

Model Checking Applications: Petri Nets and CTL

Dimitri Racordon, Damien Morard
Didier Buchs

Centre Universitaire d'Informatique, Université de Genève

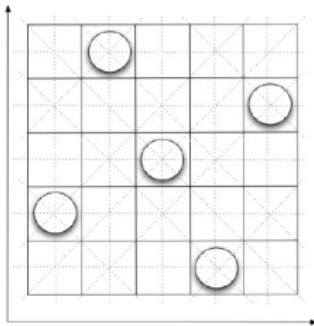
December 2, 2021

- Compute on the whole set instead of on element one by one.
- Needs homomorphic operation in order to have both approaches equivalent
- Decision Diagram operations can be optimised at the operation level and at the implementation level
 - rewriting of the homomorphisms
 - Using memoization
 - Hash consing / flyweight pattern
 - uniqueness implies fast equality checking

Exemple: Problem solving: queen on a chess board, path in graph computing, covering sets, state space computations...

Solving problems: The N-queen problem

- The N-Queens problem : How to put n queens on a standard $N \times N$ chess board?
- Define the problem as a set of configuration
 $Q_{i,j}, 0 \leq i \leq N - 1, 0 \leq j \leq N - 1.$
- the order is given by
 $Q_{i,j} > Q_{i',j'} \Leftrightarrow \max(i,j) > \max(i',j') \wedge \min(i,j) > \min(i',j')$
- Build a SFDD of configurations and reduce it progressively.



Solving problems: The N-queen problem

$S_0 = \text{enc}(\wp(\{Q_{i,j} | 0 \leq i \leq N-1, 0 \leq j \leq N-1\}))$

$S_1 = \text{size}(S_0, N)$

$S_2 = \text{checkhor}(S_1)$

$S_3 = \text{checkver}(S_2)$

$S_4 = \text{checkdiagbashaout}(S_3)$

$S_5 = \text{checkdiaghautbas}(S_4)$

Solving problems: The N-queen problem

$\text{size}(S, N)$

$$\text{size}(\perp, N) = \perp$$

$$\text{size}(\top, 0) = \top$$

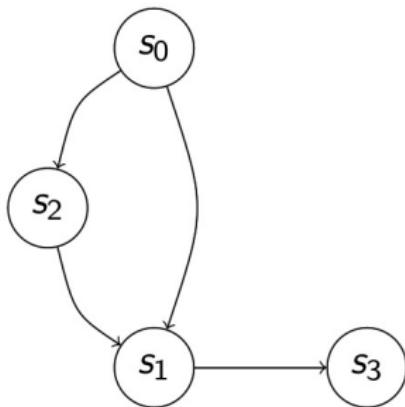
$$\text{size}(\top, \text{succ}(N)) = \perp$$

$$\text{size}(\langle t, \tau, \sigma \rangle, 0) = \perp$$

$$\text{size}(\langle t, \tau, \sigma \rangle, \text{succ}(N)) = \langle t, \text{size}(\tau, N), \text{size}(\sigma, \text{succ}(N)) \rangle$$

Set Family Decision Diagrams

Path computations



- $S = \{s_0, s_1, s_2, s_3\}$
 - $S_0 = \{s_0\}$
 - $R = \{(s_0, s_1), (s_0, s_2), (s_2, s_1), (s_1, s_3)\}$
- $K = \langle S, S_0, R \rangle$ where:

$$Paths(s_a) = \left(\bigcup_{(s_a, s_b) \in R} Paths(s_b) \right) \oplus s_a$$

This can be extended to sets:

$$Paths(\emptyset) = enc(\emptyset)$$

$$Paths(\{s_a\} \cup S') = \left(\bigcup_{(s_a, s_b) \in R} Paths(s_b) \right) \oplus s_a \cup Paths(S')$$

Set Family Decision Diagrams

Path computations for non cyclic graphs

$$\begin{aligned} \text{Paths}(s_0) &= (\text{Paths}(s_1) \cup \text{Paths}(s_2)) \oplus s_0 = \\ &\quad \text{enc}_S(\{\{s_0, s_1, s_2, s_3\}, \{s_1, s_0, s_3\}\}) \end{aligned}$$

$$\text{Paths}(s_1) = (\text{Paths}(s_3) \oplus s_1) = \text{enc}_S(\{\{s_1, s_3\}\})$$

$$\text{Paths}(s_2) = (\text{Paths}(s_1) \oplus s_2) = \text{enc}_S(\{\{s_1, s_2, s_3\}\})$$

$$\text{Paths}(s_3) = \text{enc}_S(\{\{s_3\}\})$$

Set Family Decision Diagrams

Path computations for cyclic graphs (1)

$$\text{Next}(s_a) = \bigcup_{(s_a, s_b) \in R} \{s_b\}$$

This can be defined on sets recursively for a length one:

$$\text{OnePaths}(\{s_a\} \cup S') = \text{Next}(s_a) \oplus \{s_a\} \cup \text{OnePaths}(S')$$

$$\text{OnePaths}(\emptyset) = \emptyset$$

Nevertheless not very efficient for large graph as we need to take elements of the sets one by one. This is an efficiency problem not directly solvable by DD in this case, as we need to keep track of the causality.

This principle has to be applied on families, i.e. the set of paths:

$$\text{MulPaths}(f \cup S') = \text{OnePaths}(f) \cup \text{MulPaths}(S')$$

$$\text{MulPaths}(\emptyset) = \emptyset$$

Set Family Decision Diagrams

Path computations for cyclic graphs(N)

It must be reapplied for larger length (in fact bounded by the number of nodes)

$$\text{MulPaths}_0 = \bigcup_{(s_a) \in S} \{s_a\}$$

$$\text{MulPaths}_1 = \text{MulPaths}(\text{MulPaths}_0))$$

$$\text{MulPaths}_2 = \text{MulPaths}(\text{MulPaths}_1))$$

...

$$\text{MulPaths}_n = \text{MulPaths}(\text{MulPaths}_{n-1}))$$

So it means that there is a solution to the fixpoint problem that we can express by: $\mu X. (\text{MulPaths}(X) \cup \bigcup_{(s_a) \in S} \{s_a\})$

Set Family Decision Diagrams

Petri nets

Petri nets are defined as $\langle P, T, \text{Pre}, \text{Post} \rangle$ where:

- P and T are finite disjoint sets.
- Pre and Post are functions $P \times T \rightarrow \mathbb{N}$

The state of a Petri net is the marking $M : P \rightarrow \mathbb{N}$.

A transition $t \in T$ is fireable if and only if

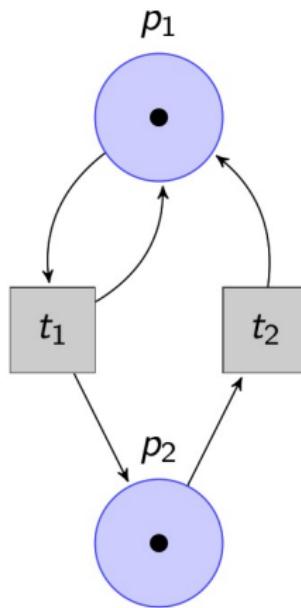
$$\forall p \in P, \text{Pre}(p, t) \leq M(p)$$

The firing of a transition modifies the marking (i.e. state):

$$\forall p \in P, M'(p) = M(p) + \text{Post}(p, t) - \text{Pre}(p, t)$$

Set Family Decision Diagrams

Petri nets



Set Family Decision Diagrams

Encoding safe PN marking in sets

($\circ \circ \circ$ jehe per place)

Encoding a safe Petri net marking M with a set S_M can be done using simply P as terms:

$$\underline{S_M} = \bigcup_{\substack{p \in P, \\ \underline{M(p)=1}}} \{p\}$$

Example:

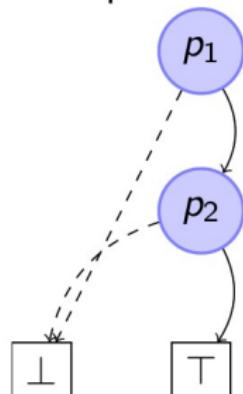
$$S_M = \{p_1, p_2\}$$

Set Family Decision Diagrams

Encoding safe PN marking in sets

which is encoded directly in SFDD as: $\text{enc}(\bigcup_{p \in P, M(p)=1} \{p\})$
with the arbitrary total order $P = \{p_1, p_2, \dots, p_k\}$ and
 $p_1 < p_2 < \dots < p_k$

Example:



Set Family Decision Diagrams

State Space Computation, state by state

Algorithm 1: State space computation on individual states

Input: s_0 : initial state.

Input: T : set of transition.

Result: set of reachable states

begin

s_{rem}, s : set of states ;

m, mt : states ;

$s_{rem} \leftarrow \{s_0\}$; $s \leftarrow \{\}$;

repeat

$m \leftarrow \text{choose}(s_{rem})$;

$s_{rem} \leftarrow s_{rem}/\{m\}$;

foreach $t \in T$ **do**

if $\text{fireable}(t, m)$ **then**

$mt \leftarrow t(m)$;

if $m \notin s$ **then** $s \leftarrow s \cup \{mt\}$; $s_{rem} \leftarrow s_{rem} \cup \{mt\}$;

until $s_{rem} = \emptyset$;

return s ;

Set Family Decision Diagrams

Global computation of state space

Algorithm 2: Global state space computation

Input: s_0 : initial state.

Input: Φ : 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 $t \in \Phi$ **do**

$temp \leftarrow t(s)$;

$s \leftarrow s \cup temp$;

until $s = s_{old}$;

return s ;

Set Family Decision Diagrams

Global computation of state space

What about $t(s)$?

$$t(m) = m + \text{post}(t) - \text{pre}(t)$$

If pre and post are functions on transition and markings.

$$t(m) = \text{post}(t, \text{pre}(t, m))$$

Extended to set of states:

$$\begin{aligned} t(s \cup \{m\}) &= t(s) \cup \{\text{post}(t, \text{pre}(t, m))\} \\ t(\emptyset) &= \emptyset \end{aligned}$$

Remark: - operator will propagate 'non firability', represented by empty set, when not enough tokens.

a code + (s)

Set Family Decision Diagrams

Encoding safe PN pre and post conditions

$$t = \text{post}(t) \circ \text{pre}(t)$$

✓ Pouch

$$\text{pre}(t) = \text{pre}(t, p_1) \circ \text{pre}(t, p_2) \circ \cdots \circ \text{pre}(t, p_n)$$

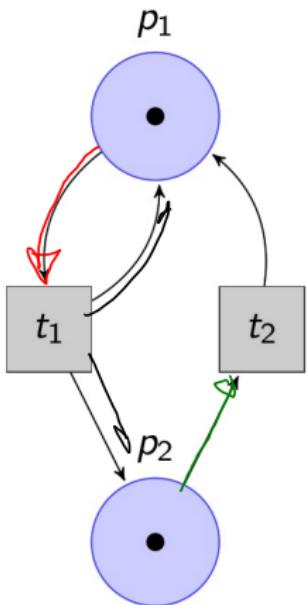
$$\text{post}(t) = \text{post}(t, p_1) \circ \text{post}(t, p_2) \circ \cdots \circ \text{post}(t, p_1)$$

$$\text{pre}(t, p_i) = \begin{cases} \ominus(p_i) \circ \text{filter}(p_i) & \text{if } \text{Pre}(t, p_i) \neq 0 \\ (\text{id}) & \text{otherwise} \end{cases}$$

$$\text{post}(t, p_i) = \begin{cases} \oplus(p_i) & \text{if } \text{Post}(t, p_i) \neq 0 \\ (\text{id}) & \text{otherwise} \end{cases}$$

Set Family Decision Diagrams

Petri nets



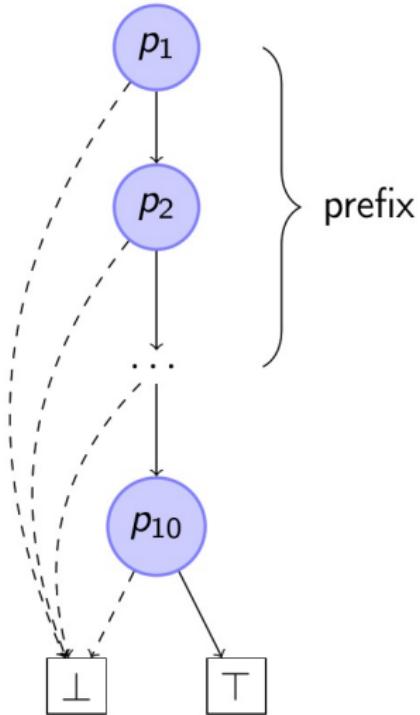
$$\begin{aligned} \underline{\text{pre}}(t_1, p_1) &= \ominus(p_1 \cancel{\cap} p_1) \circ \underline{\text{filter}}(p_1) \\ \underline{\text{pre}}(t_1, p_2) &= id \\ \underline{\text{pre}}(t_2, p_1) &= id \\ \underline{\text{pre}}(t_2, p_2) &= \ominus(p_2) \circ \underline{\text{filter}}(p_2) \end{aligned}$$

$$\begin{aligned} \underline{\text{post}}(t_1, p_1) &= \oplus(p_1) \\ \underline{\text{post}}(t_1, p_2) &= \oplus(p_2) \\ \underline{\text{post}}(t_2, p_1) &= \oplus(p_1) \\ \underline{\text{post}}(t_2, p_2) &= id \end{aligned}$$

Set Family Decision Diagrams

Optimizations

Homomorphisms may involve unnecessary operations on large prefixes:



Set Family Decision Diagrams

Optimizations

The idea is to dive as deep as possible before applying an homomorphism:

$$\text{dive}(k, \phi)(\perp) = \perp$$

$$\text{dive}(k, \phi)(\top) = \top$$

$$\text{dive}(k, \phi)(\langle t, \tau, \sigma \rangle) = \begin{cases} \langle t, \text{dive}(k, \phi)(\tau), \text{dive}(k, \phi)(\sigma) \rangle & \text{if } t < k \\ \phi(\langle t, \tau, \sigma \rangle) & \text{if } t = k \\ \langle t, \tau, \sigma \rangle & \text{if } t > k \end{cases}$$

Set Family Decision Diagrams

Optimizations

Grouping homomorphisms that work on close variables can avoid processing long prefixes multiple times:

$$\text{filter}(p_8) \circ \text{filter}(p_{10}) \equiv \text{dive}(p_8, \text{filter}(p_8) \circ \text{filter}(p_{10}))$$

Some homomorphism may be reordered so they can be grouped:

$$\text{filter}(p_i) \circ \text{filter}(p_j) \equiv \text{filter}(p_j) \circ \text{filter}(p_i)$$

Set Family Decision Diagrams

CTL model checking

We need to proceed as:

- encoding the Kripke structure
- Define homomorphisms Pre and Post on states encoded using $\text{post}(t) \circ \text{pre}(t)$
- Define homomorphism $\text{PreE}(S)$ of predecessors
- Fixpoint computations using CTL model checking algorithms

Kripke Structure

definition

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

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
- $S_0 \subseteq S$ is a non-empty set of initial states

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
- $S_0 \subseteq S$ is a non-empty set of initial states
- $R \subseteq S \times S$ is a left-total binary relation on S representing the transitions

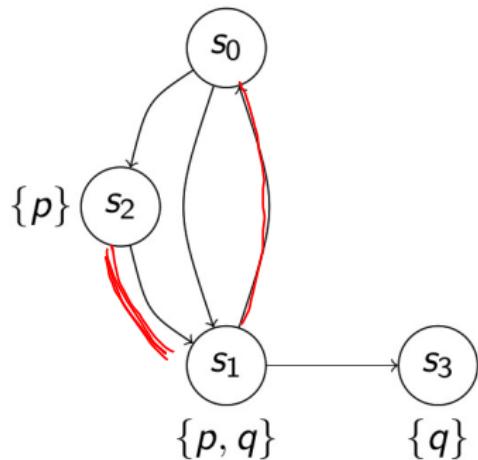
Definition

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

Kripke Structure

example



$$\begin{array}{c} \{p\} \rightarrow \{p', q'\} \\ \cancel{\{p, q\} \rightarrow \emptyset} \end{array} = \overbrace{\{p, p', q'\}}^{\cancel{\{p, q\}}} = \overbrace{\{p', q'\}}^{\cancel{\{p, q\}}}$$

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

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

The rest of the development is valid if the labeling function allows to uniquely determine states by labels, i.e. L is injective. This is obviously true if the Kripke structure has been computed and labelled by the state of a Petri net.

$$\begin{array}{llll}
 S_0 \rightarrow S_1 & \left\{ \begin{matrix} \{ \} \\ \{ p, q \} \end{matrix} \right. & \left\{ \begin{matrix} \{ p, q \} \\ \{ q \} \\ \{ q' \} \end{matrix} \right. & \left\{ \begin{matrix} \{ p', q' \} \\ \{ \cancel{\{ p, q \}}, \cancel{\{ q \}} \} \\ \{ p, q, q' \} \end{matrix} \right. \\
 S_1 \rightarrow S_3 & \left. \begin{matrix} \{ p, q \} \\ \{ q \} \end{matrix} \right. & \left. \begin{matrix} \downarrow \\ \text{remove} \end{matrix} \right. & \left. \begin{matrix} \{ \} \\ \{ p, q \} \end{matrix} \right. \\
 \cdot & & &
 \end{array}$$

$$q' = \underline{\sin(b(q))}$$

From Kripke Structure to SFDD

- Given AP, we create a sibling set $\underline{AP'}$ different from AP:
 $\underline{AP} \cap \underline{AP'} = \emptyset$ and a bijective function $\underline{sib} : \underline{AP} \rightarrow \underline{AP'}$
- We create also an order on $\underline{AP \cup AP'} <'$ from the order $<$ such as $\forall s_a$ and $s_b \in AP$:
 - $s_a < s_b \Rightarrow s_a <' sib(s_a) <' s_b$
 - $sib(s_a) <' sib(s_b) \Rightarrow sib(s_a) <' s_b <' sib(s_b)$

We can prove that $\forall s_a$ and $s_b \in AP$:

$$s_a < s_b \Leftrightarrow s_a <' s_b \Leftrightarrow sib(s_a) <' sib(s_b)$$

Example

$AP = \{p, q\}$, $AP' = \{p', q'\}$ and $sib(p) = p'$ and $sib(q) = q'$.^a

We also have the orders:

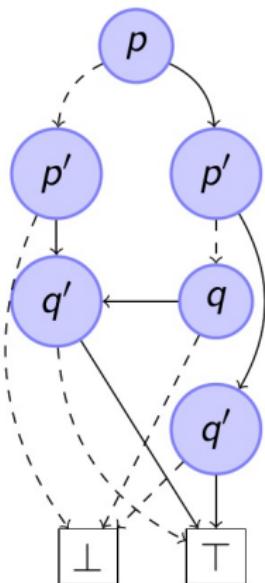
$$p < q \text{ and } p <' p' <' q <' q'$$

^asib is naturally extended to $sib : \mathcal{P}(AP) \rightarrow \mathcal{P}(AP')$ and $sib^{-1} : AP' \rightarrow AP$

From Kripke Structure to SFDD(2)

Given $K = \langle S, S_0, R, L \rangle$ the SFDD that we will build is:

$$G_K = \bigcup_{(s_a, s_b) \in R} enc_{AP \cup AP'}(\{L(s_a) \cup sib(L(s_b))\})$$



Shannon decomposition on sets

Transition relation encoded as sets:

$$S = \{\{\bar{p}, \bar{q}, p', q'\}, \{p, q, \bar{p}', \bar{q}'\}, \{\bar{p}, \bar{q}, p', \bar{q}'\}, \{p, \bar{q}, p', q'\}, \{p, q, \bar{p}', q'\}\}$$

Reorganize the sets: two sets prefixed by p or by \bar{p}

$$S = \{\{\bar{p}, \bar{q}, p', q'\}, \{\bar{p}, \bar{q}, p', \bar{q}'\}\} \cup \{\{p, q, \bar{p}', \bar{q}'\}, \{p, \bar{q}, p', q'\}, \{p, q, \bar{p}', q'\}\}$$

$$S = \{\bar{p}\} \otimes \{\{\bar{q}, p', q'\}, \{\bar{q}, p', \bar{q}'\}\} \cup \{p\} \otimes \{\{\bar{p}', q, \bar{q}'\}, \{p', \bar{q}, q'\}, \{\bar{p}', q, q'\}\}$$

Two sets prefixed by p' or by \bar{p}'

$$S = \{\bar{p}\} \otimes \{p'\} \otimes \{\{\bar{q}, q'\}, \{\bar{q}, \bar{q}'\}\} \cup \{p\} \otimes (\{\bar{p}'\} \otimes \{\{q, \bar{q}'\}, \{q, q'\}\}) \cup \{p'\} \otimes \{\{\bar{q}, q'\}\})$$

Two sets prefixed by q or by \bar{q}

$$S = \{\bar{p}\} \otimes \{p'\} \otimes \{\bar{q}\} \otimes \{\{q'\}, \{\bar{q}'\}\} \cup \{p\} \otimes (\{\bar{p}'\} \otimes \{q\} \otimes \{\{\bar{q}'\}, \{q'\}\}) \cup \{p'\} \otimes \{\bar{q}\} \otimes \{\{q'\}\})$$

- Only need algorithms for EX, EU, EG since:

- $AX\phi \iff \neg EX(\neg\phi)$
- $AF\phi \iff \neg EG(\neg\phi)$
- $AG\phi \iff \neg EF(\neg\phi)$
- $EF\phi \iff E[\text{true} \cup \phi]$
- $A[\phi \cup \theta] \iff \neg E[\neg\theta \cup (\neg\phi \wedge \neg\theta)] \wedge \neg EG(\neg\theta)$

- Let F be the set of states ($\in SFDD$) satisfying ϕ :

Algorithm 3: temporal logic computations

Input: $F : \phi$ satisfying states.

Result: set of states satisfying $EX(\phi)$

begin

s : set of states ;

$s \leftarrow PreE(F)$;

return s ;

Function PreE

$$reduce_T(H, T') = H \ominus (T - T')$$

element de T'

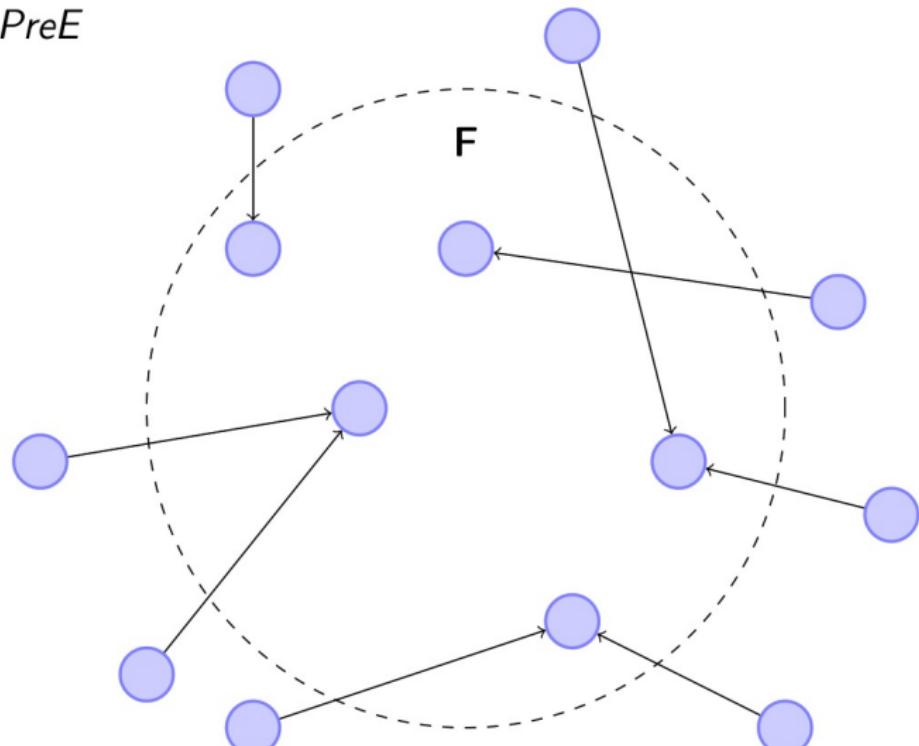
$$PreE(F) = reduce_{AP \cup AP'}(G_K \cap (enc(\mathcal{P}(AP)) \otimes enc(sib(F))), AP)$$

*

where \otimes is the cartesian product on SFDD. (it can be defined with the insertion \oplus on each elements.)

For disjoint set of AP and AP' it is a concatenation of the nodes.

$PreE$



- Let F (resp. G) be the set of states ($\in SFDD$) satisfying ϕ (resp. Φ):

Algorithm 4: temporal logic computations

Input: F : states satisfying ϕ .

Result: set of states satisfying $E(\phi Until \Phi)$

begin

N, S : set of states ;

$S \leftarrow G$;

$N \leftarrow enc(\emptyset)$;

while $N \neq S$ **do**

$N \leftarrow S$;

$S \leftarrow S \cup (F \cap preE(S))$;

return s ;

- Let F be the set of states ($\in SFDD$) satisfying ϕ

Algorithm 5: temporal logic computations

Input: F : states satisfying ϕ .

Result: set of states satisfying $EG(\phi)$

begin

N, S : set of states ;

$S \leftarrow F$;

$N \leftarrow enc(\emptyset)$;

while $N \neq S$ **do**

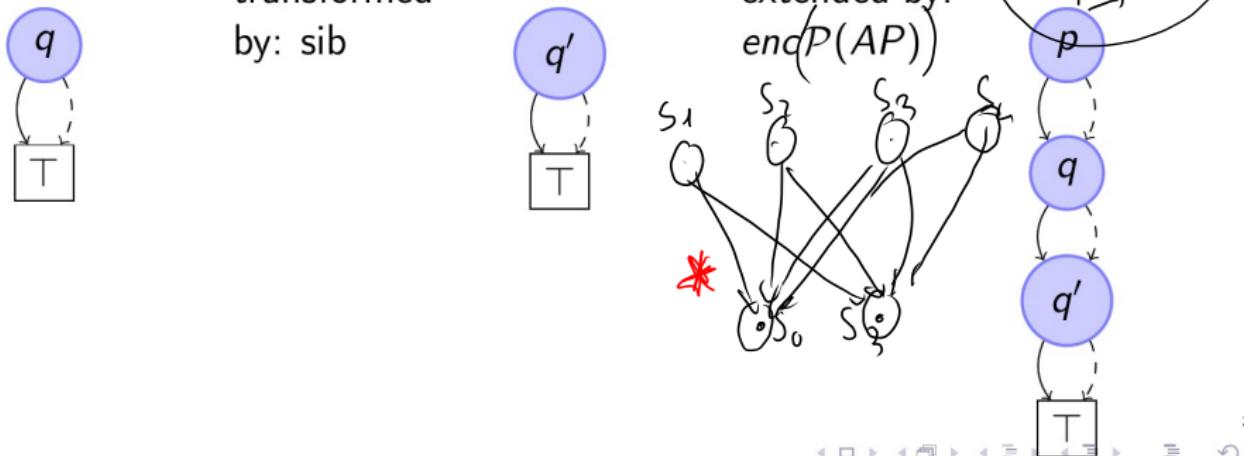
$N \leftarrow S$;

$S \leftarrow S \cap preE(S)$;

return s ;

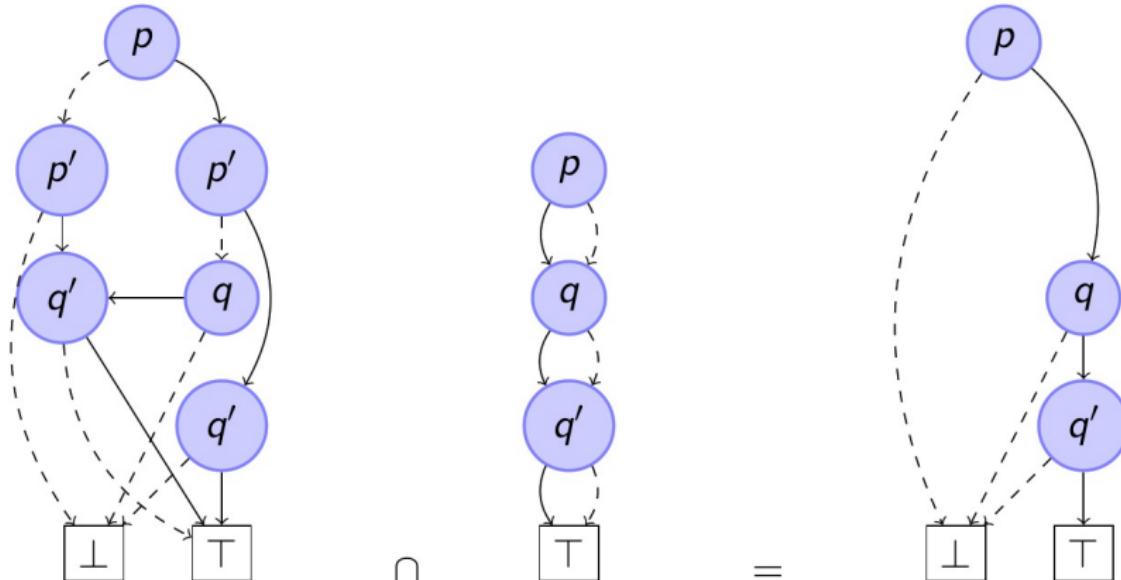
- Let's compute the set of states that satisfy $EX(\neg p)$:

The states satisfying $\neg p$ are: s_0, s_3 which are the states where the $\{\emptyset, \{q\}\}$ atomic propositions are valid

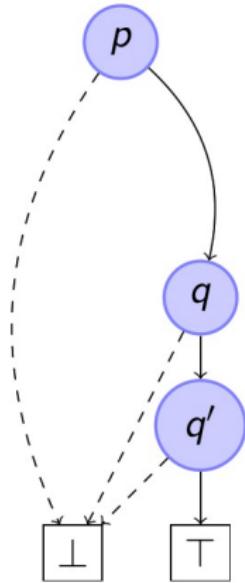


$EX(\neg p)$

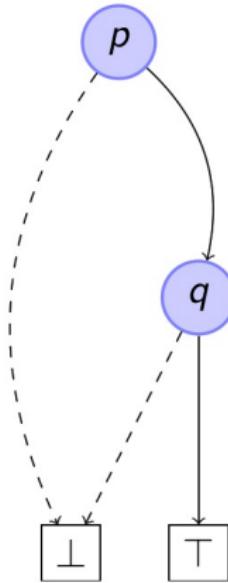
- Let's compute the intersection:



- Let's compute the reduction to AP :



reduced
into

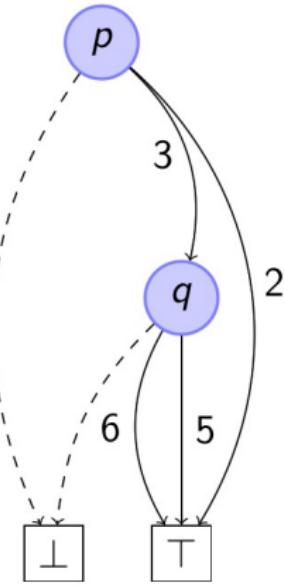
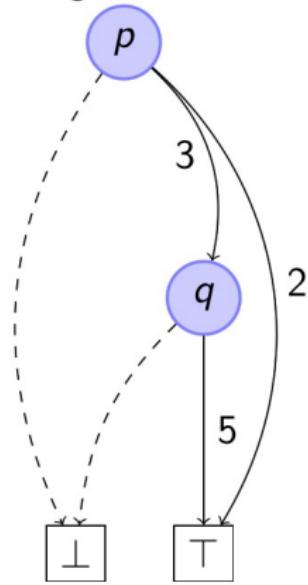


which means that $\{p, q\}$ is the only state s_1 satisfying $EX(\neg p)$.

Dealing with non-safe Petri nets

Sets of elements are functions from elements to boolean and so multi-sets are functions from elements to integers.

The general model extending SFDD is MFDD:



$$S = \{\{p \rightarrow 3, q \rightarrow 5\}, \{p \rightarrow 2\}\} \quad S' = \{\{p \rightarrow 3, q \rightarrow 6\}, \{p \rightarrow 3, q \rightarrow 5\}, \{p \rightarrow 2\}\}$$

Set Family Decision Diagrams

Conclusion

- SFDD encoding of sets
- SFDD properties such as canonization
- Homomorphic operations on SFDD
- Inductive homomorphisms as pattern of computation
- Encoding of PN markings and set of markings for safe nets
- Encoding of PN fire functions
- Computation of PN state space
- Computation of CTL Formulae

References I