s_0}\}}$ is Kripke structure with s_0 as initial state
- $\mathbf{\Phi}$ is a property expressed in a temporal logic

- Does the Kripke structure **M** model the specification $\mathbf{\Phi}$?
- Original idea by Clarke & Emerson [13] and Queille & Sifakis [27]

Model Checking

Model checking is a way to automatically prove that :

Definition

$$\mathbf{M}_{\{s_0\}} \models \mathbf{\Phi} \text{ where :}$$

- $\mathbf{M}_{\{s_0\}}$ is Kripke structure with s_0 as initial state
- $\mathbf{\Phi}$ is a property expressed in a temporal logic

- Does the Kripke structure $\mathbf{M}$ model the specification $\mathbf{\Phi}$?
- Original idea by Clarke & Emerson [13] and Queille & Sifakis [27]

Kripke Structure

Definition

A Kripke structure of a set of atomic propositions AP is a tuple $K = \langle S, S_0, R, L \rangle$ where :

- **S** is a finite set of states
- $\mathbf{S_0} \subseteq \mathbf{S}$ is a non-empty set of initial states
- $\mathbf{R} \subseteq \mathbf{S} \times \mathbf{S}$ is a left-total binary relation on S representing the transitions
- $\mathbf{L} : \mathbf{S} \rightarrow \mathcal{P}(\mathbf{AP})$ labels each state with a set of atomic propositions that hold on that state

Kripke Structure

Definition

A Kripke structure of a set of atomic propositions AP is a tuple $K = \langle S, S_0, R, L \rangle$ where :

- $\mathbf{S}$ is a finite set of states
- $\mathbf{S_0} \subseteq \mathbf{S}$ is a non-empty set of initial states
- $\mathbf{R} \subseteq \mathbf{S} \times \mathbf{S}$ is a left-total binary relation on S representing the transitions
- $\mathbf{L} : \mathbf{S} \rightarrow \mathcal{P}(\mathbf{AP})$ labels each state with a set of atomic propositions that hold on that state

Kripke Structure

Definition

A Kripke structure of a set of atomic propositions AP is a tuple $K = \langle S, S_0, R, L \rangle$ where :

- $\mathbf{S}$ is a finite set of states
- $\mathbf{S_0} \subseteq \mathbf{S}$ is a non-empty set of initial states
- $\mathbf{R} \subseteq \mathbf{S} \times \mathbf{S}$ is a left-total binary relation on S representing the transitions
- $\mathbf{L} : \mathbf{S} \rightarrow \mathcal{P}(\mathbf{AP})$ labels each state with a set of atomic propositions that hold on that state

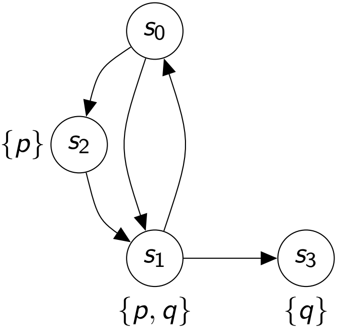
Kripke Structure

Definition

A Kripke structure of a set of atomic propositions AP is a tuple $K = \langle S, S_0, R, L \rangle$ where :

- $\mathbf{S}$ is a finite set of states
- $\mathbf{S_0} \subseteq \mathbf{S}$ is a non-empty set of initial states
- $\mathbf{R} \subseteq \mathbf{S} \times \mathbf{S}$ is a left-total binary relation on S representing the transitions
- $\mathbf{L} : \mathbf{S} \rightarrow \mathcal{P}(\mathbf{AP})$ labels each state with a set of atomic propositions that hold on that state

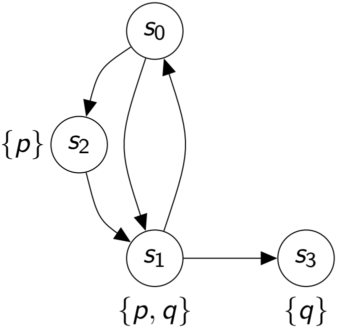
Kripke Structure (example)



$K = \langle S, S_0, R, L \rangle$ where :

- $S = \{s_0, s_1, s_2, s_3\}$
- $S_0 = \{s_0\}$
- $R = \{(s_0, s_1), (s_1, s_0),$ $(s_0, s_2), (s_2, s_1), (s_1, s_3)\}$
- $L(s_0) = \{\emptyset\}, L(s_1) = \{p, q\},$ $L(s_2) = \{p\}, L(s_3) = \{q\}$

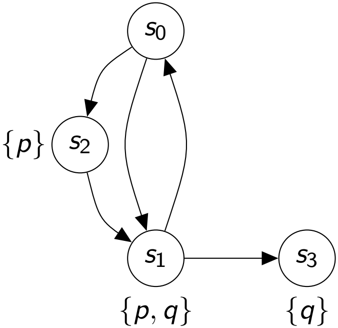
Kripke Structure (example)



$K = \langle S, S_0, R, L \rangle$ where :

- $S = \{s_0, s_1, s_2, s_3\}$
- $S_0 = \{s_0\}$
- $R = \{(s_0, s_1), (s_1, s_0), (s_0, s_2), (s_2, s_1), (s_1, s_3)\}$
- $L(s_0) = \{\emptyset\}, L(s_1) = \{p, q\}, L(s_2) = \{p\}, L(s_3) = \{q\}$

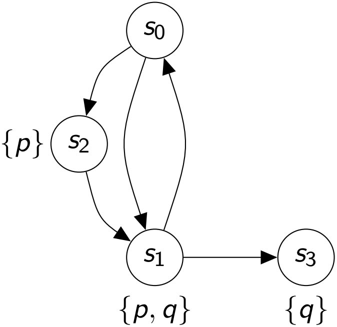
Kripke Structure (example)



$K = \langle S, S_0, R, L \rangle$ where :

- $S = \{s_0, s_1, s_2, s_3\}$
- $S_0 = \{s_0\}$
- $R = \{(s_0, s_1), (s_1, s_0),$ $(s_0, s_2), (s_2, s_1), (s_1, s_3)\}$
- $L(s_0) = \{\emptyset\}, L(s_1) = \{p, q\},$ $L(s_2) = \{p\}, L(s_3) = \{q\}$

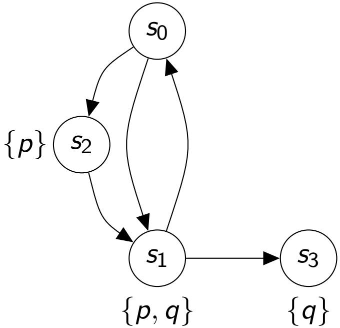
Kripke Structure (example)



$K = \langle S, S_0, R, L \rangle$ where :

- $S = \{s_0, s_1, s_2, s_3\}$
- $S_0 = \{s_0\}$
- $R = \{(s_0, s_1), (s_1, s_0),$
  $(s_0, s_2), (s_2, s_1), (s_1, s_3)\}$
- $L(s_0) = \{\emptyset\}, L(s_1) = \{p, q\},$
  $L(s_2) = \{p\}, L(s_3) = \{q\}$

Kripke Structure (example)



$K = \langle S, S_0, R, L \rangle$ where :

- $S = \{s_0, s_1, s_2, s_3\}$
- $S_0 = \{s_0\}$
- $R = \{(s_0, s_1), (s_1, s_0), (s_0, s_2), (s_2, s_1), (s_1, s_3)\}$
- $L(s_0) = \{\emptyset\}, L(s_1) = \{p, q\}, L(s_2) = \{p\}, L(s_3) = \{q\}$

Temporal logics

- Extends the propositional logic (which is good for "static" situations)
- To express linear and deterministic (or branching and non-deterministic) progression
- Several temporal logics :
  - LTL : Linear temporal logic : Linear progression of time [26]
  - CTL : Computational tree logic : branching progression [13]
  - $(LTL \cup CTL) \subset CTL^*$ [18]

Temporal logics

- Extends the propositional logic (which is good for “static” situations)
- To express linear and deterministic (or branching and non-deterministic) progression
- Several temporal logics :
  - LTL : Linear temporal logic : Linear progression of time [26]
  - CTL : Computational tree logic : branching progression [13]
  - (LTL ∪ CTL) ⊂ CTL* [18]

Temporal logics

- Extends the propositional logic (which is good for “static” situations)
- To express linear and deterministic (or branching and non-deterministic) progression
- Several temporal logics :
  - *LTL* : Linear temporal logic : Linear progression of time [26]
  - *CTL* : Computational tree logic : branching progression [13]
  - $(LTL \cup CTL) \subset CTL^*$ [18]

Temporal logics

- Extends the propositional logic (which is good for “static” situations)
- To express linear and deterministic (or branching and non-deterministic) progression
- Several temporal logics :
  - *LTL* : Linear temporal logic : Linear progression of time [26]
  - *CTL* : Computational tree logic : branching progression [13]
  - $(LTL \cup CTL) \subset CTL^*$ [18]

Temporal logics

- Extends the propositional logic (which is good for "static" situations)
- To express linear and deterministic (or branching and non-deterministic) progression
- Several temporal logics :
  - *LTL* : Linear temporal logic : Linear progression of time [26]
  - *CTL* : Computational tree logic : branching progression [13]
  - $(LTL \cup CTL) \subset CTL^*$ [18]

Computational tree logic [13] (1)

- $\phi ::= \bot | \top | p | (\neg\phi) | (\phi \wedge \phi) | (\phi \vee \phi) | (\phi \Rightarrow \phi) | (\phi \Leftrightarrow \phi) |$ $AX\phi | EX\phi | AF\phi \; | EF\phi | AG\phi | EG\phi | A[\phi U \phi] | E[\phi U \phi]$
- p is an atomic formula. Namely a predicate applied to a tuple of terms : an atomic formula is a formula of the form $P(t_1, \ldots, t_n)$ for P a predicate, and the t_k terms with $1 \leq k \leq n$.
- Terms are of the form $t ::= c \mid x \mid f(t_1, ..., t_n)$, with c a constant, x a variable and f is a n-ary function whose arguments are terms.

Computational tree logic [13] (1)

- $\phi ::= \bot | \top | p | (\neg\phi) | (\phi \wedge \phi) | (\phi \vee \phi) | (\phi \Rightarrow \phi) | (\phi \Leftrightarrow \phi) |$ $AX\phi | EX\phi | AF\phi\ | EF\phi | AG\phi | EG\phi | A[\phi U \phi] | E[\phi U \phi]$
- p is an atomic formula. Namely a predicate applied to a tuple of terms : an atomic formula is a formula of the form $P(t_1, \ldots, t_n)$ for P a predicate, and the t_k terms with $1 \leq k \leq n$.
- Terms are of the form $t ::= c \mid x \mid f(t_1, ..., t_n)$, with c a constant, x a variable and f is a n-ary function whose arguments are terms.

Computational tree logic [13] (1)

- $\phi ::= \bot | \top | p | (\neg\phi) | (\phi \wedge \phi) | (\phi \vee \phi) | (\phi \Rightarrow \phi) | (\phi \Leftrightarrow \phi) |$ $AX\phi | EX\phi | AF\phi \ | EF\phi | AG\phi | EG\phi | A[\phi U \phi] | E[\phi U \phi]$
- p is an atomic formula. Namely a predicate applied to a tuple of terms : an atomic formula is a formula of the form $P(t_1, \ldots, t_n)$ for P a predicate, and the t_k terms with $1 \leq k \leq n$.
- Terms are of the form $t ::= c \mid x \mid f(t_1, ..., t_n)$, with c a constant, x a variable and f is a n-ary function whose arguments are terms.

CTL* [18]

- Combines temporal operators with quantification over runs and apply them to atomic properties (ϕ, ψ).
- Quantifiers :
  - A : ϕ holds for (A)ll executions
  - E : there (E)xists an execution in which ϕ holds
- Temporal operators :
  - X : ϕ holds at the ne(X)t state
  - F : (F)inally ϕ holds somewhere on the subsequent path
  - G : ϕ holds (G)lobally on the subsequent path
  - U : ϕ holds (U)ntil ψ holds

CTL* [18]

- Combines temporal operators with quantification over runs and apply them to atomic properties (ϕ, ψ).
- Quantifiers :
  - **A** : ϕ holds for (**A**)ll executions
  - **E** : there (**E**)xists an execution in which ϕ holds.
- Temporal operators :
  - **X** : ϕ holds at the ne(X)t state.
  - **F** : (F)inally ϕ holds somewhere on the subsequent path
  - **G** : ϕ holds (G)lobally on the subsequent path
  - **U** : ϕ holds (U)ntil ψ holds

CTL* [18]

- Combines temporal operators with quantification over runs and apply them to atomic properties (ϕ, ψ).
- Quantifiers :
  - **A** : ϕ holds for (**A**)ll executions
  - **E** : there (**E**)xists an execution in which ϕ holds.
- Temporal operators :
  - **X** : ϕ holds in the ne(**X**)t state
  - **F** : (**F**)inally ϕ holds somewhere on the subsequent path
  - **G** : ϕ holds (**G**)lobally on the subsequent path
  - **U** : ϕ holds (**U**)ntil ψ holds

CTL* [18]

- Combines temporal operators with quantification over runs and apply them to atomic properties (ϕ, ψ).
- Quantifiers :
  - **A** : ϕ holds for (**A**)ll executions
  - **E** : there (**E**)xists an execution in which ϕ holds.
- Temporal operators :
  - X : ϕ holds at the ne(X)t state
  - F : (F)inally ϕ holds somewhere on the subsequent path
  - G : ϕ holds (G)lobally on the subsequent path
  - U : ϕ holds (U)ntil ψ holds

CTL* [18]

- Combines temporal operators with quantification over runs and apply them to atomic properties (ϕ, ψ).
- Quantifiers :
  - **A** : ϕ holds for (**A**)ll executions
  - **E** : there (**E**)xists an execution in which ϕ holds.
- Temporal operators :
  - **X** : ϕ holds at the ne(**X**)t state.
  - **F** : (**F**)inally ϕ holds somewhere on the subsequent path
  - **G** : ϕ holds (**G**)lobally on the subsequent path
  - **U** : ϕ holds (**U**)ntil ψ holds.

CTL* [18]

- Combines temporal operators with quantification over runs and apply them to atomic properties (ϕ, ψ).
- Quantifiers :
  - **A** : ϕ holds for (**A**)ll executions
  - **E** : there (**E**)xists an execution in which ϕ holds.
- Temporal operators :
  - **X** : ϕ holds at the ne(**X**)t state.
  - **F** : (**F**)inally ϕ holds somewhere on the subsequent path
  - **G** : ϕ holds (**G**)lobally on the subsequent path
  - **U** : ϕ holds (**U**)ntil ψ holds.

CTL* [18]

- Combines temporal operators with quantification over runs and apply them to atomic properties (ϕ, ψ).
- Quantifiers :
  - **A** : ϕ holds for (**A**)ll executions
  - **E** : there (**E**)xists an execution in which ϕ holds.
- Temporal operators :
  - **X** : ϕ holds at the ne(**X**)t state.
  - **F** : (**F**)inally ϕ holds somewhere on the subsequent path
  - **G** : ϕ holds (**G**)lobally on the subsequent path
  - **U** : ϕ holds (**U**)ntil ψ holds.

CTL* [18]

- Combines temporal operators with quantification over runs and apply them to atomic properties (ϕ, ψ).
- Quantifiers :
  - **A** : ϕ holds for (**A**)ll executions
  - **E** : there (**E**)xists an execution in which ϕ holds.
- Temporal operators :
  - **X** : ϕ holds at the ne(**X**)t state.
  - **F** : (**F**)inally ϕ holds somewhere on the subsequent path
  - **G** : ϕ holds (**G**)lobally on the subsequent path
  - **U** : ϕ holds (**U**)ntil ψ holds.

CTL* [18]

- Combines temporal operators with quantification over runs and apply them to atomic properties (ϕ, ψ).
- Quantifiers :
  - **A** : ϕ holds for (**A**)ll executions
  - **E** : there (**E**)xists an execution in which ϕ holds.
- Temporal operators :
  - **X** : ϕ holds at the ne(**X**)t state.
  - **F** : (**F**)inally ϕ holds somewhere on the subsequent path
  - **G** : ϕ holds (**G**)lobally on the subsequent path
  - **U** : ϕ holds (**U**)ntil ψ holds.

Q & A

- Is a CTL/LTL property a model ?
- Express "there is an execution in which p holds until q holds" using CTL.
- What does *AFAGp* mean ?
- Are the arcs of Kripke structure labelled ?

Q & A

- Is a CTL/LTL property a model ?
- Express "there is an execution in which p holds until q holds" using CTL.
- What does *AFAGp* mean ?
- Are the arcs of Kripke structure labelled ?

Q & A

- Is a CTL/LTL property a model ?
- Express “there is an execution in which p holds until q holds” using CTL.
- What does *AFAGp* mean ?
- Are the arcs of Kripke structure labelled ?

Q & A

- Is a CTL/LTL property a model ?
- Express "there is an execution in which p holds until q holds" using CTL.
- What does $AFAGp$ mean ?
- Are the arcs of Kripke structure labelled ?

Remarks

Problematic

- How to obtain the Kripke structure ?
- Manipulating the Kripke structure can be very hard.

Solution

User manipulates a model that is a symbolic representation (with clear syntax and semantics) of the Kripke structure. The actual Kripke structure is only produced at runtime.

Remarks

Problematic

- How to obtain the Kripke structure ?
- Manipulating the Kripke structure can be very hard.

Solution

User manipulates a model that is a symbolic representation (with clear syntax and semantics) of the Kripke structure. The actual Kripke structure is only produced at runtime.

Remarks

Problematic

- How to obtain the Kripke structure ?
- Manipulating the Kripke structure can be very hard.

Solution

User manipulates a model that is a symbolic representation (with clear syntax and semantics) of the Kripke structure. The actual Kripke structure is only produced at runtime.

Model Checking (Reloaded)

Model checking is a way to automatically prove that :

Definition

$$\mathbf{M}_{\{\mathbf{s_0}\}} \models \mathbf{\Phi} \text{ where :}$$

- $\mathbf{M}_{\{\mathbf{s_0}\}}$ is a **symbolic representation** of a Kripke structure with s_0 as initial state
- $\mathbf{\Phi}$ is a property expressed in a **temporal logic**

Warning

The symbolic representation must have clear semantics (Petri nets, automata, CCS, ...) to allow automatic generation of the Kripke structure.

Model Checking (Reloaded)

Model checking is a way to automatically prove that :

Definition

$$\mathbf{M}_{\{s_0\}} \models \mathbf{\Phi} \text{ where :}$$

- $\mathbf{M}_{\{s_0\}}$ is a **symbolic representation** of a Kripke structure with s_0 as initial state
- Φ is a property expressed in a **temporal logic**

Warning

The symbolic representation must have clear semantics (Petri nets, automata, CCS, ...) to allow automatic generation of the Kripke structure.

Model Checking (Reloaded)

Model checking is a way to automatically prove that :

Definition

$$\mathbf{M}_{\{s_0\}} \models \mathbf{\Phi} \text{ where :}$$

- $\mathbf{M}_{\{s_0\}}$ is a **symbolic representation** of a Kripke structure with s_0 as initial state
- $\mathbf{\Phi}$ is a property expressed in a **temporal logic**

Warning

The symbolic representation must have clear semantics (Petri nets, automata, CCS, ...) to allow automatic generation of the Kripke structure.

Model Checking (Reloaded)

Model checking is a way to automatically prove that :

Definition

$$\mathbf{M}_{\{s_0\}} \models \mathbf{\Phi} \text{ where :}$$

- $\mathbf{M}_{\{s_0\}}$ is a **symbolic representation** of a Kripke structure with s_0 as initial state
- $\mathbf{\Phi}$ is a property expressed in a **temporal logic**

Warning

The symbolic representation must have clear semantics (Petri nets, automata, CCS, . . .) to allow automatic generation of the Kripke structure.

Ok seems nice to me, what's the catch ?

Systems are often very complex

DSL : ✔

Formal languages are . . . formal

DSL : ✔

The infamous state space explosion problem [30]

? ? ? : ✖

Ok seems nice to me, what's the catch ?

Systems are often very complex

Formal languages are ... formal

The infamous state space explosion problem [30]

DSL : ✔

DSL : ✔

? ? ? : ✖

Ok seems nice to me, what's the catch ?

Systems are often very complex

DSL : ✔

Formal languages are . . . formal

DSL : ✔

The infamous state space explosion problem [30]

? ? ? : ✖

Ok seems nice to me, what's the catch ?

Systems are often very complex

Formal languages are . . . formal

The infamous state space explosion problem [30]

DSL : ✔

DSL : ✔

? ? ? : ✖

Ok seems nice to me, what's the catch ?

Systems are often very complex

Formal languages are . . . formal

The infamous state space explosion problem [30]

DSL : ✔

DSL : ✔

? ? ? : ✖

Ok seems nice to me, what's the catch ?

Systems are often very complex

Formal languages are . . . formal

The infamous state space explosion problem [30]

DSL : ✔

DSL : ✔

? ? ? : ✖

Ok seems nice to me, what's the catch ?

Systems are often very complex

Formal languages are ... formal

The infamous state space explosion problem [30]

DSL : ✔

DSL : ✔

? ? ? : ✖

The State Space Explosion (Example)

Dining philosopher problem

1 byte per state

$\Rightarrow$

200 philosophers

$2.5x10^{125}$ states
(# of atoms in the observable universe : 10^{80})

28421709400000000000000000000000
00000000000000000000000000000000

The State Space Explosion (Example)

Dining philosopher problem

1 byte per state

⇒

200 philosophers

2.5×10^{125} states
(# of atoms in the observable universe : 10^{80})

28421709400000000000000000000000
00000000000000000000000000000000

The State Space Explosion (Example)

Dining philosopher problem

1 byte per state

$\Rightarrow$

200 philosophers

2.5×10^{125} states
(# of atoms in the observable universe : 10^{80})

284217094000000000000000000000
000000000000000000000000000000

The State Space Explosion (Example)

Dining philosopher problem

1 byte per state

$\Rightarrow$

200 philosophers

$2.5x10^{125}$ states
(# of atoms in the observable universe : 10^{80})

28421709400000000000000000000000
00000000000000000000000000000000

The State Space Explosion (Example)

Dining philosopher problem

1 byte per state

200 philosophers

$\Rightarrow$

$2.5x10^{125}$ states
(# of atoms in the observable universe : 10^{80})

28421709400000000000000000000
00000000000000000000000000000

The State Space Explosion (Solutions)

- Reduce the search space
  - Partial orders (Model checking by representatives)
  - Abstractions
  - Symmetry based (Quotient graphs)
- Improve the computation and storing power
- Better state space representation
  - Symbolic approaches

The State Space Explosion (Solutions)

- Reduce the search space
  - Partial orders (Model checking by representatives)
  - Abstractions
  - Symmetry based (Quotient graphs)
- Improve the computation and storing power
- Better state space representation
  - Symbolic approaches

The State Space Explosion (Solutions)

- Reduce the search space
  - Partial orders (Model checking by representatives)
  - Abstractions
  - Symmetry based (Quotient graphs)
- Improve the computation and storing power
- Better state space representation
  - Symbolic approaches

The State Space Explosion (Solutions)

- Reduce the search space
  - Partial orders (Model checking by representatives)
  - Abstractions
  - Symmetry based (Quotient graphs)
- Improve the computation and storing power
- Better state space representation
  - Symbolic approaches

The State Space Explosion (Solutions)

- Reduce the search space
  - Partial orders (Model checking by representatives)
  - Abstractions
  - Symmetry based (Quotient graphs)
- Improve the computation and storing power
- Better state space representation
  - Symbolic approaches

The State Space Explosion (Solutions)

- Reduce the search space
  - Partial orders (Model checking by representatives)
  - Abstractions
  - Symmetry based (Quotient graphs)
- Improve the computation and storing power
- Better state space representation
  - Symbolic approaches

The State Space Explosion (Solutions)

- Reduce the search space
  - Partial orders (Model checking by representatives)
  - Abstractions
  - Symmetry based (Quotient graphs)
- Improve the computation and storing power
- Better state space representation
  - Symbolic approaches

Reducing the search space : Partial Orders (1)

- Avoid storing and checking paths that are considered equivalent w.r.t the property to check.
- Exploit the commutativity of concurrently executed transitions, which result in the same state when executed in different orders (diamond property) [20, 21].
- Different techniques (ample sets [24], stubborn sets [29])

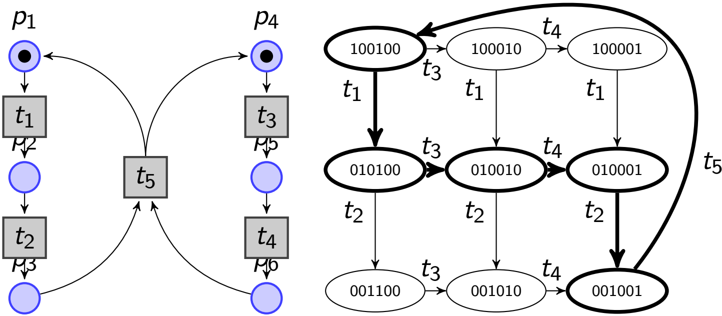
Reducing the search space : Partial Orders (1)

- Avoid storing and checking paths that are considered equivalent w.r.t the property to check.
- Exploit the commutativity of concurrently executed transitions, which result in the same state when executed in different orders (diamond property) [20, 21].
- Different techniques (ample sets [24], stubborn sets [29])

Reducing the search space : Partial Orders (1)

- Avoid storing and checking paths that are considered equivalent w.r.t the property to check.
- Exploit the commutativity of concurrently executed transitions, which result in the same state when executed in different orders (diamond property) [20, 21].
- Different techniques (ample sets [24], stubborn sets [29])

Reducing the search space : Partial Orders (2)



Reducing the search space : Abstractions

- Big models
- Make a reduced model according to what one wants to check, while preserving interesting properties
- Check the counter-example against the complete model and refine the abstraction accordingly
- CEGAR approach [22, 19] (Counter Example Guided Abstraction Refinement)

Reducing the search space : Abstractions

- Big models
- Make a reduced model according to what one wants to check, while preserving interesting properties
- Check the counter-example against the complete model and refine the abstraction accordingly
- CEGAR approach [22, 19] (Counter Example Guided Abstraction Refinement)

Reducing the search space : Abstractions

- Big models
- Make a reduced model according to what one wants to check, while preserving interesting properties
- Check the counter-example against the complete model and refine the abstraction accordingly
- CEGAR approach [22, 19] (Counter Example Guided Abstraction Refinement)

Reducing the search space : Abstractions

- Big models
- Make a reduced model according to what one wants to check, while preserving interesting properties
- Check the counter-example against the complete model and refine the abstraction accordingly
- CEGAR approach [22, 19] (Counter Example Guided Abstraction Refinement)

Reducing the search space : CEGAR

To verify $C \models AGp$ do

1. 1. build finite Kripke structure $A > C$ an abstraction (homomorphic),
2. 2. model-check $A \models AGp$,
3. 3. if this holds then report $C \models AGp$ and stop,
4. 4. otherwise validate the counterexample on C, i.e., find a corresponding concrete counterexample,
5. 5. if a corresponding concrete counterexample exists then report $C \not\models AGp$ and stop,
6. 6. otherwise use the spurious counterexample to refine A and restart from 2.

Reducing the search space : Quotient graphs (1)

- Exploit symmetries between the states
- Define equivalence relation on the state space
- Bisimulation equivalent to the original model
- Efficiency is highly dependent on the system (exponential at best, no reduction at worst)

Reducing the search space : Quotient graphs (1)

- Exploit symmetries between the states
- Define equivalence relation on the state space
- Bisimulation equivalent to the original model
- Efficiency is highly dependent on the system (exponential at best, no reduction at worst)

Reducing the search space : Quotient graphs (1)

- Exploit symmetries between the states
- Define equivalence relation on the state space
- Bisimulation equivalent to the original model
- Efficiency is highly dependent on the system (exponential at best, no reduction at worst)

Reducing the search space : Quotient graphs (1)

- Exploit symmetries between the states
- Define equivalence relation on the state space
- Bisimulation equivalent to the original model
- Efficiency is highly dependent on the system (exponential at best, no reduction at worst)

Reducing the search space : Quotient graphs (2)

Example

- Let's consider a system in which two clients (c_1 and c_2) communicate with a server.
- c_1 and c_2 have a identical behaviour.
- Instead of considering the state s_1 in which the client c_1 sends a message to the server and another state in which the client c_2 does the same, we consider the equivalent state s' in which one of the client sends a message.

Reducing the search space : Quotient graphs (2)

Example

- Let's consider a system in which two clients (c_1 and c_2) communicate with a server.
- c_1 and c_2 have a identical behaviour.
- Instead of considering the state s_1 in which the client c_1 sends a message to the server and another state in which the client c_2 does the same, we consider the equivalent state s' in which one of the client sends a message.

Reducing the search space : Quotient graphs (2)

Example

- Let's consider a system in which two clients (c_1 and c_2) communicate with a server.
- c_1 and c_2 have a identical behaviour.
- Instead of considering the state s_1 in which the client c_1 sends a message to the server and another state in which the client c_2 does the same, we consider the equivalent state s' in which one of the client sends a message.

Better state space representation : Symbolism

Two approaches :

Representation of symbolic states

Define equivalence classes between states and perform check on the classes

Symbolic representation of the states

Efficient representation of all the states using dedicated data structures (BDD [9], MDD [11], DDD [16]. . .) based on their similarities [9]

Better state space representation : Symbolism

Two approaches :

Representation of symbolic states

Define equivalence classes between states and perform check on the classes

Symbolic representation of the states

Efficient representation of all the states using dedicated data structures (BDD [9], MDD [11], DDD [16]. . .) based on their similarities [9]

Better state space representation : Symbolism

Two approaches :

Representation of symbolic states

Define equivalence classes between states and perform check on the classes

Symbolic representation of the states

Efficient representation of all the states using dedicated data structures (BDD [9], MDD [11], DDD [16]. . .) based on their similarities [9]

Q & A

- What are the main reasons of the state space explosion ?
- Compare the “quotient graph” approach and the “partial order” approach.
- Compare the two kind of symbolic approaches.

Q & A

- What are the main reasons of the state space explosion ?
- Compare the “quotient graph” approach and the “partial order” approach.
- Compare the two kind of symbolic approaches.

Q & A

- What are the main reasons of the state space explosion ?
- Compare the “quotient graph” approach and the “partial order” approach.
- Compare the two kind of symbolic approaches.

PART II : Symbolic Model Checking

using a symbolic representaton of the states

Symbolic Model Checking

Ideas

- Symbolic encoding of the Kripke structure [9, 23, 14, 15].
- Represent a set of states instead of only one.
- Representation based on ROBDD [4].

Symbolic Model Checking

Ideas

- Symbolic encoding of the Kripke structure [9, 23, 14, 15].
- Represent a set of states instead of only one.
- Representation based on ROBDD [4].

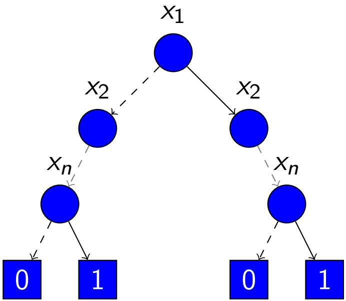
Symbolic Model Checking

Ideas

- Symbolic encoding of the Kripke structure [9, 23, 14, 15].
- Represent a set of states instead of only one.
- Representation based on ROBDD [4].

Binary Decision Trees

- Are trees that represent both a Boolean expression and its solutions. Based on the Shannon Expansion theorem : $F(x1, x2, .., xn) = x1.F(x2, .., xn) + \overline{x1}.F(x2, .., xn)$



Reduced Ordered Binary Decision Diagrams

ROBDD

- Reduced Ordered Binary Decision Diagrams provide a compact representation of Boolean functions.
- Directed Acyclic Graph.
- Reduced : Compress the representation.
- Ordered : Variable order is set ⇒ "dont'care" reduction.
- Mostly used in problem solving and model checking.

Reduced Ordered Binary Decision Diagrams

ROBDD

- Reduced Ordered Binary Decision Diagrams provide a compact representation of Boolean functions.
- Directed Acyclic Graph.
- Reduced : Compress the representation.
- Ordered : Variable order is set $\Rightarrow$ "dont'care" reduction.
- Mostly used in problem solving and model checking.

Reduced Ordered Binary Decision Diagrams

ROBDD

- Reduced Ordered Binary Decision Diagrams provide a compact representation of Boolean functions.
- Directed Acyclic Graph.
- Reduced : Compress the representation.
- Ordered : Variable order is set ⇒ "dont'care" reduction.
- Mostly used in problem solving and model checking.

Reduced Ordered Binary Decision Diagrams

ROBDD

- Reduced Ordered Binary Decision Diagrams provide a compact representation of Boolean functions.
- Directed Acyclic Graph.
- Reduced : Compress the representation.
- Ordered : Variable order is set $\Rightarrow$ "dont'care" reduction.
- Mostly used in problem solving and model checking.

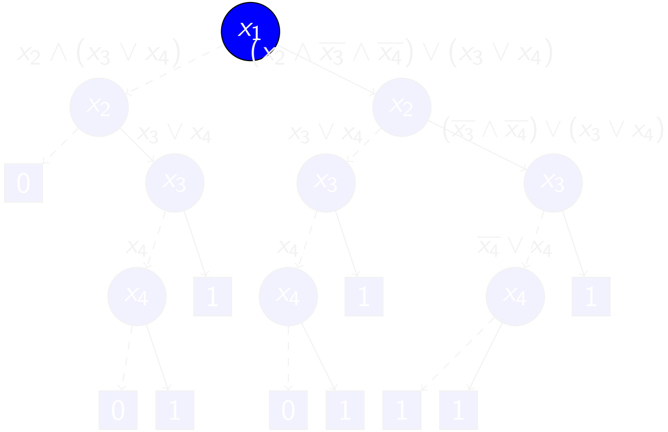
Reduced Ordered Binary Decision Diagrams

ROBDD

- Reduced Ordered Binary Decision Diagrams provide a compact representation of Boolean functions.
- Directed Acyclic Graph.
- Reduced : Compress the representation.
- Ordered : Variable order is set ⇒ "dont'care" reduction.
- Mostly used in problem solving and model checking.

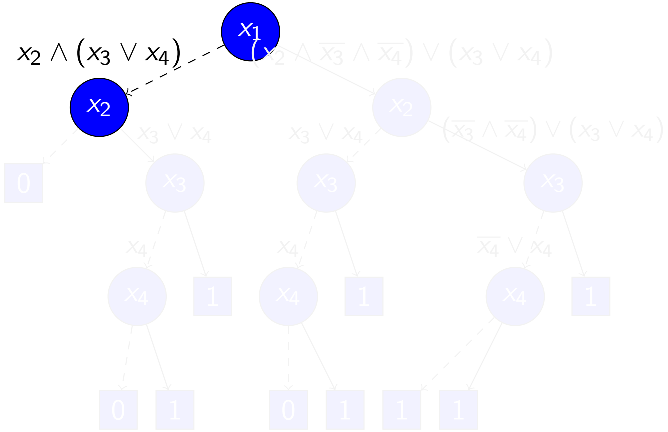
ROBDD Example : Build the BDD

$$(x_1 \wedge x_2 \wedge \overline{x_3} \wedge \overline{x_4}) \vee (x_1 \vee x_2) \wedge (x_3 \vee x_4)$$

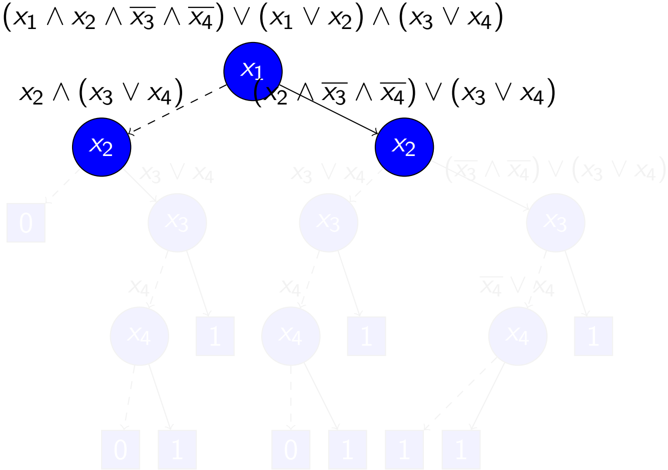


ROBDD Example : Build the BDD

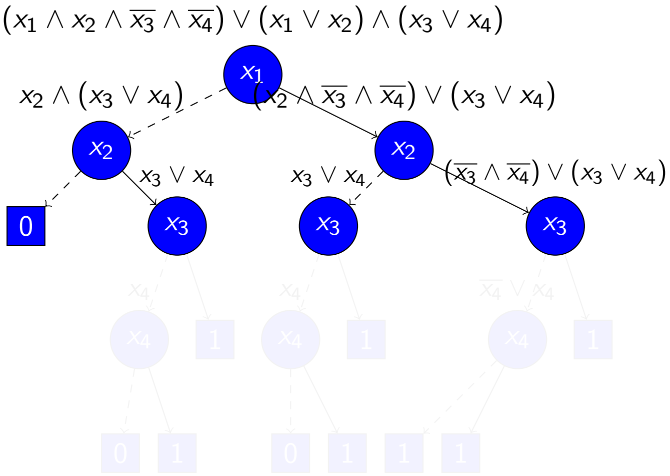
$$(x_1 \wedge x_2 \wedge \overline{x_3} \wedge \overline{x_4}) \vee (x_1 \vee x_2) \wedge (x_3 \vee x_4)$$



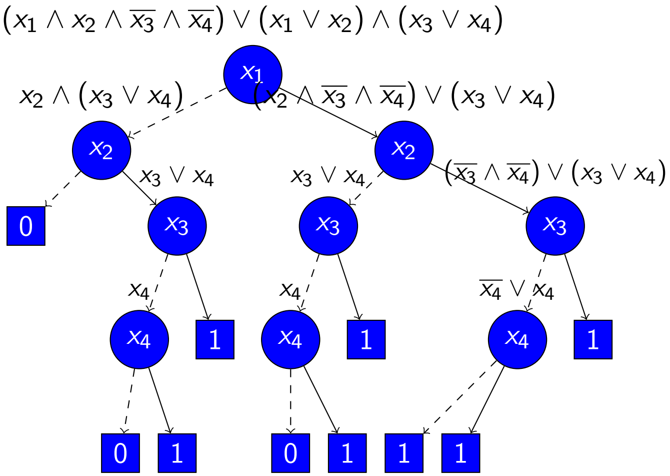
ROBDD Example : Build the BDD



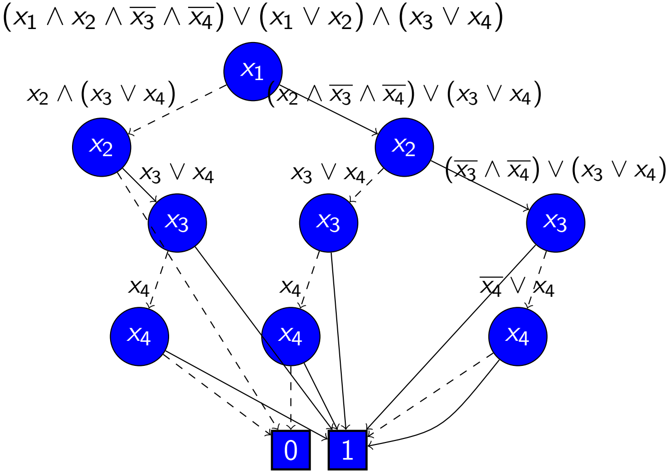
ROBDD Example : Build the BDD



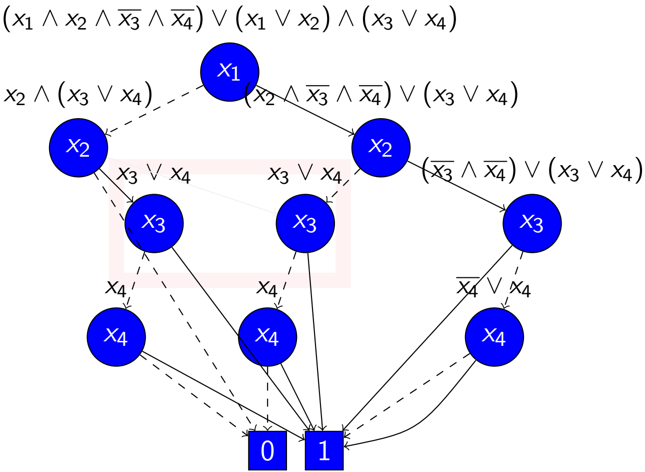
ROBDD Example : Build the BDD



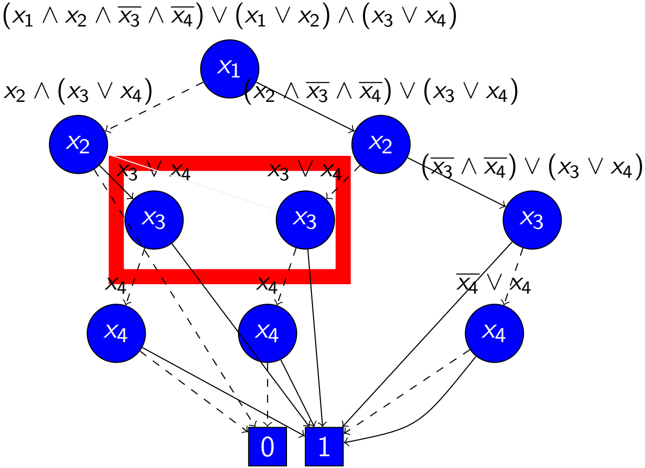
ROBDD Example : Factorize terminals



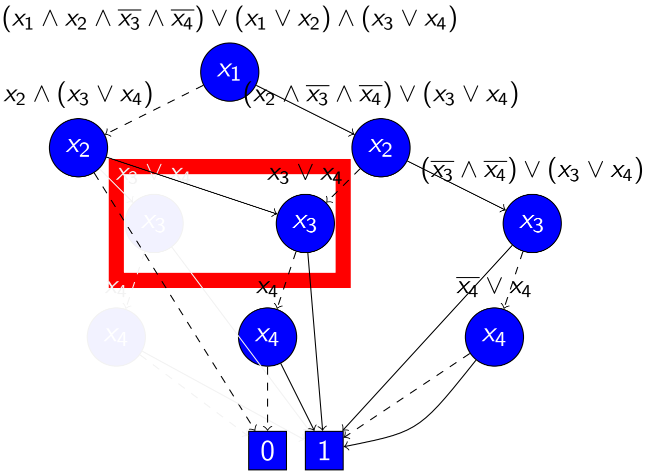
ROBDD Example : Factorize nodes



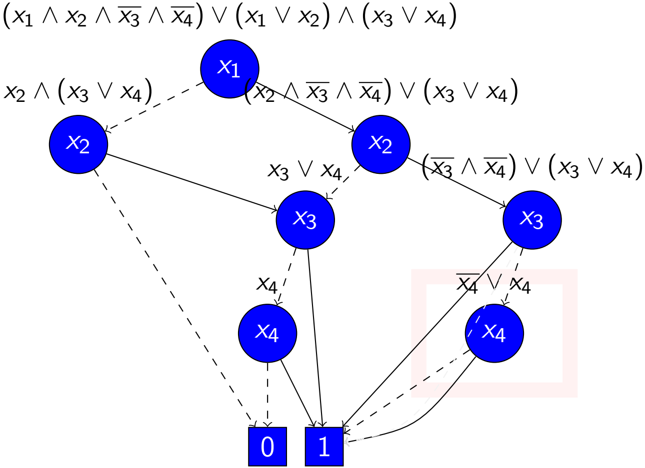
ROBDD Example : Factorize nodes



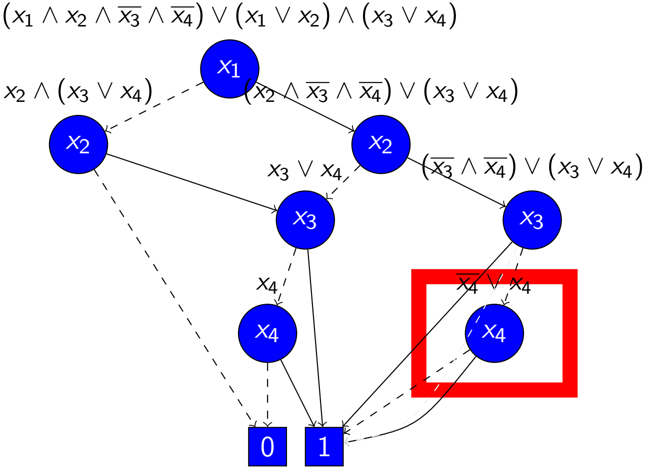
ROBDD Example : Factorize nodes



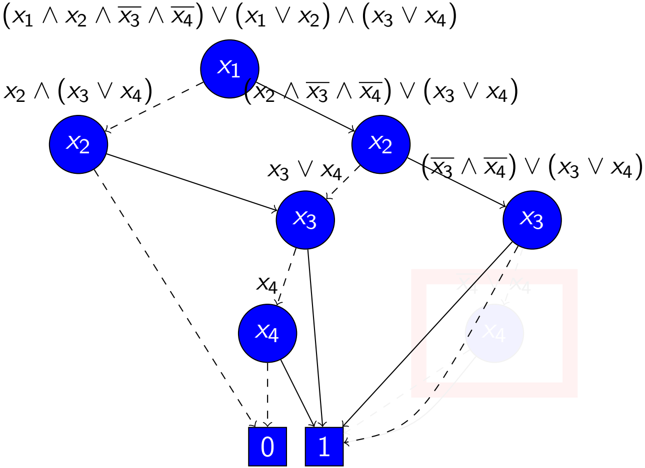
ROBDD Example : Remove useless nodes (Don't care)



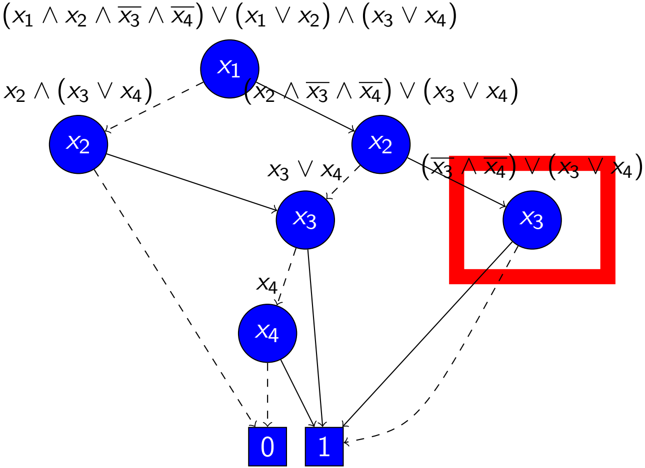
ROBDD Example : Remove useless nodes (Don't care)



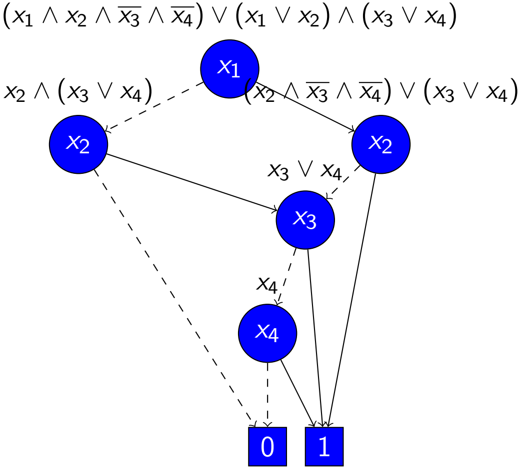
ROBDD Example : Remove useless nodes (Don't care)



ROBDD Example : Remove useless nodes (Don't care)



ROBDD Example : Final result



Reduce BDD on the fly

- Isomorphic subgraphs are factorized.
- Remove node x if $f(x) = f(\overline{x})$:
- Uses memoization [3].

Reduce BDD on the fly

- Isomorphic subgraphs are factorized.
- Remove node x if $f(x) = f(\overline{x})$:
- Uses memoization [3].

Reduce BDD on the fly

- Isomorphic subgraphs are factorized.
- Remove node x if $f(x) = f(\overline{x})$:
- Uses memoization [3].

ROBDD : Properties

- Given a variable ordering, ROBDD are canonical (proof by induction on the size of the variable set)
- Hash-consing (fly-weight pattern) [2, 1]
- Operations are cached (Memoization) [3]

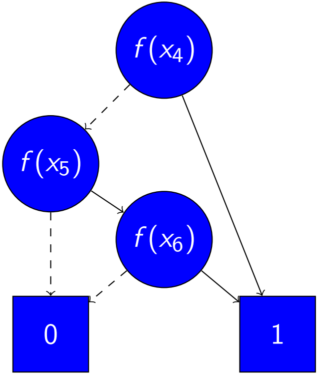
ROBDD : Properties

- Given a variable ordering, ROBDD are canonical (proof by induction on the size of the variable set)
- Hash-consing (fly-weight pattern) [2, 1]
- Operations are cached (Memoization) [3]

ROBDD : Properties

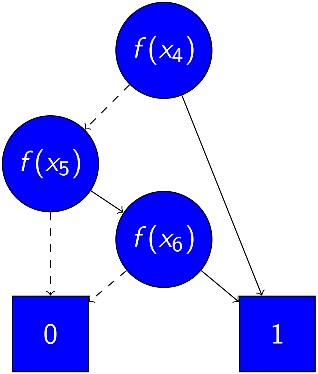
- Given a variable ordering, ROBDD are canonical (proof by induction on the size of the variable set)
- Hash-consing (fly-weight pattern) [2, 1]
- Operations are cached (Memoization) [3]

ROBDD : Memoization



$$f(x_4) = x$$
$$f(x_5) = y$$
$$f(x_6) = z$$

ROBDD : Memoization

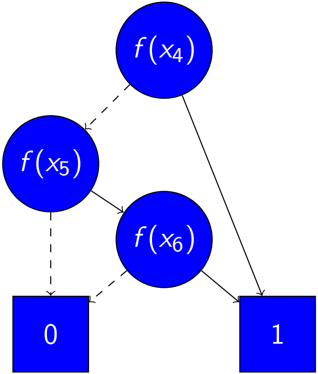


$$f(x_4) = x$$

$$f(x_5) = y$$

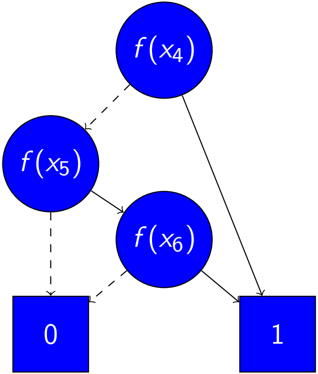
$$f(x_6) = z$$

ROBDD : Memoization



$$f(x_4) = x$$
$$f(x_5) = y$$
$$f(x_6) = z$$

ROBDD : Memoization



$$f(x_4) = x$$
$$f(x_5) = y$$
$$f(x_6) = z$$

ROBDD (2)

Complexity

- Let K be the number of variables in the ROBDD
- Best case complexity : $O(K)$.
- Worst case complexity : $O(2^K)$.
- Test whether a function is constantly true/false in $O(1)$.

ROBDD (2)

Complexity

- Let K be the number of variables in the ROBDD
- Best case complexity : $O(K)$.
- Worst case complexity : $O(2^K)$.
- Test whether a function is constantly true/false in $O(1)$.

ROBDD (2)

Complexity

- Let K be the number of variables in the ROBDD
- Best case complexity : $O(K)$.
- Worst case complexity : $O(2^K)$.
- Test whether a function is constantly true/false in $O(1)$.

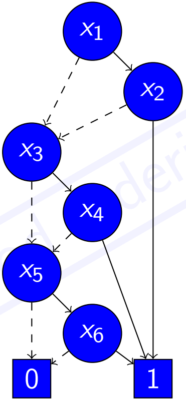
ROBDD (2)

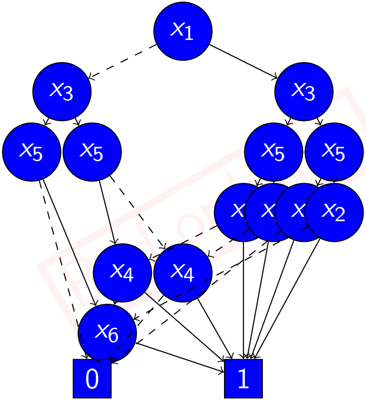
Complexity

- Let K be the number of variables in the ROBDD
- Best case complexity : $O(K)$.
- Worst case complexity : $O(2^K)$.
- Test whether a function is constantly true/false in $O(1)$.

Ordering (1)

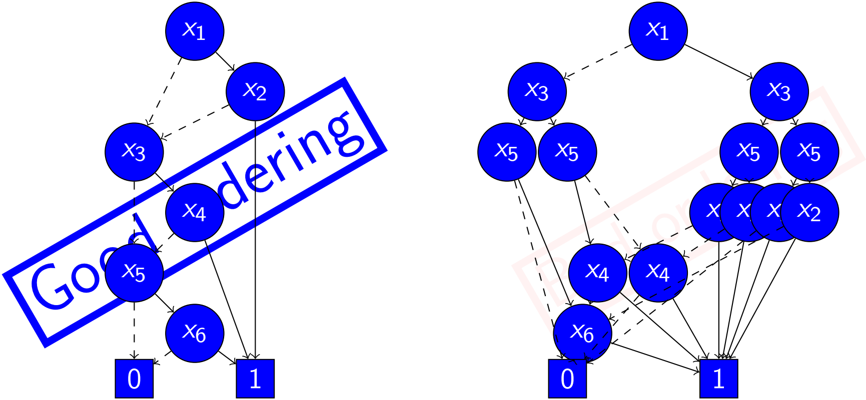
$$(x_1 \wedge x_2) \vee (x_3 \wedge x_4) \vee (x_5 \wedge x_6)$$





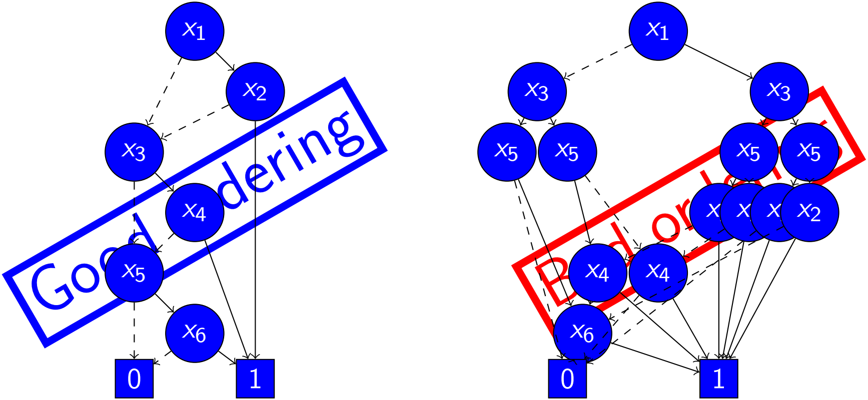
Ordering (1)

$$(x_1 \land x_2) \lor (x_3 \land x_4) \lor (x_5 \land x_6)$$



Ordering (1)

$$(x_1 \wedge x_2) \vee (x_3 \wedge x_4) \vee (x_5 \wedge x_6)$$



Ordering (2)

Ordering

- Is an NP complete problem.
- Dynamic optimization : Reorganized ordering on the fly.
- Heuristics.

Ordering (2)

Ordering

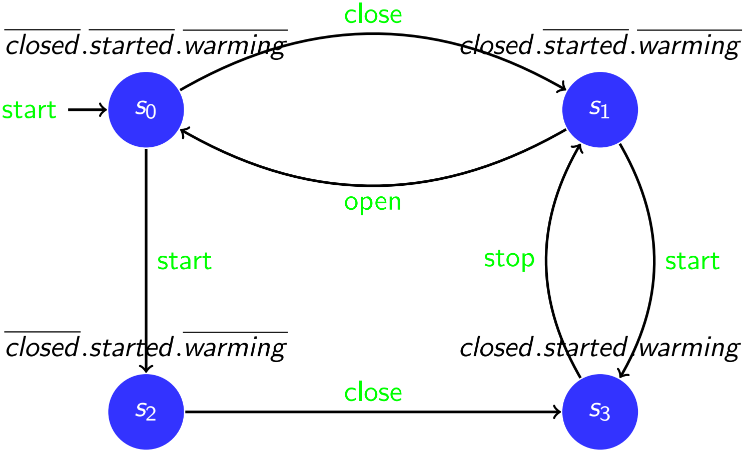
- Is an NP complete problem.
- Dynamic optimization : Reorganized ordering on the fly.
- Heuristics.

Ordering (2)

Ordering

- Is an NP complete problem.
- Dynamic optimization : Reorganized ordering on the fly.
- Heuristics.

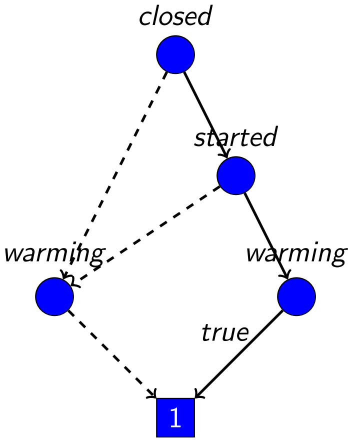
Symbolic Model Checking : Example (1)



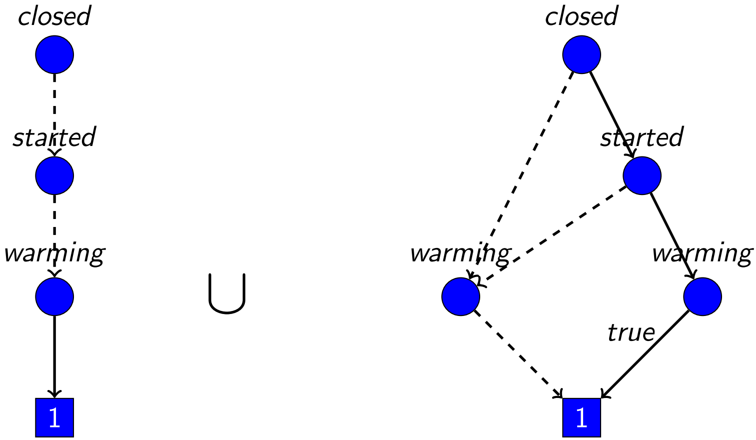
Symbolic Model Checking : Example (2)

Encode states on 3 bits :

$\overline{closed}.\overline{started}.\overline{warming} +$
$closed.\overline{started}.\overline{warming} +$
$closed.started.warming +$
$\overline{closed}.started.\overline{warming}$



Symbolic Model Checking : OR as union



Symbolic Model Checking : Example (3)

- The transition relation is encoded by interlacing the pre and post state.
- A good variable ordering is : $x_1 Old < x_1 New < x_2 Old < x_2 New < \ldots < x_i Old < x_i New$.

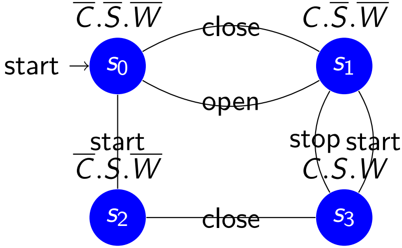
Symbolic Model Checking : Example (3)

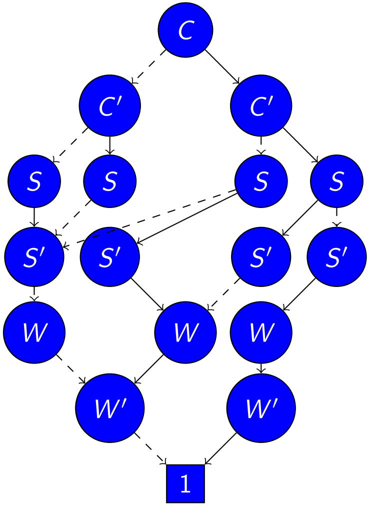
- The transition relation is encoded by interlacing the pre and post state.
- A good variable ordering is :
  $x_1 Old < x_1 New < x_2 Old < x_2 New < \ldots < x_i Old < x_i New$.

The τ and the τ^{-1} functions

Example of the encoding of τ

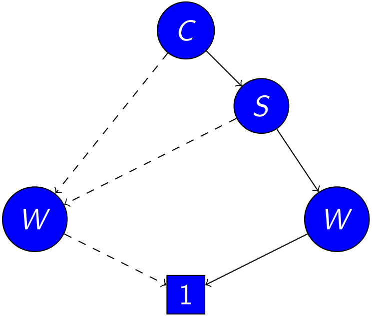
$$
\begin{aligned}
&\overline{closed}.closed'.\overline{started}.\overline{started'}.\overline{warming}.\overline{warming'} + \\
&closed.\overline{closed'}.\overline{started}.\overline{started'}.\overline{warming}.\overline{warming'} + \\
&closed.closed'.\overline{started}.started'.\overline{warming}.warming + \\
&closed.closed'.started.\overline{started'}.warming.\overline{warming'} + \\
&\overline{closed}.\overline{closed'}.\overline{started}.started'.\overline{warming}.\overline{warming'} + \\
&\overline{closed}.closed'.started.started'.\overline{warming}.warming'
\end{aligned}
$$





Symbolic Model Checking : Example (5)

Let's compute the set of states that satisfy $EX(\neg warming)$. That is the set of states of which the next state satisfy $\neg warming$.

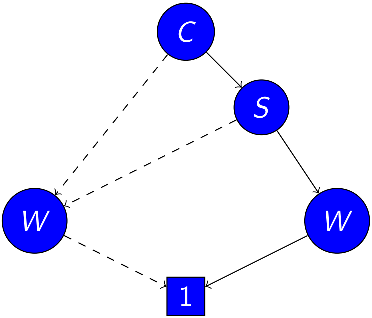


Full state space

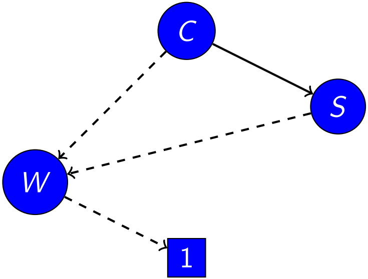
$\neg warming$

Symbolic Model Checking : Example (5)

Let's compute the set of states that satisfy $EX(\neg warming)$. That is the set of states of which the next state satisfy $\neg warming$.



Full state space

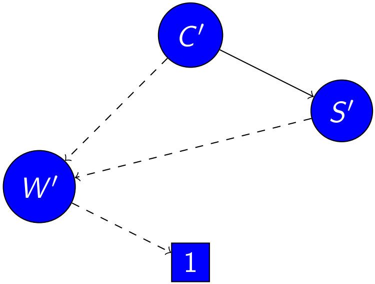


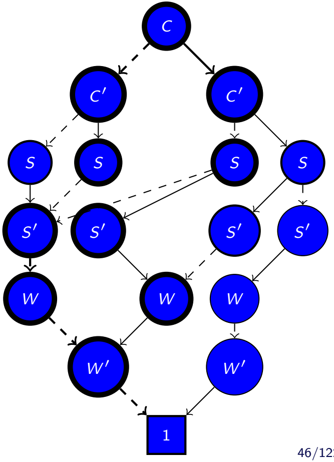
$\neg warming$

Symbolic Model Checking : Example (6)

How to compute the states ?

$\tau^{-1} \cap \neg \mathit{warning}$

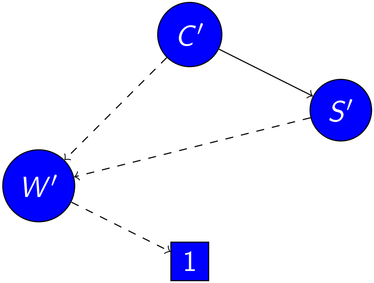


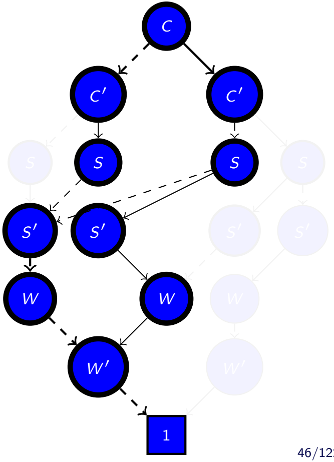


Symbolic Model Checking : Example (6)

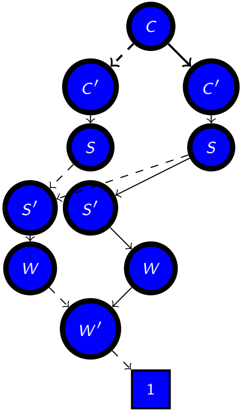
How to compute the states ?

$\tau^{-1} \cap \neg warning$





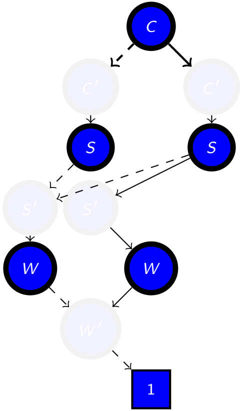
Symbolic Model Checking : Example (7)



How to compute the states?

Discard post variables

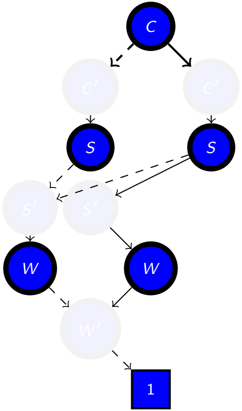
Symbolic Model Checking : Example (7)



How to compute the states ?

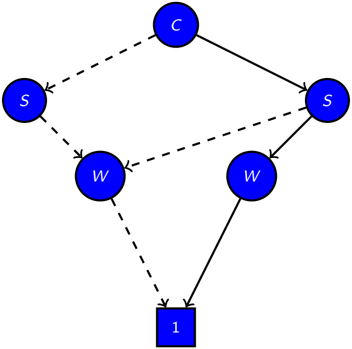
Discard post variables

Symbolic Model Checking : Example (7)



How to compute the states ?

Discard post variables



Q & A

- What is the particularity of the BDD based model checking ?
- Cite the main properties of the (RO)BDDs.
- Why are (RO)BDD not well suited to software model checking ?

Q & A

- What is the particularity of the BDD based model checking ?
- Cite the main properties of the (RO)BDDs.
- Why are (RO)BDD not well suited to software model checking ?

Q & A

- What is the particularity of the BDD based model checking ?
- Cite the main properties of the (RO)BDDs.
- Why are (RO)BDD not well suited to software model checking ?

Critic of the approach

Problem

BDD are not very efficient for complex types.

Solution

Improve the encoding to support complex types.

Critic of the approach

Problem

BDD are not very efficient for complex types.

Solution

Improve the encoding to support complex types.

Sets Decision Diagrams

Some properties

- Extension of the BDD.
- Very efficient operation from the set theory.
- Efficient encoding of the Cartesian product.
- Canonical representation of sets.
- Powerful caching mechanism.

Sets Decision Diagrams

Some properties

- Extension of the BDD.
- Very efficient operation from the set theory.
- Efficient encoding of the Cartesian product.
- Canonical representation of sets.
- Powerful caching mechanism.

Sets Decision Diagrams

Some properties

- Extension of the BDD.
- Very efficient operation from the set theory.
- Efficient encoding of the Cartesian product.
- Canonical representation of sets.
- Powerful caching mechanism.

Sets Decision Diagrams

Some properties

- Extension of the BDD.
- Very efficient operation from the set theory.
- Efficient encoding of the Cartesian product.
- Canonical representation of sets.
- Powerful caching mechanism.

Sets Decision Diagrams

Some properties

- Extension of the BDD.
- Very efficient operation from the set theory.
- Efficient encoding of the Cartesian product.
- Canonical representation of sets.
- Powerful caching mechanism.

SDD Example (1)

```java
import java.util.Arrays;
class MyObject {
    Integer a = 2;
    Character b = 'a';
    Integer[] c = new Integer[]{3,4};

    public void step1() {
        b = 'c';
        c[0] = 5;
    }
    public void step2() {
        b = 'a';
        c[0] = 6;
    }
    @Override
    public String toString() {
        return "a=" + a + "␣b='" + b + "'␣c=" + Arrays.toString(c);
    }

}
```

```java
public class Test {

    public static void main(String[] args) {
        MyObject o = new MyObject();
        System.out.println(o);
        o.step1();
        System.out.println(o);
        o.step2();
        System.out.println(o);
    }
}
```

SDD Example (2)

Initial state

- $a = 2$
- $b = `a`$
- $c = [3, 4]$

SDD Example (2)

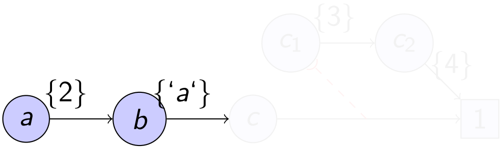
Initial state

- $a = 2$
- $b = \text{'}a\text{'}$
- $c = [3, 4]$

SDD Example (2)

Initial state

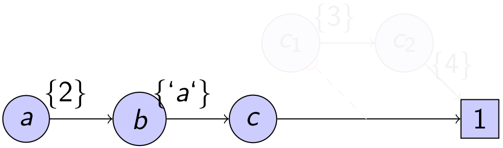
- $a = 2$
- $b = `a`$
- $c = [3, 4]$



SDD Example (2)

Initial state

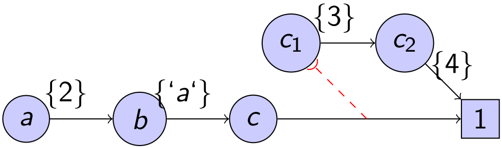
- $a = 2$
- $b = \text{‘}a\text{‘}$
- $c = [3, 4]$



SDD Example (2)

Initial state

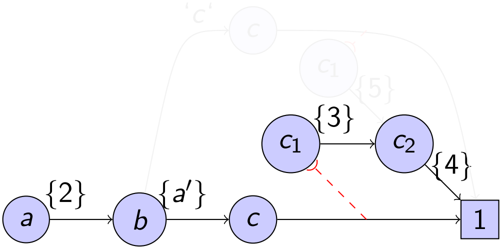
- $a = 2$
- $b = `a`$
- $c = [3, 4]$



SDD Example (3)

Step1(Initial state)

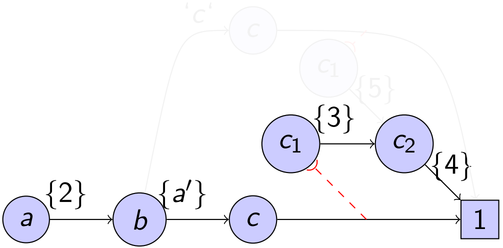
- $a = 2$
- $b = `c`$
- $c = [5, 4]$



SDD Example (3)

Step1(Initial state)

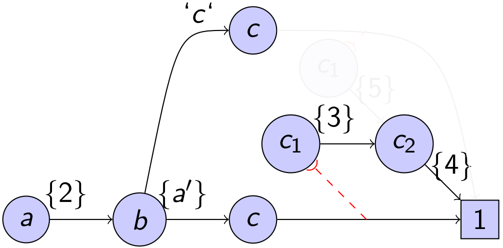
- $a = 2$
- $b = \text{‘}c\text{‘}$
- $c = [5, 4]$



SDD Example (3)

Step1(Initial state)

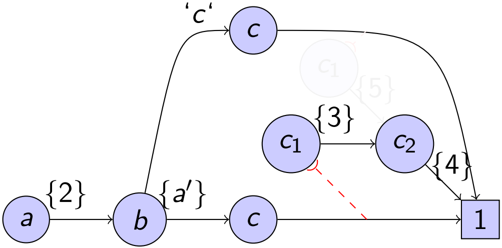
- $a = 2$
- $b = \text{‘}c\text{‘}$
- $c = [5, 4]$



SDD Example (3)

Step1(Initial state)

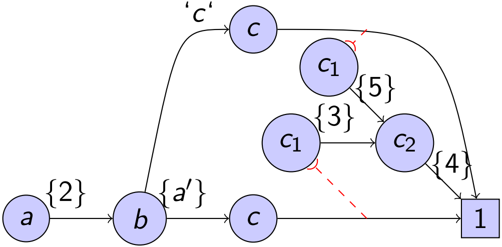
- $a = 2$
- $b = \text{`}c\text{`}$
- $c = [5, 4]$



SDD Example (3)

Step1(Initial state)

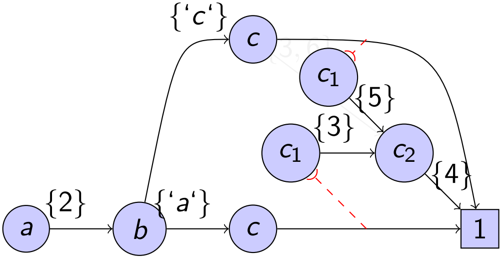
- $a = 2$
- $b = `c`$
- $c = [5, 4]$



SDD Example (4)

Step2(Step1(Initial state))

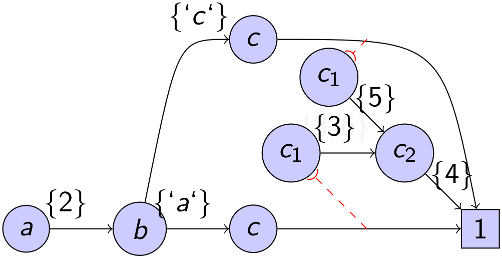
- $a = 2$
- $b = \text{‘}a\text{‘}$
- $c = [6, 4]$



SDD Example (4)

Step2(Step1(Initial state))

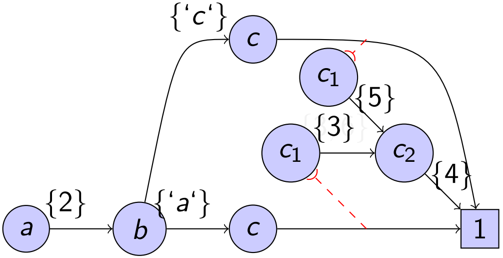
- $a = 2$
- $b = `a`$
- $c = [6, 4]$



SDD Example (4)

Step2(Step1(Initial state))

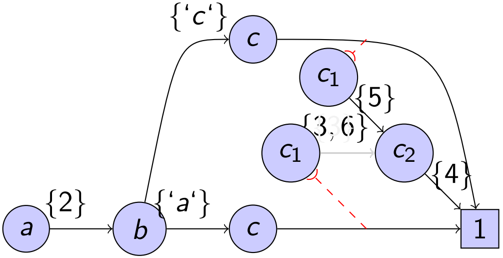
- $a = 2$
- $b =$ ‘a‘
- $c = [6, 4]$



SDD Example (4)

Step2(Step1(Initial state))

- $a = 2$
- $b = \text{`}a\text{`}$
- $c = [6, 4]$



SDD Manipulation

Problem

SDD is a complex data-structure. BDD based manipulation such as union, difference, intersection and apply is no more sufficient.

Solution

User can define functions that are homomorphisms w.r.t. the union and that are defined along to the SDD structure. These functions are called SDD-Homomorphisms

SDD Manipulation

Problem

SDD is a complex data-structure. BDD based manipulation such as union, difference, intersection and apply is no more sufficient.

Solution

User can define functions that are homomorphisms w.r.t. the union and that are defined along to the SDD structure. These functions are called SDD-Homomorphisms

SDD Homomorphisms

Some properties

- User defined operations.
- Support of union and composition.
- Locally defined $\langle Variable, Value\rangle$.
- Each atomic operation can be cached.
- Efficient homomorphism fix-point computation.
- $H_1 \circ H_2(S) = H_2(H_1(S))$ with H_1, H_2 : SDD homs
- $H(0_{sdd}) = 0_{sdd}$
- $H(S_1 \cup S_2) = H(S_1) \cup H(S_2)$

SDD Homomorphisms

Some properties

- User defined operations.
- Support of union and composition.
- Locally defined $\langle Variable, Value \rangle$.
- Each atomic operation can be cached.
- Efficient homomorphism fix-point computation.
- $H_1 \circ H_2(S) = H_2(H_1(S))$ with H_1, H_2 : SDD homs
- $H(0_{sdd}) = 0_{sdd}$
- $H(S_1 \cup S_2) = H(S_1) \cup H(S_2)$

SDD Homomorphisms

Some properties

- User defined operations.
- Support of union and composition.
- Locally defined $\langle Variable, Value \rangle$.
- Each atomic operation can be cached.
- Efficient homomorphism fix-point computation.
- $H_1 \circ H_2(S) = H_2(H_1(S))$ with H_1, H_2 : SDD homs
- $H(0_{sdd}) = 0_{sdd}$
- $H(S_1 \cup S_2) = H(S_1) \cup H(S_2)$

SDD Homomorphisms

Some properties

- User defined operations.
- Support of union and composition.
- Locally defined $\langle Variable, Value \rangle$.
- Each atomic operation can be cached.
- Efficient homomorphism fix-point computation.
- $H_1 \circ H_2(S) = H_2(H_1(S))$ with H_1, H_2 : SDD homs
- $H(0_{sdd}) = 0_{sdd}$
- $H(S_1 \cup S_2) = H(S_1) \cup H(S_2)$

SDD Homomorphisms

Some properties

- User defined operations.
- Support of union and composition.
- Locally defined $\langle Variable, Value \rangle$.
- Each atomic operation can be cached.
- Efficient homomorphism fix-point computation.
- $H_1 \circ H_2(S) = H_2(H_1(S))$ with H_1, H_2 : SDD homs
- $H(0_{sdd}) = 0_{sdd}$
- $H(S_1 \cup S_2) = H(S_1) \cup H(S_2)$

SDD Homomorphisms

Some properties

- User defined operations.
- Support of union and composition.
- Locally defined $\langle Variable, Value \rangle$.
- Each atomic operation can be cached.
- Efficient homomorphism fix-point computation.
- $H_1 \circ H_2(S) = H_2(H_1(S))$ with H_1, H_2 : SDD homs
- $H(0_{sdd}) = 0_{sdd}$
- $H(S_1 \cup S_2) = H(S_1) \cup H(S_2)$

SDD Homomorphisms

Some properties

- User defined operations.
- Support of union and composition.
- Locally defined $\langle Variable, Value \rangle$.
- Each atomic operation can be cached.
- Efficient homomorphism fix-point computation.
- $H_1 \circ H_2(S) = H_2(H_1(S))$ with H_1, H_2 : SDD homs
- $H(0_{sdd}) = 0_{sdd}$
- $H(S_1 \cup S_2) = H(S_1) \cup H(S_2)$

SDD Homomorphisms

Some properties

- User defined operations.
- Support of union and composition.
- Locally defined $\langle Variable, Value \rangle$.
- Each atomic operation can be cached.
- Efficient homomorphism fix-point computation.
- $H_1 \circ H_2(S) = H_2(H_1(S))$ with H_1, H_2 : SDD homs
- $H(0_{sdd}) = 0_{sdd}$
- $H(S_1 \cup S_2) = H(S_1) \cup H(S_2)$

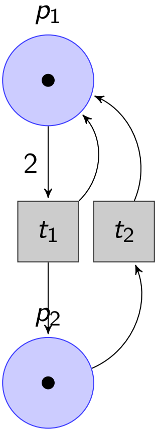
SDD Homomorphisms : Example (1)

- The next state function (τ) is encoded as a composition of SDD homomorphisms.
- $H^{-}_{p,w}(S_1)$ consumes w tokens from place p or returns the empty SDD (0_{sdd}) if it is not possible
- $H^{+}_{p,w}(S_1)$ produces w tokens in place p

Encoding τ

$t_1 = H^{+}_{p_1,1} \circ H^{+}_{p_2,1} \circ H^{-}_{p_1,2}$ and $t_2 = H^{+}_{p_1,1} \circ H^{-}_{p_2,1}$

$\tau = t_1 \cup t_2$



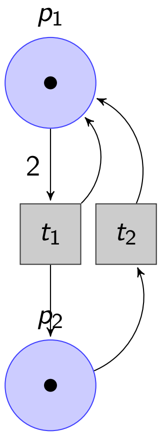
SDD Homomorphisms : Example (1)

- The next state function (τ) is encoded as a composition of SDD homomorphisms.
- $H^{-}_{p,w}(S_1)$ consumes w tokens from place p or returns the empty SDD (0_{sdd}) if it is not possible
- $H^{+}_{p,w}(S_1)$ produces w tokens in place p

Encoding τ

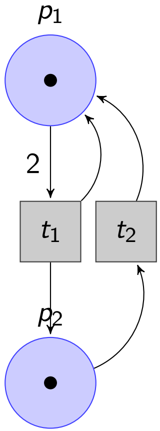
$t_1 = H^{+}_{p_1,1} \circ H^{+}_{p_2,1} \circ H^{-}_{p_1,2}$ and $t_2 = H^{+}_{p_1,1} \circ H^{-}_{p_2,1}$

$\tau = t_1 \cup t_2$



SDD Homomorphisms : Example (1)

- The next state function (τ) is encoded as a composition of SDD homomorphisms.
- $H^{-}_{p,w}(S_1)$ consumes w tokens from place p or returns the empty SDD (0_{sdd}) if it is not possible
- $H^{+}_{p,w}(S_1)$ produces w tokens in place p



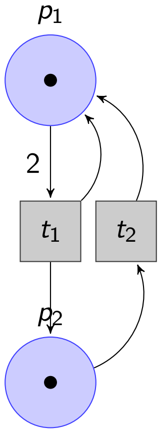
Encoding τ

$t_1 = H^{+}_{p_1,1} \circ H^{+}_{p_2,1} \circ H^{-}_{p_1,2}$ and $t_2 = H^{+}_{p_1,1} \circ H^{-}_{p_2,1}$

$\tau = t_1 \cup t_2$

SDD Homomorphisms : Example (1)

- The next state function (τ) is encoded as a composition of SDD homomorphisms.
- $H^{-}_{p,w}(S_1)$ consumes w tokens from place p or returns the empty SDD (0_{sdd}) if it is not possible
- $H^{+}_{p,w}(S_1)$ produces w tokens in place p



Encoding τ

$t_1 = H^{+}_{p_1,1} \circ H^{+}_{p_2,1} \circ H^{-}_{p_1,2}$ and $t_2 = H^{+}_{p_1,1} \circ H^{-}_{p_2,1}$
$\tau = t_1 \cup t_2$

SDD Homomorphisms : Example (2)

Algorithm 1: Compute the state space : Breadth-first exploration (BFS)

Input: s_0 : initial state.
Input: τ : set of transition homomorphisms.
Result: set of reachable states
begin
    $s, s_{old}, temp$: set of states ; ;
    $s \leftarrow \{s_0\}$; ;
    repeat
        $s_{old} \leftarrow s$; ;
        foreach $t \in \tau$ **do**
            $temp \leftarrow t(s)$;;
            $s \leftarrow s \cup temp$;;
    until $s = s_{old}$;
    return s ;

SDD Homomorphisms : Example (2)

Compute the state space

Algorithm 2: Breadth-first exploration (BFS)

Input: s_0 : initial state.
Input: τ : set of transition homomorphisms.
Result: set of reachable states
begin

```
    s, s_old, temp : set of states ; ;
    s ← {s_0} ; ;
    repeat
        s_old ← s ; ;
        foreach t ∈ τ do
            temp ← t(s) ;;
            s ← s ∪ temp ;;
    until s = s_old;
    return s;
```

Reachability

	s	s_{old}	t	$temp$	line
0	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$				3
1	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$			5
2	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$	t_1		6
3	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$	t_1	0_{sdd}	7
4	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$	t_1	0_{sdd}	8
5	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$	t_2	$p_1 \xrightarrow{2} p_2 \xrightarrow{0} \boxed{1}$	7
6	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$; $p_1 \xrightarrow{2} p_2 \xrightarrow{0} \boxed{1}$	$p_1 \xrightarrow{2} p_2 \xrightarrow{0} \boxed{1}$	t_2	$p_1 \xrightarrow{2} p_2 \xrightarrow{0} \boxed{1}$	8

$$t_1 = H^-_{p_1,2} \circ H^+_{p_1,1} \circ H^+_{p_2,1} \text{ and } t_2 = H^-_{p_2,1} \circ H^+_{p_1,1}$$

SDD Homomorphisms : Example (2)

Compute the state space

Algorithm 3: Breadth-first exploration (BFS)

Input: s_0 : initial state.
Input: τ : set of transition homomorphisms.
Result: set of reachable states
begin
  $s, s_{old}, temp$: set of states ; ;
  $s \leftarrow \{s_0\}$; ;
  repeat
    $s_{old} \leftarrow s$; ;
    foreach $t \in \tau$ **do**
      $temp \leftarrow t(s)$;;
      $s \leftarrow s \cup temp$;;
  until $s = s_{old}$;
  return s;

Reachability

	s	s_{old}	t	$temp$	line
0	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$				3
1	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$			5
2	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$	t_1		6
3	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$	t_1	0_{sdd}	7
4	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$	t_1	0_{sdd}	8
5	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$	t_2	$p_1 \xrightarrow{2} p_2 \xrightarrow{0} \boxed{1}$	7
6	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$ $p_1 \xrightarrow{2} p_2 \xrightarrow{0} \boxed{1}$	$p_1 \xrightarrow{2} p_2 \xrightarrow{0} \boxed{1}$	t_2	$p_1 \xrightarrow{2} p_2 \xrightarrow{0} \boxed{1}$	8

$$t_1 = H^-_{p_1,2} \circ H^+_{p_1,1} \circ H^+_{p_2,1} \text{ and } t_2 = H^-_{p_2,1} \circ H^+_{p_1,1}$$

SDD Homomorphisms : Example (2)

Compute the state space

Algorithm 4: Breadth-first exploration (BFS)

Input: s_0 : initial state.
Input: τ : set of transition homomorphisms.
Result: set of reachable states
begin
    $s, s_{old}, temp$: set of states ; ;
    $s \leftarrow \{s_0\}$; ;
    repeat
        $s_{old} \leftarrow s$; ;
        foreach $t \in \tau$ **do**
            $temp \leftarrow t(s)$;;
            $s \leftarrow s \cup temp$;;
    until $s = s_{old}$;
    return s ;

Reachability

	s	s_{old}	t	$temp$	line
0	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$				3
1	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$			5
2	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$	t_1		6
3	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$	t_1	0_{sdd}	7
4	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$	t_1	0_{sdd}	8
5	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$	t_2	$p_1 \xrightarrow{2} p_2 \xrightarrow{0} \boxed{1}$	7
6	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$, $p_1 \xrightarrow{2} p_2 \xrightarrow{0} \boxed{1}$	$p_1 \xrightarrow{2} p_2 \xrightarrow{0} \boxed{1}$	t_2	$p_1 \xrightarrow{2} p_2 \xrightarrow{0} \boxed{1}$	8

$$t_1 = H^{-}_{p_1,2} \circ H^{+}_{p_1,1} \circ H^{+}_{p_2,1} \text{ and } t_2 = H^{-}_{p_2,1} \circ H^{+}_{p_1,1}$$

SDD Homomorphisms : Example (2)

Compute the state space

Algorithm 5: Breadth-first exploration (BFS)

Input: s_0 : initial state.
Input: τ : set of transition homomorphisms.
Result: set of reachable states
begin
  $s, s_{old}, temp$: set of states ; ;
  $s \leftarrow \{s_0\}$; ;
  repeat
    $s_{old} \leftarrow s$; ;
    foreach $t \in \tau$ **do**
      $temp \leftarrow t(s)$;;
      $s \leftarrow s \cup temp$;;
  until $s = s_{old}$;
  return s ;

Reachability

	s	s_{old}	t	$temp$	line
0	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$				3
1	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$			5
2	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$	t_1		6
3	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$	t_1	0_{sdd}	7
4	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$	t_1	0_{sdd}	8
5	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$	t_2	$p_1 \xrightarrow{2} p_2 \xrightarrow{0} \boxed{1}$	7
6	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$, $p_1 \xrightarrow{2} p_2 \xrightarrow{0} \boxed{1}$	$p_1 \xrightarrow{2} p_2 \xrightarrow{0} \boxed{1}$	t_2	$p_1 \xrightarrow{2} p_2 \xrightarrow{0} \boxed{1}$	8

$$t_1 = H^{-}_{p_1,2} \circ H^{+}_{p_1,1} \circ H^{+}_{p_2,1} \text{ and } t_2 = H^{-}_{p_2,1} \circ H^{+}_{p_1,1}$$

SDD Homomorphisms : Example (2)

Compute the state space

Algorithm 6: Breadth-first exploration (BFS)

Input: s_0 : initial state.
Input: τ : set of transition homomorphisms.
Result: set of reachable states
begin
    $s, s_{old}, temp$: set of states ; ;
    $s \leftarrow \{s_0\}$; ;
    repeat
        $s_{old} \leftarrow s$; ;
        foreach $t \in \tau$ **do**
            $temp \leftarrow t(s)$;;
            $s \leftarrow s \cup temp$;;
    until $s = s_{old}$;
    return s ;

Reachability

	s	s_{old}	t	$temp$	line
0	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$				3
1	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$			5
2	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$	t_1		6
3	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$	t_1	0_{sdd}	7
4	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$	t_1	0_{sdd}	8
5	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$	t_2	$p_1 \xrightarrow{2} p_2 \xrightarrow{0} \boxed{1}$	7
6	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$; $p_1 \xrightarrow{2} p_2 \xrightarrow{0} \boxed{1}$	$p_1 \xrightarrow{2} p_2 \xrightarrow{0} \boxed{1}$	t_2	$p_1 \xrightarrow{2} p_2 \xrightarrow{0} \boxed{1}$	8

$$t_1 = H^-_{p_1,2} \circ H^+_{p_1,1} \circ H^+_{p_2,1} \text{ and } t_2 = H^-_{p_2,1} \circ H^+_{p_1,1}$$

SDD Homomorphisms : Example (2)

Compute the state space

Algorithm 7: Breadth-first exploration (BFS)

Input: s_0 : initial state.
Input: τ : set of transition homomorphisms.
Result: set of reachable states
begin
  $s, s_{old}, temp$: set of states ; ;
  $s \leftarrow \{s_0\}$; ;
  repeat
    $s_{old} \leftarrow s$; ;
    foreach $t \in \tau$ **do**
      $temp \leftarrow t(s)$;;
      $s \leftarrow s \cup temp$;;
  until $s = s_{old}$;
  return s ;

Reachability

	s	s_{old}	t	$temp$	line
0	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$				3
1	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$			5
2	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$	t_1		6
3	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$	t_1	0_{sdd}	7
4	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$	t_1	0_{sdd}	8
5	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$	t_2	$p_1 \xrightarrow{2} p_2 \xrightarrow{0} \boxed{1}$	7
6	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$; $p_1 \xrightarrow{2} p_2 \xrightarrow{0} \boxed{1}$	$p_1 \xrightarrow{2} p_2 \xrightarrow{0} \boxed{1}$	t_2	$p_1 \xrightarrow{2} p_2 \xrightarrow{0} \boxed{1}$	8

$$t_1 = H^-_{p_1,2} \circ H^+_{p_1,1} \circ H^+_{p_2,1} \text{ and } t_2 = H^-_{p_2,1} \circ H^+_{p_1,1}$$

SDD Homomorphisms : Example (2)

Compute the state space

Algorithm 8: Breadth-first exploration (BFS)

Input: s_0 : initial state.
Input: τ : set of transition homomorphisms.
Result: set of reachable states
begin
    $s, s_{old}, temp$: set of states ; ;
    $s \leftarrow \{s_0\}$; ;
    repeat
        $s_{old} \leftarrow s$; ;
        foreach $t \in \tau$ **do**
            $temp \leftarrow t(s)$;;
            $s \leftarrow s \cup temp$;;
    until $s = s_{old}$;
    return s ;

Reachability

	s	s_{old}	t	$temp$	line
0	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$				3
1	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$			5
2	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$	t_1		6
3	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$	t_1	0_{sdd}	7
4	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$	t_1	0_{sdd}	8
5	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$	t_2	$p_1 \xrightarrow{2} p_2 \xrightarrow{0} \boxed{1}$	7
6	$p_1 \xrightarrow{1} p_2 \xrightarrow{1} \boxed{1}$ $p_1 \xrightarrow{2} p_2 \xrightarrow{0} \boxed{1}$	$p_1 \xrightarrow{2} p_2 \xrightarrow{0} \boxed{1}$	t_2	$p_1 \xrightarrow{2} p_2 \xrightarrow{0} \boxed{1}$	8

$$t_1 = H^-_{p_1,2} \circ H^+_{p_1,1} \circ H^+_{p_2,1} \text{ and } t_2 = H^-_{p_2,1} \circ H^+_{p_1,1}$$

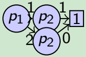
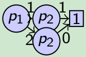
SDD Homomorphisms : Example (3)

Compute the state space

Algorithm 9: Breadth-first exploration (BFS)

Input: s_0 : initial state.
Input: τ : set of transition homomorphisms.
Result: set of reachable states
begin
  $s, s_{old}, temp$: set of states ; ;
  $s \leftarrow \{s_0\}$; ;
  repeat
    $s_{old} \leftarrow s$; ;
    foreach $t \in \tau$ **do**
      $temp \leftarrow t(s)$;;
      $s \leftarrow s \cup temp$;;
  until $s = s_{old}$;
  return s ;

Reachability

	s	s_{old}	t	$temp$	line
7					5
8			t_1		6
9			t_1	part. cache	7
10			t_1		8
11			t_2		6

$$t_1 = H^-_{p_1,2} \circ H^+_{p_1,1} \circ H^+_{p_2,1} \text{ and } t_2 = H^-_{p_2,1} \circ H^+_{p_1,1}$$

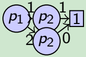
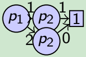
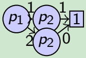
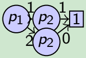
SDD Homomorphisms : Example (3)

Compute the state space

Algorithm 10: Breadth-first exploration (BFS)

Input: s_0 : initial state.
Input: τ : set of transition homomorphisms.
Result: set of reachable states
begin
    $s, s_{old}, temp$: set of states ; ;
    $s \leftarrow \{s_0\}$; ;
    repeat
        $s_{old} \leftarrow s$; ;
        foreach $t \in \tau$ **do**
            $temp \leftarrow t(s)$;;
            $s \leftarrow s \cup temp$;;
    until $s = s_{old}$;
    return s ;

Reachability

	s	s_{old}	t	$temp$	line
7					5
8			t_1		6
9			t_1	part. cache	7
10			t_1		8
11			t_2		6

$$t_1 = H^-_{p_1,2} \circ H^+_{p_1,1} \circ H^+_{p_2,1} \text{ and } t_2 = H^-_{p_2,1} \circ H^+_{p_1,1}$$

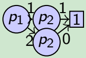
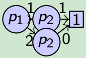
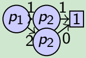
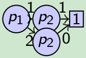
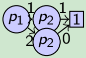
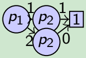
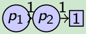
SDD Homomorphisms : Example (3)

Compute the state space

Algorithm 11: Breadth-first exploration (BFS)

Input: s_0 : initial state.
Input: τ : set of transition homomorphisms.
Result: set of reachable states
begin
  $s, s_{old}, temp$: set of states; ;
  $s \leftarrow \{s_0\}$; ;
  repeat
    $s_{old} \leftarrow s$; ;
    foreach $t \in \tau$ **do**
      $temp \leftarrow t(s)$;;
      $s \leftarrow s \cup temp$;;
  until $s = s_{old}$;
  return s ;

Reachability

	s	s_{old}	t	$temp$	line
7					5
8			t_1		6
9			t_1	 part. cache	7
10			t_1		8
11			t_2		6

$$t_1 = H^-_{p_1,2} \circ H^+_{p_1,1} \circ H^+_{p_2,1} \text{ and } t_2 = H^-_{p_2,1} \circ H^+_{p_1,1}$$

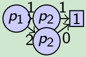
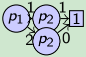
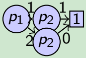
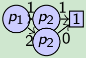
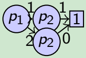
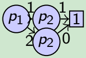
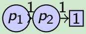
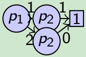
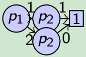
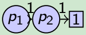
SDD Homomorphisms : Example (3)

Compute the state space

Algorithm 12: Breadth-first exploration (BFS)

Input: s_0 : initial state.
Input: τ : set of transition homomorphisms.
Result: set of reachable states
begin
  $s, s_{old}, temp$: set of states ; ;
  $s \leftarrow \{s_0\}$; ;
  repeat
    $s_{old} \leftarrow s$; ;
    foreach $t \in \tau$ **do**
      $temp \leftarrow t(s)$;;
      $s \leftarrow s \cup temp$;;
  until $s = s_{old}$;
  return s ;

Reachability

	s	s_{old}	t	$temp$	line
7					5
8			t_1		6
9			t_1	 part. cache	7
10			t_1		8
11			t_2		6

$$t_1 = H^{-}_{p_1,2} \circ H^{+}_{p_1,1} \circ H^{+}_{p_2,1} \text{ and } t_2 = H^{-}_{p_2,1} \circ H^{+}_{p_1,1}$$

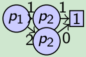
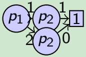
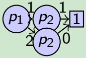
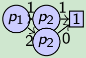
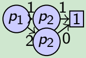
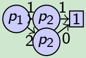
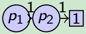
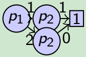
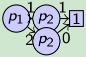
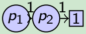
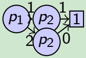
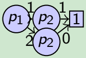
SDD Homomorphisms : Example (3)

Compute the state space

Algorithm 13: Breadth-first exploration (BFS)

Input: s_0 : initial state.
Input: τ : set of transition homomorphisms.
Result: set of reachable states
begin
  $s, s_{old}, temp$: set of states ; ;
  $s \leftarrow \{s_0\}$; ;
  repeat
    $s_{old} \leftarrow s$; ;
    foreach $t \in \tau$ **do**
      $temp \leftarrow t(s)$;;
      $s \leftarrow s \cup temp$;;
  until $s = s_{old}$;
  return s ;

Reachability

	s	s_{old}	t	$temp$	line
7					5
8			t_1		6
9			t_1	 part. cache	7
10			t_1		8
11			t_2		6

$$t_1 = H^-_{p_1,2} \circ H^+_{p_1,1} \circ H^+_{p_2,1} \text{ and } t_2 = H^-_{p_2,1} \circ H^+_{p_1,1}$$

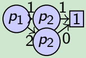
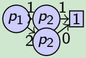
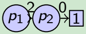
SDD Homomorphisms : Example (4)

Compute the state space

Algorithm 14: Breadth-first exploration (BFS)

Input: s_0 : initial state.
Input: τ : set of transition homomorphisms.
Result: set of reachable states
begin
  $s, s_{old}, temp$: set of states ; ;
  $s \leftarrow \{s_0\}$; ;
  repeat
    $s_{old} \leftarrow s$; ;
    foreach $t \in \tau$ **do**
      $temp \leftarrow t(s)$;;
      $s \leftarrow s \cup temp$;;
  until $s = s_{old}$;
  return s ;

Reachability

	s	s_{old}	t	$temp$	line
12			t_2	 part. cache	7
13			t_2		8
14					10

$$t_1 = H^+_{p_1,1} \circ H^+_{p_2,1} \circ H^-_{p_1,2} \text{ and } t_2 = H^+_{p_1,1} \circ H^-_{p_2,1}$$

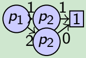
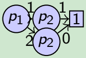
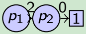
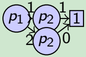
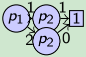
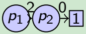
SDD Homomorphisms : Example (4)

Compute the state space

Algorithm 15: Breadth-first exploration (BFS)

Input: s_0 : initial state.
Input: τ : set of transition homomorphisms.
Result: set of reachable states
begin
    $s, s_{old}, temp$: set of states ; ;
    $s \leftarrow \{s_0\}$; ;
    repeat
        $s_{old} \leftarrow s$; ;
        foreach $t \in \tau$ **do**
            $temp \leftarrow t(s)$;;
            $s \leftarrow s \cup temp$;;
    until $s = s_{old}$;
    return s ;

Reachability

	s	s_{old}	t	$temp$	line
12			t_2	 part. cache	7
13			t_2		8
14					10

$$t_1 = H^+_{p_1,1} \circ H^+_{p_2,1} \circ H^-_{p_1,2} \text{ and } t_2 = H^+_{p_1,1} \circ H^-_{p_2,1}$$

SDD Homomorphisms : Example (4)

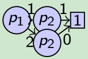
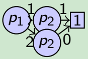
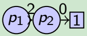
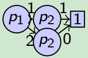
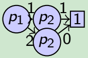
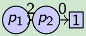
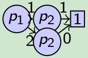
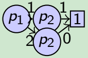
Compute the state space

Algorithm 16: Breadth-first exploration (BFS)

Input: s_0 : initial state.
Input: τ : set of transition homomorphisms.
Result: set of reachable states
begin

```
    s, s_old, temp : set of states ; ;
    s ← {s_0} ; ;
    repeat
        s_old ← s ; ;
        foreach t ∈ τ do
            temp ← t(s) ;;
            s ← s ∪ temp ;;
    until s = s_old;
    return s ;
```

Reachability

	s	s_{old}	t	$temp$	line
12			t_2	 part. cache	7
13			t_2		8
14					10

$$t_1 = H^{+}_{p_1,1} \circ H^{+}_{p_2,1} \circ H^{-}_{p_1,2} \text{ and } t_2 = H^{+}_{p_1,1} \circ H^{-}_{p_2,1}$$

Critic of the approach

Problem

Because of the SDD canonization process, the cost of doing nothing is almost as high as changing everything ! Thus even if a transition does not touch a variable the whole diagram must be re-canonized.

Solution

Cluster the transitions and the state space representation in order to minimize unnecessary SDD manipulation. This is done by reducing the size of :

Critic of the approach

Problem

Because of the SDD canonization process, the cost of doing nothing is almost as high as changing everything ! Thus even if a transition does not touch a variable the whole diagram must be re-canonized.

Solution

Cluster the transitions and the state space representation in order to minimize unnecessary SDD manipulation. This is done by reducing the size of :

- the homomorphims that are applied.
- the DD the homomorphisms are applied to.

Critic of the approach

Problem

Because of the SDD canonization process, the cost of doing nothing is almost as high as changing everything ! Thus even if a transition does not touch a variable the whole diagram must be re-canonized.

Solution

Cluster the transitions and the state space representation in order to minimize unnecessary SDD manipulation. This is done by reducing the size of :

- the homomorphims that are applied.
- the DD the homomorphisms are applied to.

Critic of the approach

Problem

Because of the SDD canonization process, the cost of doing nothing is almost as high as changing everything ! Thus even if a transition does not touch a variable the whole diagram must be re-canonized.

Solution

Cluster the transitions and the state space representation in order to minimize unnecessary SDD manipulation. This is done by reducing the size of :

- the homomorphims that are applied.
- the DD the homomorphisms are applied to.

Chaining Loop Exploration [28]

Algorithm 17: Chaining Loop : A more efficient way to compute the state space

Input: s_0 : initial state.
Input: C : set of clusters. Variable that are somehow linked together (require structural information)
Input: $\tau = \bigcup_{c \in C} \tau_c$: set of transition homomorphisms.
Result: set of reachable states
begin
  s, s_{old} : set of states ;
  $s \leftarrow \{s_0\}$;;
  repeat
    $s_{old} \leftarrow s$;
    foreach $c \in C$ **do**
      $s \leftarrow s \cup \tau_C(s)$;;
  until $s = s_{old}$;
  return s ;

Critic of the approach

Problem

Too many intermediate nodes are created. Those states are not part of the state space. They are temporarily created for the computation.

Solution

Any time a variable is modified by a transition it is (re)saturated, i.e. the set of transition that corresponds to this variable is applied to the node until a fixpoint is reached. When saturating a node, if lower nodes in the data structure are modified they will themselves be (re)saturated. Empirically an order of magnitude better than the chaining loop exploration. The algorithm can be found in [12, 10].

Critic of the approach

Problem

Too many intermediate nodes are created. Those states are not part of the state space. They are temporarily created for the computation.

Solution

Any time a variable is modified by a transition it is (re)saturated, i.e. the set of transition that corresponds to this variable is applied to the node until a fixpoint is reached. When saturating a node, if lower nodes in the data structure are modified they will themselves be (re)saturated. Empirically an order of magnitude better than the chaining loop exploration. The algorithm can be found in [12, 10].

Topological clustering

- Developed for Petri nets [12]
- Originally based on the Kronecker matrix
- Group places that are somehow related to avoid walking through the complete graph each time.
- If all places involved in a given transition t are in the same cluster c, t is said to be local to c.

Topological clustering

- Developed for Petri nets [12]
- Originally based on the Kronecker matrix
- Group places that are somehow related to avoid walking through the complete graph each time.
- If all places involved in a given transition t are in the same cluster c, t is said to be local to c.

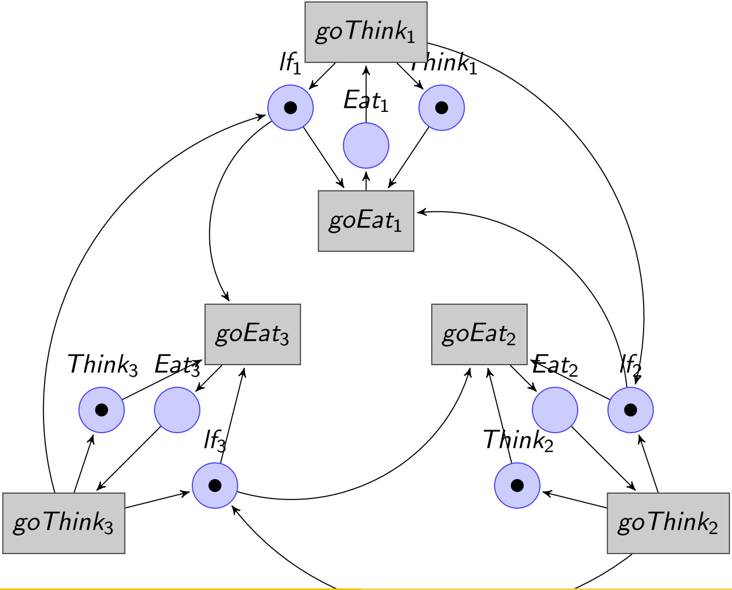
Topological clustering

- Developed for Petri nets [12]
- Originally based on the Kronecker matrix
- Group places that are somehow related to avoid walking through the complete graph each time.
- If all places involved in a given transition t are in the same cluster c, t is said to be local to c.

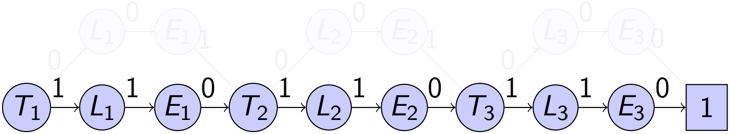
Topological clustering

- Developed for Petri nets [12]
- Originally based on the Kronecker matrix
- Group places that are somehow related to avoid walking through the complete graph each time.
- If all places involved in a given transition t are in the same cluster c, t is said to be local to c.

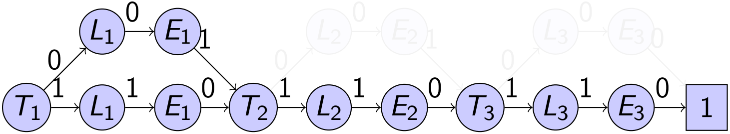
Topological clustering : Example (1)



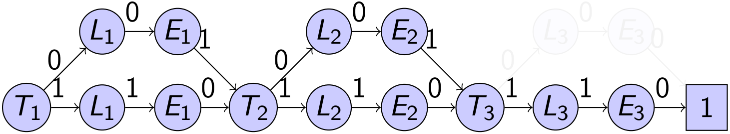
Topological clustering : Example (2)



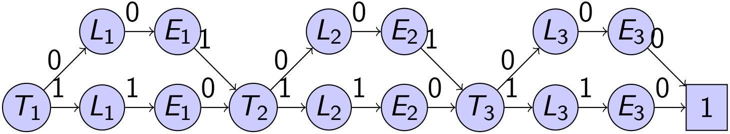
Topological clustering : Example (2)



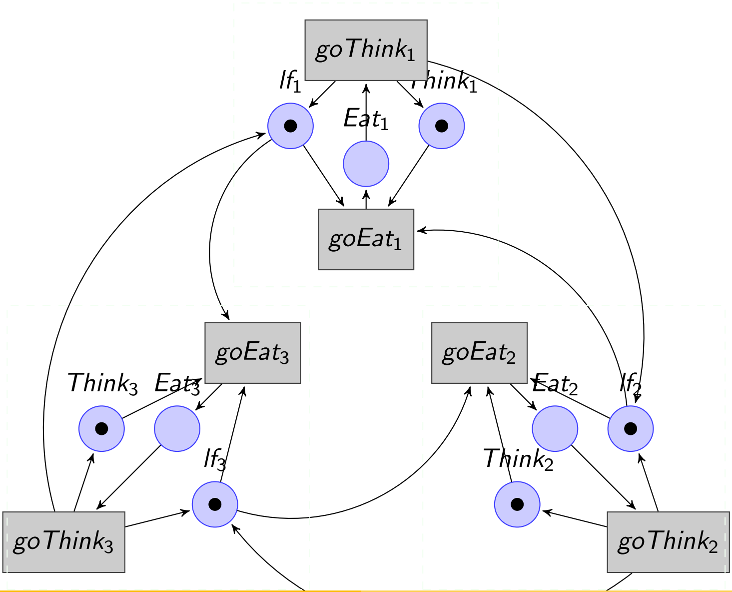
Topological clustering : Example (2)



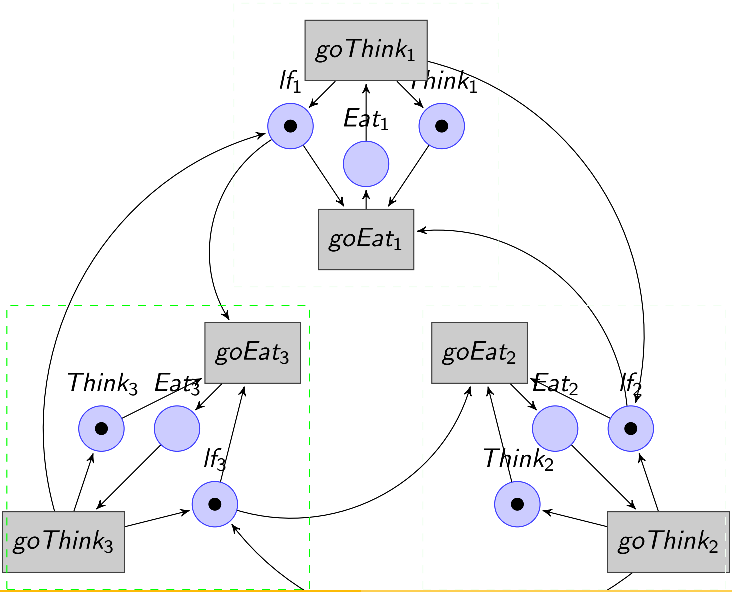
Topological clustering : Example (2)



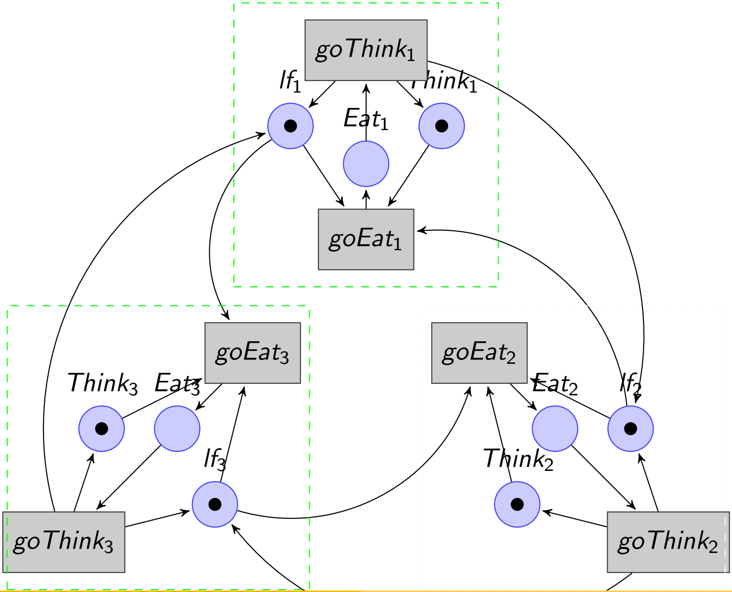
Topological clustering : Example (3)



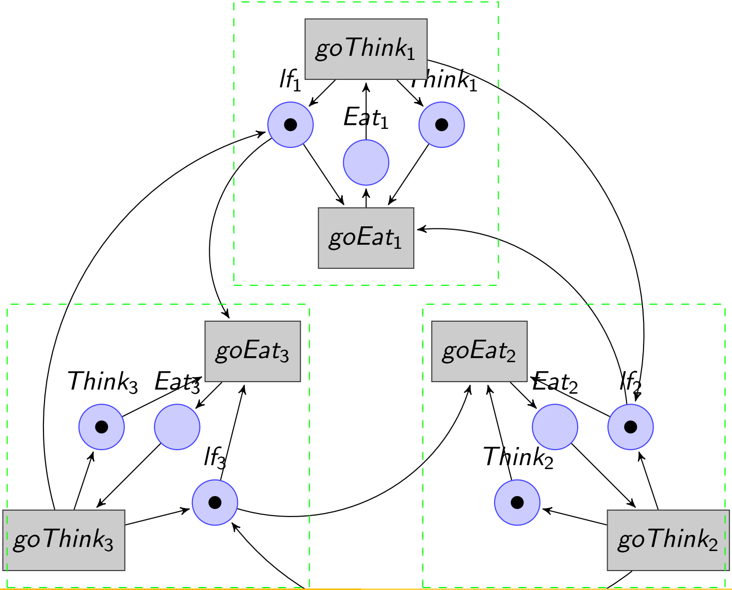
Topological clustering : Example (3)



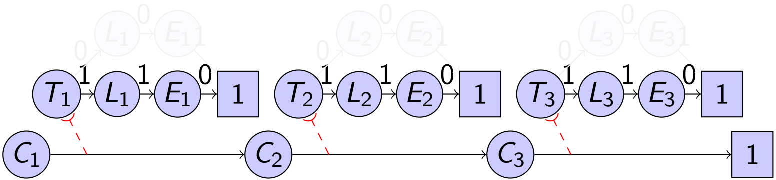
Topological clustering : Example (3)



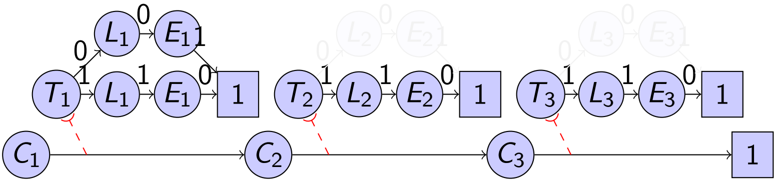
Topological clustering : Example (3)



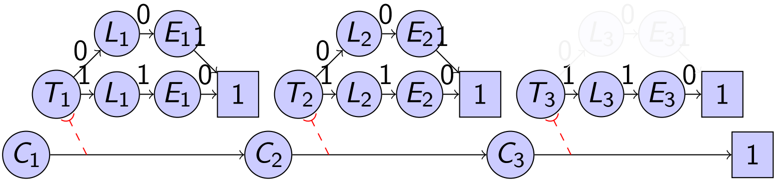
Topological clustering : Example (4)



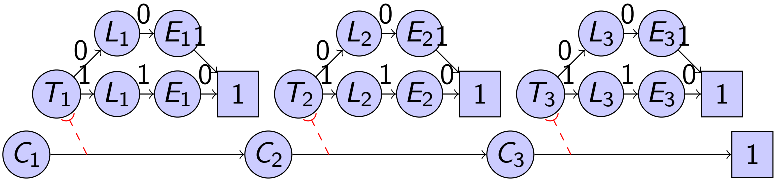
Topological clustering : Example (4)



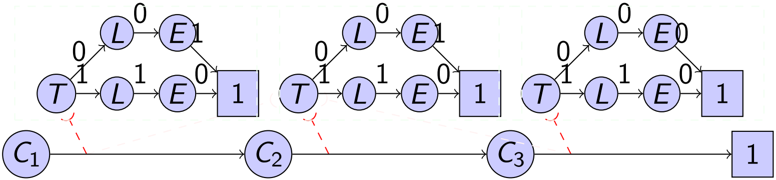
Topological clustering : Example (4)



Topological clustering : Example (4)

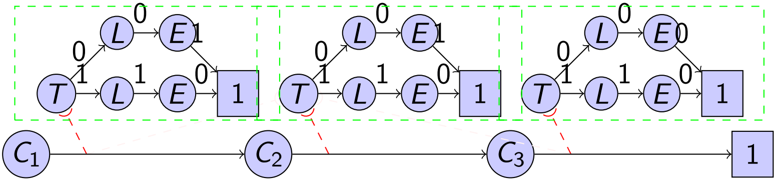


Topological clustering (Anonymisation) : Example (5)



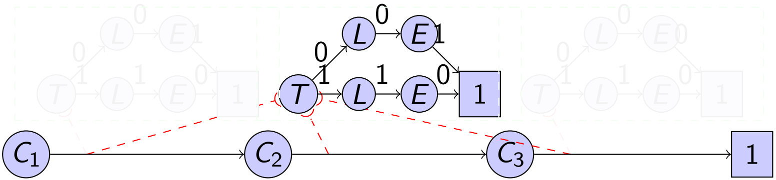
Similar to quotient graphs, the anonymisation enables more sharing as it abstracts for instance the process id. It enables more sharing and a better use of memoization.

Topological clustering (Anonymisation) : Example (5)



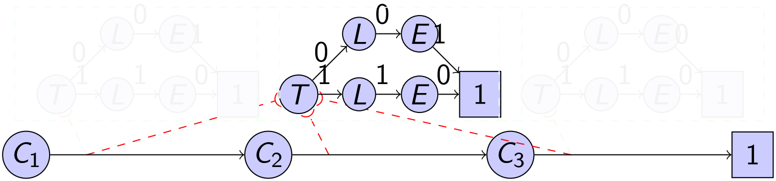
Similar to quotient graphs, the anonymisation enables more sharing as it abstracts for instance the process id. It enables more sharing and a better use of memoization.

Topological clustering (Anonymisation) : Example (5)



Similar to quotient graphs, the anonymisation enables more sharing as it abstracts for instance the process id. It enables more sharing and a better use of memoization.

Topological clustering (Anonymisation) : Example (5)



Similar to quotient graphs, the anonymisation enables more sharing as it abstracts for instance the process id. It enables more sharing and a better use of memoization.

Q & A

- Compare (RO)BDDs and SDDs ?
- What is the difference between the chaining loop and saturation ?

Q & A

- Compare (RO)BDDs and SDDs ?
- What is the difference between the chaining loop and saturation ?

PART III : AlPiNA : Basics

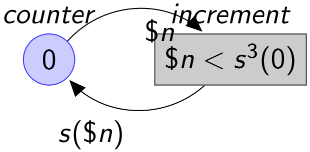
Algebraic

Petr**i** **N**ets

Analyzer

HLPN (High level Petri nets)

```
import "boolean.adt"
Adt naturals
    Sorts nat;
    Generators
        0 : nat;
        s : nat -> nat;
    Operations
        < : nat, nat -> bool;
    Axioms
        $x < 0 = false;
        0 < s($x) = true;
        s($y) < s($x) = $y < $x;
    Variables
        x : nat;   y : nat;
```



Model checking of High Level Petri Nets

Requirements

- Manipulate complex data ⇒ (RO)BDD are not adapted
- Next state depends on complex operations
- Markings are (hierarchical) vectors of terms (and not just booleans or even integers)

Solution

- Use high level and hierarchical extension of BDDs, i.e. **SDD**s (Set Decision Diagrams [17])

Model checking of High Level Petri Nets

Requirements

- Manipulate complex data ⇒ (RO)BDD are not adapted
- Next state depends on complex operations
- Markings are (hierarchical) vectors of terms (and not just booleans or even integers)

Solution

- Use high level and hierarchical extension of BDDs, i.e. **SDD**s (Set Decision Diagrams [17])

Model checking of High Level Petri Nets

Requirements

- Manipulate complex data $\Rightarrow$ (RO)BDD are not adapted
- Next state depends on complex operations
- Markings are (hierarchical) vectors of terms (and not just booleans or even integers)

Solution

- Use high level and hierarchical extension of BDDs, i.e. **SDD**s (Set Decision Diagrams [17])

Model checking of High Level Petri Nets

Requirements

- Manipulate complex data $\Rightarrow$ (RO)BDD are not adapted
- Next state depends on complex operations
- Markings are (hierarchical) vectors of terms (and not just booleans or even integers)

Solution

- Use high level and hierarchical extension of BDDs, i.e. **SDD**s (Set Decision Diagrams [17])

Model checking of High Level Petri Nets

How to represent the terms of a signature ?

- SDD are hierarchical
- SDD lack typing and structural information
- Rewriting requires complex homomorphisms

Solution

Extended version of SDDs that supports typing and equipped with a rewriting procedure $\Rightarrow$ **ΣDD**s

Model checking of High Level Petri Nets

How to represent the terms of a signature ?

- SDD are hierarchical
- SDD lack typing and structural information
- Rewriting requires complex homomorphisms

Solution

Extended version of SDDs that supports typing and equipped with a rewriting procedure $\Rightarrow$ **ΣDD**s

Model checking of High Level Petri Nets

How to represent the terms of a signature ?

- SDD are hierarchical
- SDD lack typing and structural information
- Rewriting requires complex homomorphisms

Solution

Extended version of SDDs that supports typing and equipped with a rewriting procedure ⇒ **ΣDD**s

Model checking of High Level Petri Nets

How to represent the terms of a signature ?

- SDD are hierarchical
- SDD lack typing and structural information
- Rewriting requires complex homomorphisms

Solution

Extended version of SDDs that supports typing and equipped with a rewriting procedure $\Rightarrow$ **ΣDD***s*

Σ Decision Diagrams [6]

Some properties

- Modelling formalism : AADT.
- Based on SDD
- Encoding of set of terms.
- Support of order-sorted terms.
- Encoding and decoding are consistent w.r.t. union.

Σ Decision Diagrams [6]

Some properties

- Modelling formalism : AADT.
- Based on SDD
- Encoding of set of terms.
- Support of order-sorted terms.
- Encoding and decoding are consistent w.r.t. union.

Σ Decision Diagrams [6]

Some properties

- Modelling formalism : AADT.
- Based on SDD
- Encoding of set of terms.
- Support of order-sorted terms.
- Encoding and decoding are consistent w.r.t. union.

Σ Decision Diagrams [6]

Some properties

- Modelling formalism : AADT.
- Based on SDD
- Encoding of set of terms.
- Support of order-sorted terms.
- Encoding and decoding are consistent w.r.t. union.

Σ Decision Diagrams [6]

Some properties

- Modelling formalism : AADT.
- Based on SDD
- Encoding of set of terms.
- Support of order-sorted terms.
- Encoding and decoding are consistent w.r.t. union.

Σ Decision Diagrams Example (1)

$$\{s(0) + s(0),\ 0 + s(0)\ \}$$

$$\{\ +(s(0)\ ,\ s(0));+(0,\ s(0))\ \}$$

$$\{\ +(\ \{s(0)\ ;\ 0\ \}\ ,\ s(0)\ \ )\ \}$$

Σ Decision Diagrams Example (1)

$$\{s(0) + s(0),\ 0 + s(0)\ \}$$

$$\{\ +(s(0)\ ,\ s(0));+(0,\ s(0))\ \}$$

$$\{\ +(\ \{s(0)\ ;\ 0\ \}\ ,\ s(0)\ \ )\ \}$$

Σ Decision Diagrams Example (1)

$$\{\mathbf{s(0) + s(0), 0 + s(0)}\}$$

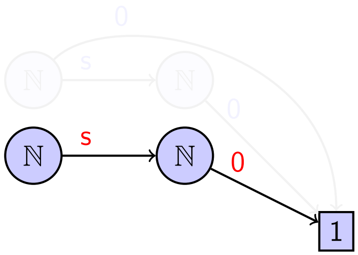
$$\{ +(\mathbf{s(0)}, \mathbf{s(0)}); +(\mathbf{0}, \mathbf{s(0)}) \}$$

$$\{ +(\{\mathbf{s(0)}; \mathbf{0}\}, \mathbf{s(0)}) \}$$

Σ Decision Diagrams Example (2)

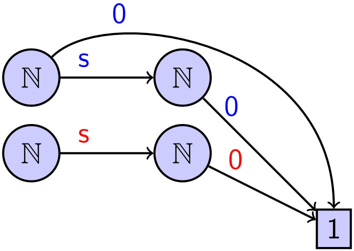
$$+(\ \{s(0)\ ;\ 0\ \}\ ,\ s(0)\ \ )$$

Σ Decision Diagrams Example (2)



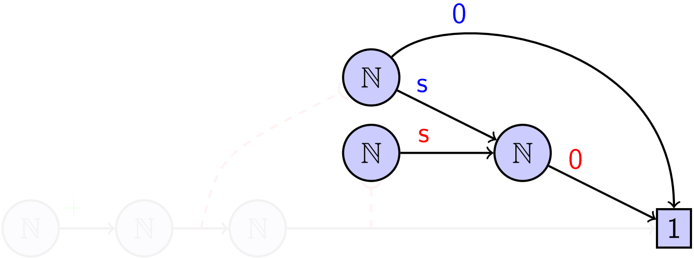
$$+(\ \{s(0)\ ;\ 0\ \}\ ,\ s(0)\ \ )$$

Σ Decision Diagrams Example (2)



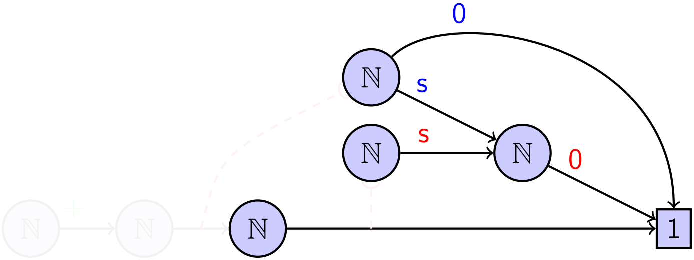
$$+(\ \{s(0)\ ;\ 0\ \}\ ,\ s(0)\ \ )$$

Σ Decision Diagrams Example (2)



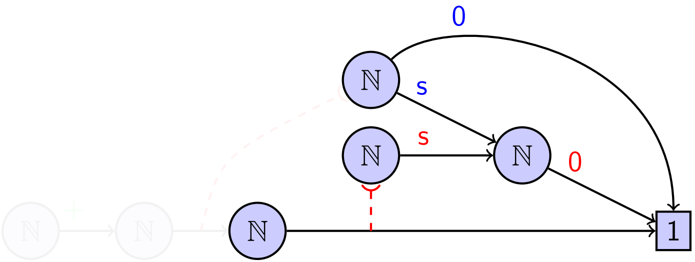
$$+(\ \{\mathbf{s}(\mathbf{0})\ ;\ \mathbf{0}\ \}\ ,\ \mathbf{s}(\mathbf{0})\ \ )$$

Σ Decision Diagrams Example (2)



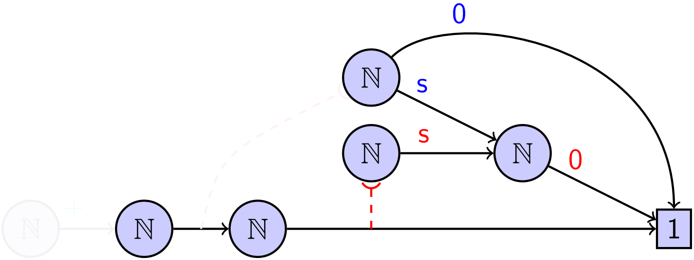
$$+(\ \{s(0)\ ;\ 0\ \}\ ,\ s(0)\ \ )$$

Σ Decision Diagrams Example (2)



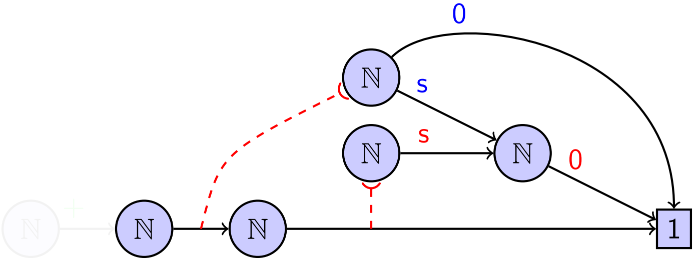
$+(\ \{s(0)\ ;\ 0\ \}\ ,\ s(0)\ \ )$

Σ Decision Diagrams Example (2)



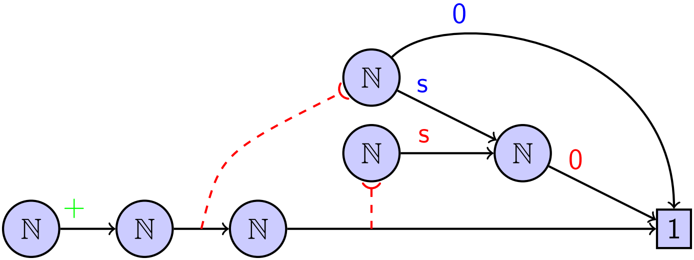
$+(\ \{s(0)\ ;\ 0\ \}\ ,\ s(0)\ \ )$

Σ Decision Diagrams Example (2)



$$+(\ \{s(0)\ ;\ 0\ \}\ ,\ s(0)\ \ )$$

Σ Decision Diagrams Example (2)



$$+(\ \{s(0)\ ;\ 0\ \}\ ,\ s(0)\ \ )$$

Σ Decision Diagrams Rewriting [6]

Some properties

- Rewrite sets of terms per rewriting step (see example).
- By exploiting sharing / caching
- Composition of simple homomorphisms.
  - Innermost/outermost rewriting, Rule application
  - Pattern matching, Variable substitution
- Termination and confluence are preserved.
- Works as Term Graph Rewriting [25] : $Host \setminus LHS \cup RHS$

Σ Decision Diagrams Rewriting [6]

Some properties

- Rewrite sets of terms per rewriting step (see example).
- By exploiting sharing / caching
- Composition of simple homomorphisms.
  - Innermost/outermost rewriting ; Rule application
  - Pattern matching ; Variable substitution
- Termination and confluence are preserved.
- Works as Term Graph Rewriting [25] : $Host \setminus LHS \cup RHS$

Σ Decision Diagrams Rewriting [6]

Some properties

- Rewrite sets of terms per rewriting step (see example).
- By exploiting sharing / caching
- Composition of simple homomorphisms.
  - Innermost/outermost rewriting ; Rule application.
  - Pattern matching ; Variable substitution.
- Termination and confluence are preserved.
- Works as Term Graph Rewriting [25] : $Host \setminus LHS \cup RHS$

Σ Decision Diagrams Rewriting [6]

Some properties

- Rewrite sets of terms per rewriting step (see example).
- By exploiting sharing / caching
- Composition of simple homomorphisms.
  - Innermost/outermost rewriting ; Rule application.
  - Pattern matching ; Variable substitution.
- Termination and confluence are preserved.
- Works as Term Graph Rewriting [25] : $Host \setminus LHS \cup RHS$

Σ Decision Diagrams Rewriting [6]

Some properties

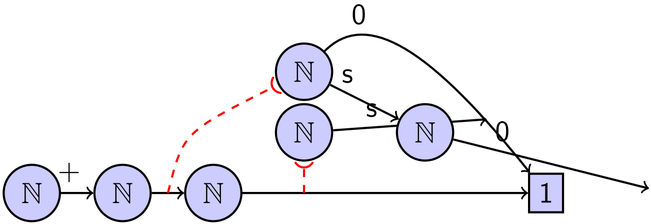
- Rewrite sets of terms per rewriting step (see example).
- By exploiting sharing / caching
- Composition of simple homomorphisms.
  - Innermost/outermost rewriting ; Rule application.
  - Pattern matching ; Variable substitution.
- Termination and confluence are preserved.
- Works as Term Graph Rewriting [25] : $Host \setminus LHS \cup RHS$

Σ Decision Diagrams Rewriting [6]

Some properties

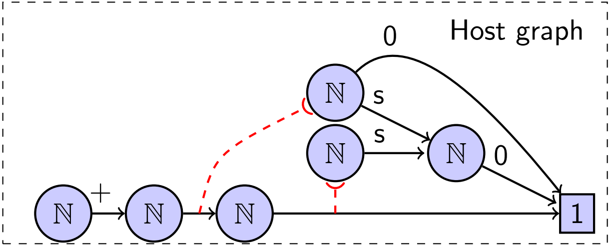
- Rewrite sets of terms per rewriting step (see example).
- By exploiting sharing / caching
- Composition of simple homomorphisms.
  - Innermost/outermost rewriting ; Rule application.
  - Pattern matching ; Variable substitution.
- Termination and confluence are preserved.
- Works as Term Graph Rewriting [25] : $Host \setminus LHS \cup RHS$

Σ Decision Diagrams Rewriting Example (1)



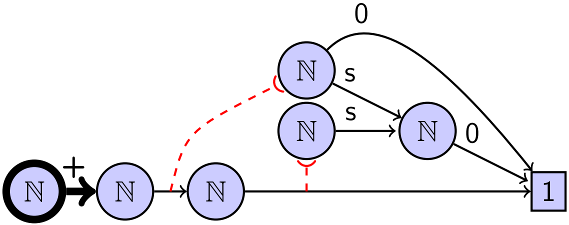
$$+(x, s(y)) \rightsquigarrow s(+(x, y))$$

Σ Decision Diagrams Rewriting Example (1)



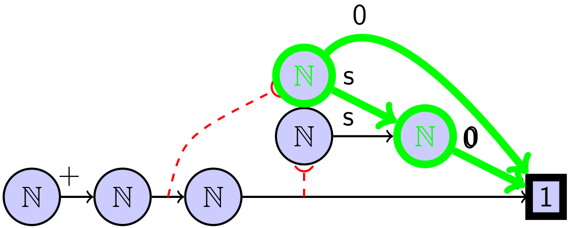
$$+(x, s(y)) \rightsquigarrow s(+(x, y))$$

Σ Decision Diagrams Rewriting Example (1)



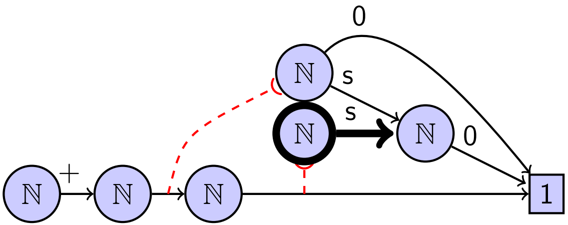
$$+(x, s(y)) \rightsquigarrow s(+(x, y))$$

$\boldsymbol{\Sigma}$ Decision Diagrams Rewriting Example (1)



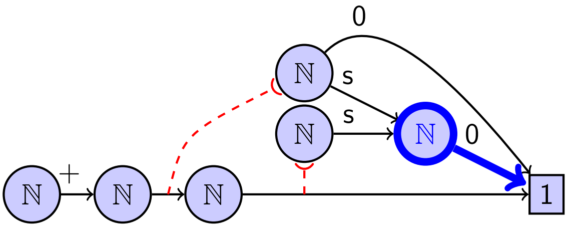
$$+(x, s(y)) \rightsquigarrow s(+(x, y))$$

Σ Decision Diagrams Rewriting Example (1)



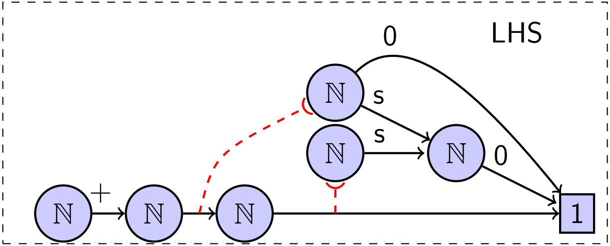
$$+(x, s(y)) \leadsto s(+(x, y))$$

Σ Decision Diagrams Rewriting Example (1)



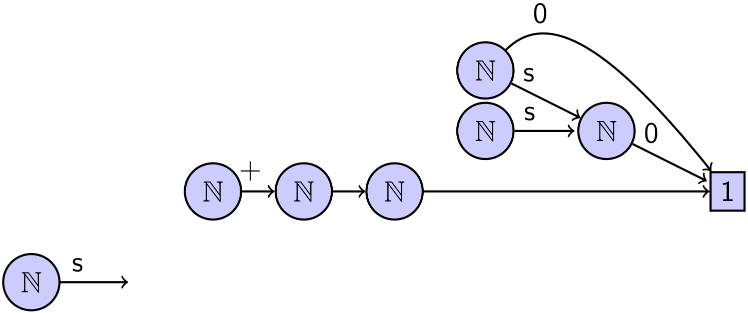
$$+(x, s(y)) \rightsquigarrow s(+(x, y))$$

Σ Decision Diagrams Rewriting Example (1)



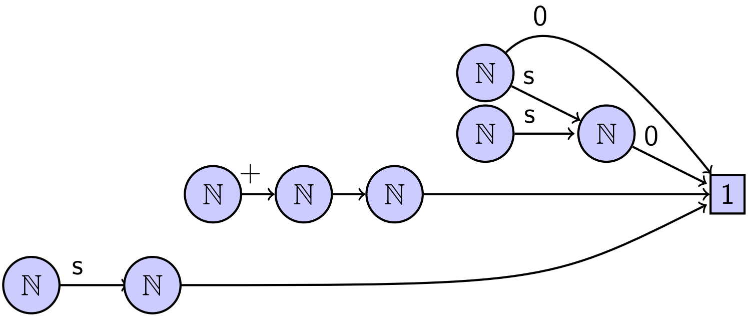
$$+(x, s(y)) \rightsquigarrow s(+(x, y))$$

Σ Decision Diagrams Rewriting Example (1)



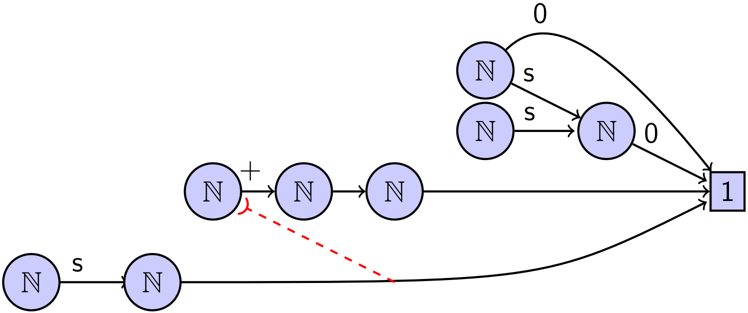
$$+(x, s(y)) \rightsquigarrow s(+(x, y))$$

Σ Decision Diagrams Rewriting Example (1)



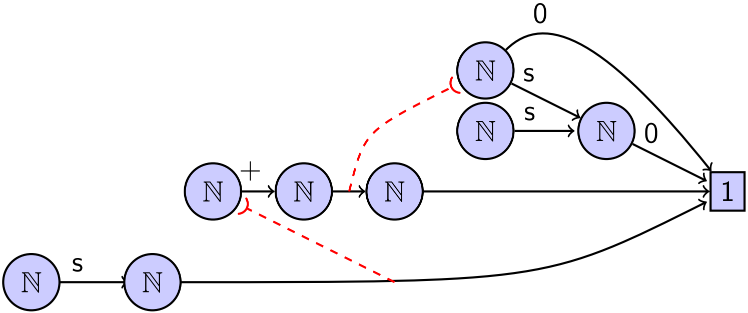
$$+(x, s(y)) \rightsquigarrow s(+(x, y))$$

Σ Decision Diagrams Rewriting Example (1)



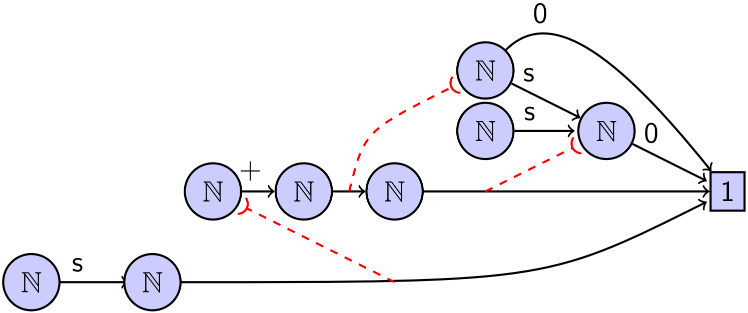
$$+(x, s(y)) \rightsquigarrow s(+(x, y))$$

Σ Decision Diagrams Rewriting Example (1)



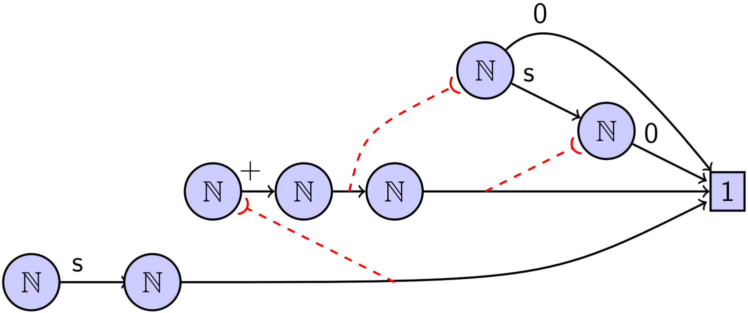
$$+(x, s(y)) \rightsquigarrow s(+(x, y))$$

Σ Decision Diagrams Rewriting Example (1)



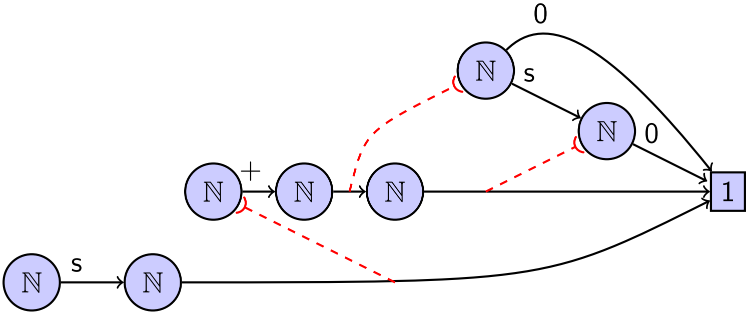
$$+(x, s(y)) \rightsquigarrow s(+(x, y))$$

Σ Decision Diagrams Rewriting Example (1)



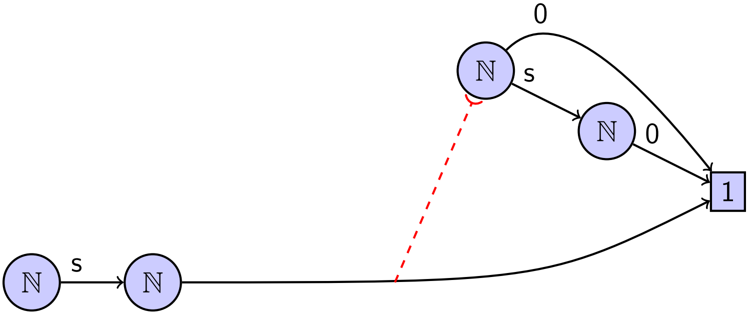
$$+(x, s(y)) \rightsquigarrow s(+(x, y))$$

Σ Decision Diagrams Rewriting Example (2)



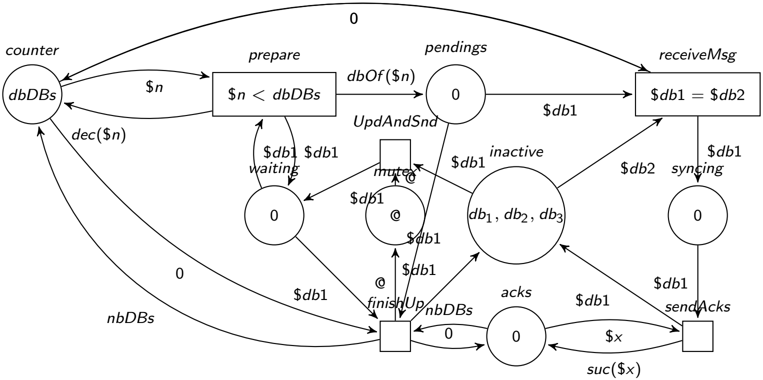
$$+(x,0) \rightsquigarrow x$$

Σ Decision Diagrams Rewriting Example (2)

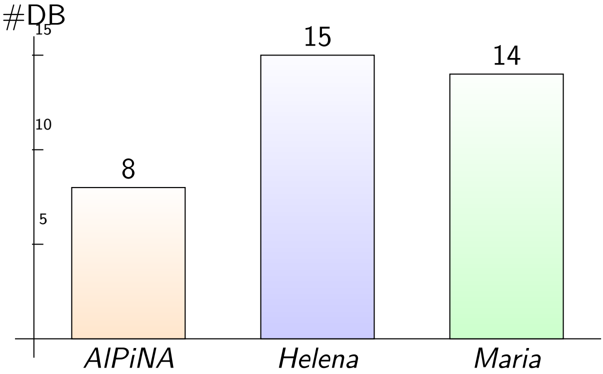


$$+(x,0) \rightsquigarrow x$$

Distributed Databases : Petri net Example



Benchmarks (using only the **ΣDD**)



Distributed Database model : number of databases the tool was able to calculate the state space for.

PART IV : AlPiNA : Advanced

Critic of the approach

Problem

Usually, high level Petri nets have less structural information than P/T nets simply because the formalism is more expressive. Because of that, less transitions and places can be grouped based on their topological information.

Solution

Use the algebraic information to group variables together and therefore extract clustering information from the algebraic part. This is called algebraic clustering [5] (vs topological clustering).

Critic of the approach

Problem

Usually, high level Petri nets have less structural information than P/T nets simply because the formalism is more expressive. Because of that, less transitions and places can be grouped based on their topological information.

Solution

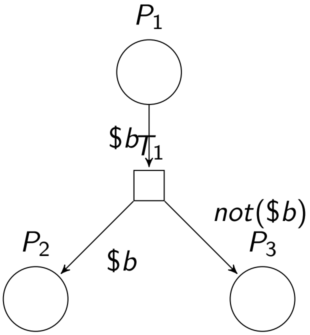
Use the algebraic information to group variables together and therefore extract clustering information from the algebraic part. This is called algebraic clustering [5] (vs topological clustering).

Algebraic Clustering [5]

Idea

The main idea is to group places (resp. transitions) based on the algebraic values they contain (resp. handle). The partition is made based on a user-defined function $f_{cluster} : T \times P \rightarrow C$.
f associates a cluster to a pair $\langle token, place \rangle$. Where C is the set of clusters, T is the set of values of the algebras and P is the set of places.
This can be seen as a kind of user-controlled unfolding.

Algebraic Clustering : Example



Cluster function

$$Cluster \ : \ p, \ token \mapsto \begin{cases} C_1 \text{ if } t = true \\ C_2 \text{ if } t = false \end{cases}$$

Critic of the approach

Problem

The clustering function can be hard to define.

Solution

Although theoretically difficult, user can rely on heuristics to determine the clustering function. An effective heuristic is to cluster together processes and their resources, while shared resources are kept together in other clusters [5]. Optimally, a DSL would provide the necessary syntactic sugar to the end-user.

Critic of the approach

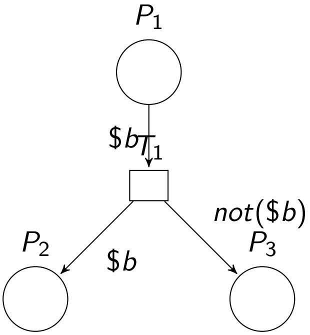
Problem

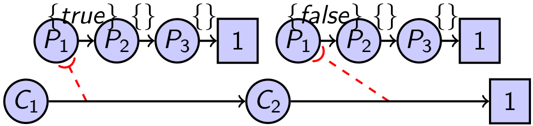
The clustering function can be hard to define.

Solution

Although theoretically difficult, user can rely on heuristics to determine the clustering function. An effective heuristic is to cluster together processes and their resources, while shared resources are kept together in other clusters [5]. Optimally, a DSL would provide the necessary syntactic sugar to the end-user.

Algebraic Clustering : Example





$$H^{+}_{P_2,\$b} \circ H^{+}_{P_2,not(\$b)} \circ H^{-}_{P_1,\$b}$$

Critic of the approach

Problem

The homomorphisms are very complex because they have to manage every possible substitution, as well as clustering (putting the values in the right clusters).

Solution

Build small and efficient homomorphisms dedicated to a specific substitution and compose them. This is what we call unfolding (of the transitions) [5].

Critic of the approach

Problem

The homomorphisms are very complex because they have to manage every possible substitution, as well as clustering (putting the values in the right clusters).

Solution

Build small and efficient homomorphisms dedicated to a specific substitution and compose them. This is what we call unfolding (of the transitions) [5].

Unfolding

How to discover local behaviors w.r.t the cluster function ?

Unfold each transition. That is, compute all the possible substitutions and keep those that fulfill the guards. By performing this static analysis (before runtime), we enable the grouping of the substitution that are related to the same cluster before runtime and we can build an optimized version of the homomorphisms.

If the domain to unfold is not bounded (domain bound), the user MUST set a bound (user bound).

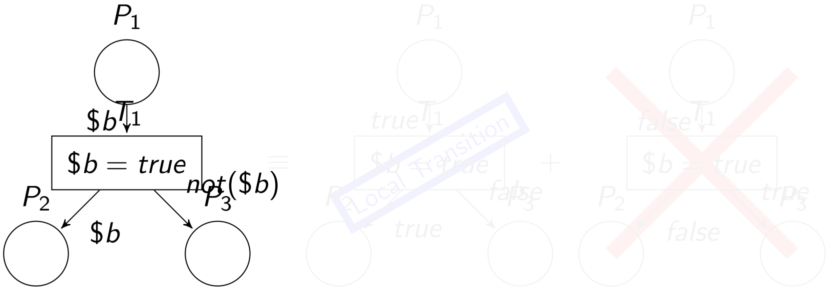
Unfolding

How to discover local behaviors w.r.t the cluster function ?

Unfold each transition. That is, compute all the possible substitutions and keep those that fulfill the guards. By performing this static analysis (before runtime), we enable the grouping of the substitution that are related to the same cluster before runtime and we can build an optimized version of the homomorphisms.

If the domain to unfold is not bounded (domain bound), the user MUST set a bound (user bound).

Unfolding : Example



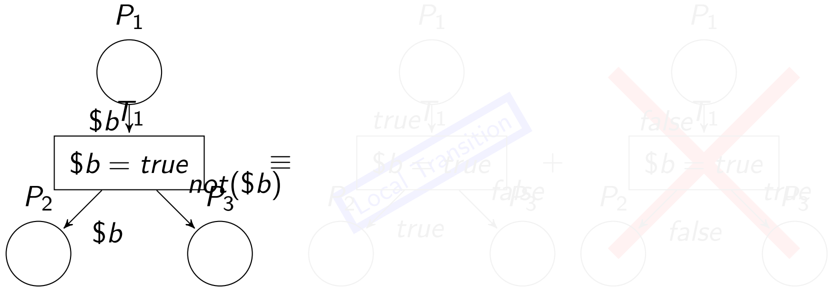
Cluster function

$Cluster(true, P_1) = Cluster(true, P_2) = Cluster(false, P_3) = C_1$

Clustered homomorphisms

$Local(C_1, H^{+}_{P_2,true} \circ H^{+}_{P_3,false} \circ H^{-}_{P_1,true})$

Unfolding : Example



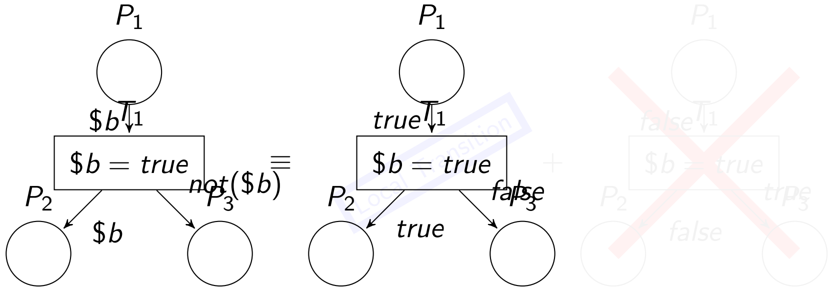
Cluster function

$Cluster(true, P_1) = Cluster(true, P_2) = Cluster(false, P_3) = C_1$

Clustered homomorphisms

$Local(C_1, H^{+}_{P_2,true} \circ H^{+}_{P_3,false} \circ H^{-}_{P_1,true})$

Unfolding : Example



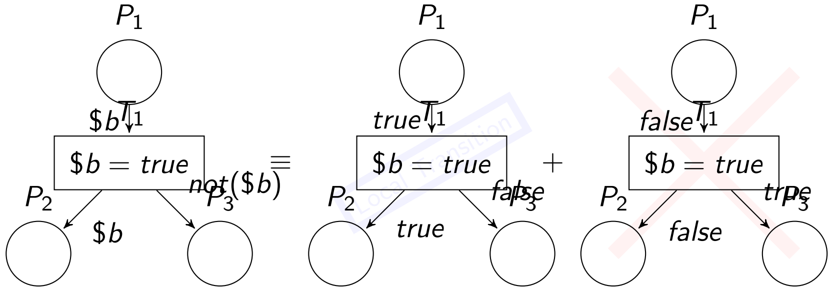
Cluster function

$Cluster(true, P_1) = Cluster(true, P_2) = Cluster(false, P_3) = C_1$

Clustered homomorphisms

$Local(C_1, H^{+}_{P_2,true} \circ H^{+}_{P_3,false} \circ H^{-}_{P_1,true})$

Unfolding : Example



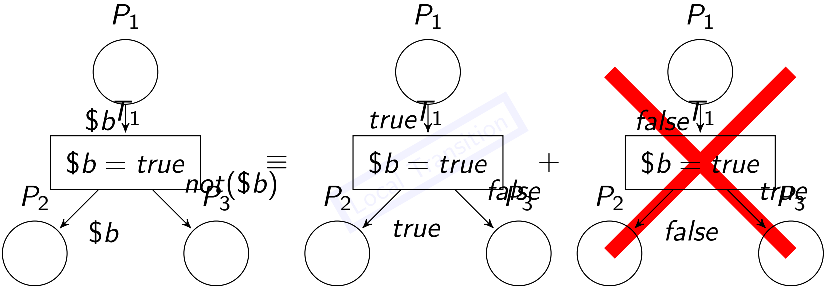
Cluster function

$Cluster(true, P_1) = Cluster(true, P_2) = Cluster(false, P_3) = C_1$

Clustered homomorphisms

$Local(C_1, H^+_{P_2,true} \circ H^+_{P_3,false} \circ H^-_{P_1,true})$

Unfolding : Example



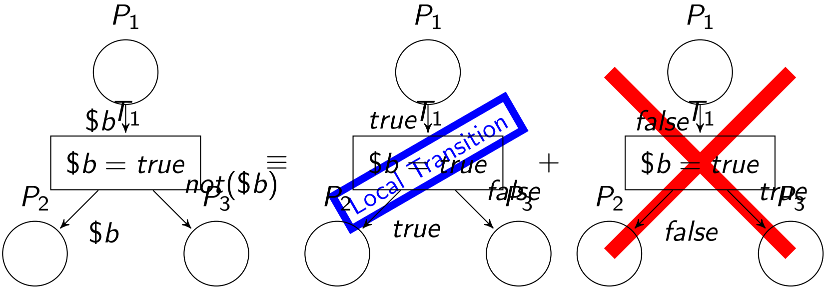
Cluster function

$Cluster(true, P_1) = Cluster(true, P_2) = Cluster(false, P_3) = C_1$

Clustered homomorphisms

$Local(C_1, H^{+}_{P_2,true} \circ H^{+}_{P_3,false} \circ H^{-}_{P_1,true})$

Unfolding : Example



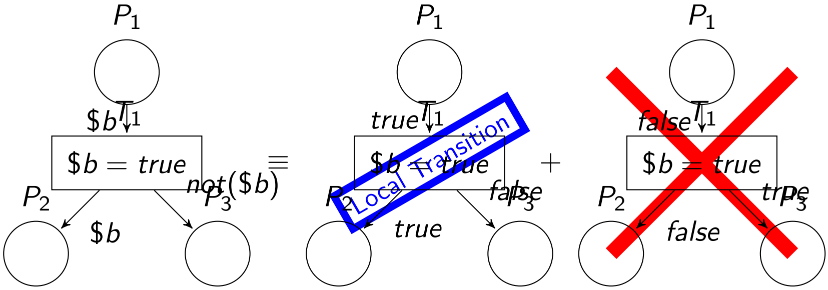
Cluster function

$Cluster(true, P_1) = Cluster(true, P_2) = Cluster(false, P_3) = C_1$

Clustered homomorphisms

$Local(C_1, H^{+}_{P_2,true} \circ H^{+}_{P_3,false} \circ H^{-}_{P_1,true})$

Unfolding : Example



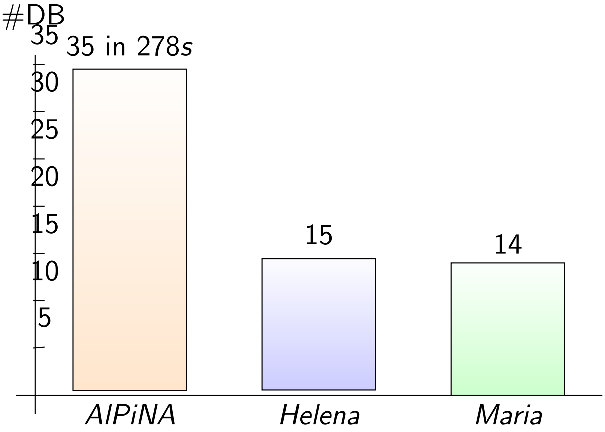
Cluster function

$Cluster(true, P_1) = Cluster(true, P_2) = Cluster(false, P_3) = C_1$

Clustered homomorphisms

$Local(C_1, H^{+}_{P_2,true} \circ H^{+}_{P_3,false} \circ H^{-}_{P_1,true})$

Benchmarks (using **ΣDD**, clustering and unfolding)



Distributed Database model : number of databases the tool was able to calculate the state space for (with time).

Critic of the approach

To unfold user MUST set a bound to unbounded domains

- Unfolding complexity : $O(|A|^{|Places|})$ with $|A|$ the size of the largest algebra and $|Places|$ the number of places.
- If the user-bound is too small then the model checking may be wrong, if it is too high then it may waste resources.

Critic of the approach

To unfold user MUST set a bound to unbounded domains

- Unfolding complexity : $O(|A|^{|Places|})$ with $|A|$ the size of the largest algebra and $|Places|$ the number of places.
- If the user-bound is too small then the model checking may be wrong, if it is too high then it may waste resources.

Critic of the approach

- Even if the domain is bounded it may be difficult to figure out the actual bound
- Sparse domains : 1, 234543, 10000001 $\Leftrightarrow$ unfolding up to 10000001 ? ? ? ?.
- Complex types (lists)

Solution

Not all domains have to be unfolded $\Rightarrow$ Partial net unfolding

Critic of the approach

- Even if the domain is bounded it may be difficult to figure out the actual bound
- Sparse domains : 1, 234543, 10000001 ⇔ unfolding up to 10000001 ? ? ? ?.
- Complex types (lists)

Solution

Not all domains have to be unfolded ⇒ Partial net unfolding

Critic of the approach

- Even if the domain is bounded it may be difficult to figure out the actual bound
- Sparse domains : 1, 234543, 10000001 $\Leftrightarrow$ unfolding up to 10000001 ? ? ? ?.
- Complex types (lists)

Solution

Not all domains have to be unfolded $\Rightarrow$ Partial net unfolding

Critic of the approach

- Even if the domain is bounded it may be difficult to figure out the actual bound
- Sparse domains : 1, 234543, 10000001 $\Leftrightarrow$ unfolding up to 10000001 ? ? ? ?.
- Complex types (lists)

Solution

Not all domains have to be unfolded $\Rightarrow$ Partial net unfolding

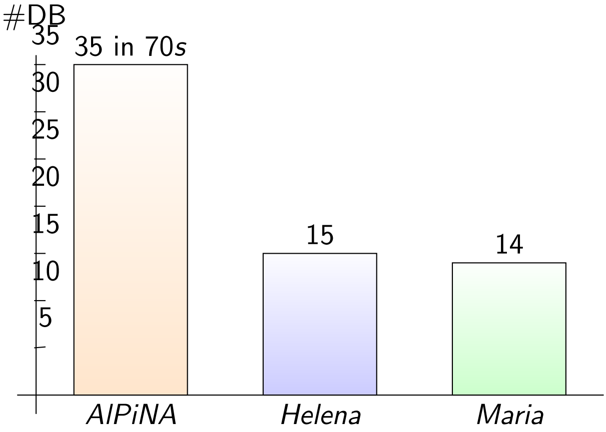
Partial Unfolding [5]

- User decides whether or not if an algebra is unfolded.
- Trade-off between the static analysis (unfolding) and the run-time analysis.

Partial Unfolding [5]

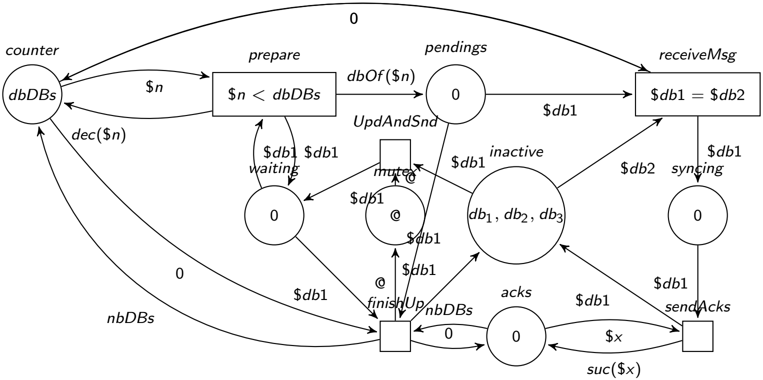
- User decides whether or not if an algebra is unfolded.
- Trade-off between the static analysis (unfolding) and the run-time analysis.

Benchmarks



Distributed Database model : number of databases the tool was able to calculate the state space for (with time).

Distributed Databases : Petri net Example (1)



Distributed Databases : Adt Example (2)

```
import "nat.adt"

Adt Databases
Sorts db;
Generators
    db0 : db;
    d : db -> db;
Operations
    processOf : nat -> db;
    number_of_dbs : nat;
Axioms
    d^3(db0) = db0; //change this to change the number of DBs
    number_of_dbs = suc^3(zero); //change this to change the number of DBs
    //There is no processOf(zero)
    processOf(suc(zero)) = db0;
    if $x != zero then processOf(suc($x)) = d(processOf($x));
Variables
    x : nat;
```

Distributed Databases : Cluster Example (3)

```
import "boolean.adt"
import "db.adt"
import "nat.adt"
import "DDB.apnmm"

Clusters
    3*c0;
Rules
    cluster of Counter is default;
    cluster of db0 in Pending, Inactive, Syncing, Waiting is c0;
    cluster of d($dbv) in Pending, Inactive,
        Syncing, Waiting is next(cluster of $dbv);

Variables
    dbv : db;
    nat : nat;
```

Distributed Databases : Property Example (4)

```
import "boolean.spec"
import "blacktoken.spec"
import "db.adt"
import "nat.spec"
import "DDB.apnmm"

Expressions
    Waiting : card($db in Waiting) =< 1;
    WaitingInactive :
        card($db in Waiting : exists($db1 in Inactive : $db = $db1)) = 0;
    WaitingSyncing :
        card($db in Waiting : exists($db1 in Syncing : $db = $db1)) = 0;
    SyncingInactive :
        card($db in Syncing : exists($db1 in Inactive : $db = $db1)) = 0;
Check
    @Waiting;
Variables
    db : db;
    db1 : db;
```

Q & A

- Why are SDDs not well suited for algebraic Petri nets ?
- Compare SDDs and ΣDDs.
- Explain the difference between topological clustering and algebraic clustering.
- Why is unfolding a possible issue with respect to model checking correctness ?
- Why is unfolding a possible issue with respect to model checking performances ?

Q & A

- Why are SDDs not well suited for algebraic Petri nets ?
- Compare SDDs and ΣDDs.
- Explain the difference between topological clustering and algebraic clustering.
- Why is unfolding a possible issue with respect to model checking correctness ?
- Why is unfolding a possible issue with respect to model checking performances ?

Q & A

- Why are SDDs not well suited for algebraic Petri nets ?
- Compare SDDs and ΣDDs.
- Explain the difference between topological clustering and algebraic clustering.
- Why is unfolding a possible issue with respect to model checking correctness ?
- Why is unfolding a possible issue with respect to model checking performances ?

Q & A

- Why are SDDs not well suited for algebraic Petri nets ?
- Compare SDDs and ΣDDs.
- Explain the difference between topological clustering and algebraic clustering.
- Why is unfolding a possible issue with respect to model checking correctness ?
- Why is unfolding a possible issue with respect to model checking performances ?

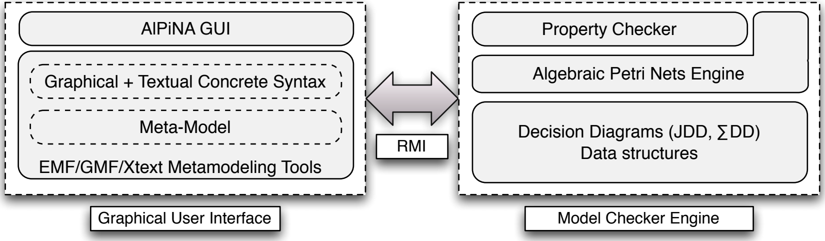
Q & A

- Why are SDDs not well suited for algebraic Petri nets ?
- Compare SDDs and ΣDDs.
- Explain the difference between topological clustering and algebraic clustering.
- Why is unfolding a possible issue with respect to model checking correctness ?
- Why is unfolding a possible issue with respect to model checking performances ?

Benchmarks [7, 8]

		AlPiNA				Maria		Helena	
		Partial Unfold.		Total Unfold.					
Model Size	States #	Mem (MB)	Time (s)	Mem (MB)	Time (s)	Mem (MB)	Time (s)	Mem (MB)	Time (s)
Distributed Database									
10	197E3	10	0.8	12.4	1.3	47	44.3	24	9
15	7.2E7	33	2.6	41	5.8	-	-	1.4E3	7.5E3
35	5.8E17	544	69.4	789	278	-	-	-	-
Dining Philosophers									
10	186E4			1.9	0.15	375	141	11	5
15	2.5E9			2.6	0.18	-	-	409	822
300	1.2E188			162	48.5	-	-	-	-
Slotted Ring									
5	53856			4.9	0.2	23	4.3	10	5
10	8.3E9			55.6	1.7	-	-	-	-
15	1.5E15			330	9.8	-	-	-	-
Leader Election									
10	31302			10.3	0.72	20	3.4	10	7
15	399E4			27.7	1.4	795	361	107	142
50	1.7E21			702	76	-	-	-	-

Architecture [7, 8]



What did we learn so far ?

-

Future Work

- Add modularity to the APN model
- Improve property checking phase
- DSL to express clustering heuristics (process and resources)
- ...

Future Work

- Add modularity to the APN model
- Improve property checking phase
- DSL to express clustering heuristics (process and resources)
- ...

Future Work

- Add modularity to the APN model
- Improve property checking phase
- DSL to express clustering heuristics (process and resources)
- ...

Future Work

- Add modularity to the APN model
- Improve property checking phase
- DSL to express clustering heuristics (process and resources)
- ...

Conclusion

- New symbolic data structure : ΣDD
- Model Checker Tool :
- Reachability analysis for Algebraic Petri Nets
- Algebraic Clustering
- Partial net unfolding

Conclusion

- New symbolic data structure : ΣDD
- Model Checker Tool :
- Reachability analysis for Algebraic Petri Nets
- Algebraic Clustering
- Partial net unfolding

Conclusion

- New symbolic data structure : ΣDD
- Model Checker Tool :
- Reachability analysis for Algebraic Petri Nets
- Algebraic Clustering
- Partial net unfolding

Conclusion

- New symbolic data structure : ΣDD
- Model Checker Tool :
- Reachability analysis for Algebraic Petri Nets
- Algebraic Clustering
- Partial net unfolding

Conclusion

- New symbolic data structure : ΣDD
- Model Checker Tool :
- Reachability analysis for Algebraic Petri Nets
- Algebraic Clustering
- Partial net unfolding

References I

[1] Anonymous. Flyweight pattern [online], undated. Accessed 21 July 2010.

[2] Anonymous. Hash consing [online], undated. Accessed 21 July 2010.

[3] Anonymous. Memoization [online], undated. Accessed 21 July 2010.

[4] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. In *Transactions on Computers, C-35*, pages 677–691. IEEE, 1986.

References II

[5] Didier Buchs and Steve Hostettler. Managing Complexity in Model Checking with Decision Diagrams for Algebraic Petri Net. In Daniel Moldt, editor, *Pre-proceedings of the International Workshop on Petri Nets and Software Engineering*, pages 255–271, 2009. Available at `http://www.informatik.uni-hamburg.de/TGI/events/pnse09/pnse09-proceedings-firstpages.pdf`.

References III

[6] Didier Buchs and Steve Hostettler. Sigma Decision Diagrams : Toward efficient rewriting of sets of terms. In Andrea Corradini, editor, *TERMGRAPH 2009 : Preliminary proceedings of the 5th International Workshop on Computing with Terms and Graphs*, number TR-09-05 in TERMGRAPH workshops, pages 18–32. Università di Pisa, 2009. Available at http://smv.unige.ch/publications/pdfs/termgraph09.pdf.

References IV

[7] Didier Buchs, Steve Hostettler, Alexis Marechal, and Matteo Risoldi. AlPiNA : A symbolic model checker. In Johan Lilius and Wojciech Penczek, editors, *Applications and Theory of Petri Nets - 31st International Conference, PETRI NETS 2010, Braga, Portugal, June 21-25, 2010. Proceedings*, volume 6128 of *Lecture Notes in Computer Science*, pages 287–296. Springer Berlin / Heidelberg, 2010.

References V

[8] Didier Buchs, Steve Hostettler, Alexis Marechal, and Matteo Risoldi. AlPiNA : An Algebraic Petri Net Analyzer. In Javier Esparza and Rupak Majumdar, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 16th International Conference, TACAS 2010, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2010, Paphos, Cyprus, March 20-28, 2010. Proceedings*, volume 6015 of *Lecture Notes in Computer Science*, pages 349–352. Springer, 2010.

[9] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking : 10^{20} states and beyond. *Inf. Comput.*, 98(2) :142–170, 1992.

References VI

[10] Ming-Ying Chung and Gianfranco Ciardo. Saturation now. In *QEST '04 : Proceedings of the The Quantitative Evaluation of Systems, First International Conference*, pages 272–281, Washington, DC, USA, 2004. IEEE Computer Society.

[11] Gianfranco Ciardo, Gerald Lüttgen, and Radu Siminiceanu. Efficient symbolic state-space construction for asynchronous systems. In *Application and Theory of Petri Nets 2000 (Proc. 21th Int. Conf. on Applications and Theory of Petri Nets, Aarhus, Denmark), Lecture Notes in Computer Science 1825*, pages 103–122. Springer-Verlag, 2000.

References VII

[12] Gianfranco Ciardo, Gerald Lüttgen, and Radu Siminiceanu. Saturation : An efficient iteration strategy for symbolic state-space generation. In *TACAS 2001 : Proceedings of the 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 328–342, London, UK, 2001. Springer-Verlag.

[13] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logic of Programs, Workshop*, pages 52–71, London, UK, 1982. Springer-Verlag.

References VIII

[14] Olivier Coudert, Christian Berthet, and Jean Christophe Madre. Formal boolean manipulations for the verification of sequential machines. In *EURO-DAC '90 : Proceedings of the conference on European design automation*, pages 57–61, Los Alamitos, CA, USA, 1990. IEEE Computer Society Press.

[15] Olivier Coudert, Jean Christophe Madre, and Christian Berthet. Verifying temporal properties of sequential machines without building their state diagrams. In *CAV '90 : Proceedings of the 2nd International Workshop on Computer Aided Verification*, pages 23–32, London, UK, 1991. Springer-Verlag.

References IX

[16] Jean-Michel Couvreur, Emmanuelle Encrenaz, Emmanuel Paviot-Adet, Denis Poitrenaud, and Pierre-André Wacrenier. Data decision diagrams for Petri net analysis. In *Proceedings of the 23th International Conference on Application and Theory of Petri Nets (ICATPN'02)*, volume 2360 of *Lecture Notes in Computer Science*, pages 101–120, Adelaide, Australia, June 2002. Springer Verlag.

[17] Jean-Michel Couvreur and Yann Thierry-Mieg. Hierarchical decision diagrams to exploit model structure. In *FORTE*, volume 3731 of *Lecture Notes in Computer Science*, pages 443–457. Springer, 2005.

[18] E. Allen Emerson and Joseph Y. Halpern. "sometimes" and "not never" revisited : on branching versus linear time temporal logic. *J. ACM*, 33(1) :151–178, 1986.

References X

[19] E. Allen Emerson and A. Prasad Sistla, editors. *Computer Aided Verification, 12th International Conference, CAV 2000, Chicago, IL, USA, July 15-19, 2000, Proceedings*, volume 1855 of *Lecture Notes in Computer Science*. Springer, 2000.

[20] Patrice Godefroid. Using partial orders to improve automatic verification methods. In *CAV '90 : Proceedings of the 2nd International Workshop on Computer Aided Verification*, pages 176–185, London, UK, 1991. Springer-Verlag.

[21] Patrice Godefroid and Pierre Wolper. A partial approach to model checking. In *Information and Computation*, pages 406–415, 1994.

References XI

[22] Robert P. Kurshan. *Computer-aided verification of coordinating processes : the automata-theoretic approach*. Princeton University Press, Princeton, NJ, USA, 1994.

[23] Kenneth L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, Norwell, MA, USA, 1993.

[24] Doron Peled. All from one, one for all : on model checking using representatives. In *CAV '93 : Proceedings of the 5th International Conference on Computer Aided Verification*, pages 409–423, London, UK, 1993. Springer-Verlag.

[25] D. Plump. Term graph rewriting. *Handbook of graph grammars and computing by graph transformation : vol. 2 : applications, languages, and tools*, pages 3–61, 1999.

References XII

[26] Amir Pnueli. The Temporal Logic of Programs. In *18th Annual Symposium on Foundations of Computer Science, 31 October-2 November, Providence, Rhode Island, USA*, pages 46–57. IEEE, 1977.

[27] Jean-Pierre Queille and Joseph Sifakis. Specification and verification of concurrent systems in cesar. In *Proceedings of the 5th Colloquium on International Symposium on Programming*, pages 337–351, London, UK, 1982. Springer-Verlag.

[28] Oriol Roig, Jordi Cortadella, and Enric Pastor. Verification of asynchronous circuits by bdd-based model checking of petri nets. In *In 16th Int. Conf. on Application and Theory of Petri Nets, volume 935 of LNCS*, pages 374–391. Springer-Verlag, 1996.

References XIII

[29] Antti Valmari. A stubborn attack on state explosion. *Form. Methods Syst. Des.*, 1(4) :297–322, 1992.

[30] Antti Valmari. The state explosion problem. In *Lectures on Petri Nets I : Basic Models, Advances in Petri Nets, the volumes are based on the Advanced Course on Petri Nets*, pages 429–528, London, UK, 1998. Springer-Verlag.