Model Checking Applications: Petri Nets and CTL

Dimitri Racordon, Damien Morard
Didier Buchs

Centre Universitaire d'Informatique, Université de Genève

December 2, 2021

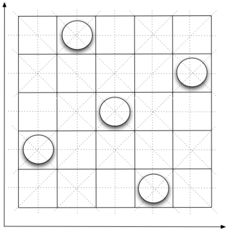
Application

- 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 x 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

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

$$S_1 = size(S_0, N)$$

$$S_2 = checkhor(S_1)$$

$$S_3 = checkver(S_2)$$

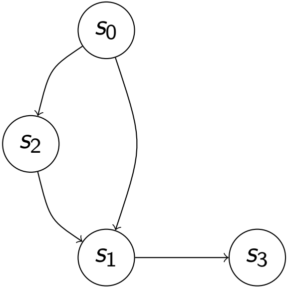
$$S_4 = checkdiagbashaut(S_3)$$

$$S_5 = checkdiaghautbas(S_4)$$

Solving problems: The N-queen problem

size(S,N)

$$
\begin{aligned}
size(\bot, N) &= \bot \\
size(\top, 0) &= \top \\
size(\top, succ(N)) &= \bot \\
size(\langle t, \tau, \sigma \rangle, 0) &= \bot \\
size(\langle t, \tau, \sigma \rangle, succ(N)) &= \langle t, size(\tau, N), size(\sigma, succ(N)) \rangle
\end{aligned}
$$



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

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

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

This can be extended to sets:

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

$$Paths(s_0) = (Paths(s_1) \cup Paths(s_2)) \oplus s_0 =$$
$$enc_S(\{\{s_0, s_1, s_2, s_3\}, \{s_1, s_0, s_3\}\})$$

$$Paths(s_1) = (Paths(s_3) \oplus s_1) = enc_S(\{\{s_1, s_3\}\})$$

$$Paths(s_2) = (Paths(s_1) \oplus s_2) = enc_S(\{\{s_1, s_2, s_3\}\})$$

$$Paths(s_3) = enc_S(\{\{s_3\}\})$$

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

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

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

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:

$$MulPaths(f \cup S') = OnePaths(f) \cup MulPaths(S')$$
$$MulPaths(\emptyset) = \emptyset$$

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

$$MulPaths_0 = \bigcup_{(s_a) \in S} \{s_a\}$$

$$MulPaths_1 = MulPaths(MulPaths_0))$$

$$MulPaths_2 = MulPaths(MulPaths_1))$$

...

$$MulPaths_n = MulPaths(MulPaths_{n-1}))$$

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

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

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

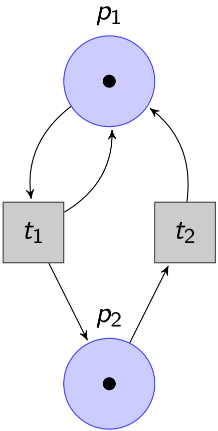
The state of a Petri net is the marking $M : P \to \mathbb{N}$.

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

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

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

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


p_1
t_1
t_2
p_2

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

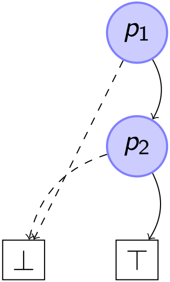
$$S_M = \bigcup_{p \in P, M(p)=1} \{p\}$$

Example:

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

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

Example:



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
    $m \leftarrow choose(s_{rem})$;
    $s_{rem} \leftarrow s_{rem}/\{m\}$;
    foreach $t \in T$ **do**
      if *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;

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 ← {s_0} ;
    repeat
        s_old ← s ;
        foreach t ∈ Φ do
            temp ← t(s) ;
            s ← s ∪ temp ;
    until s = s_old;
    return s;
```

What about $t(s)$?

$$t(m) = m + post(t) - pre(t)$$

If *pre* and *post* are functions on transition and markings.

$$t(m) = post(t, pre(t, m))$$

Extended to set of states:

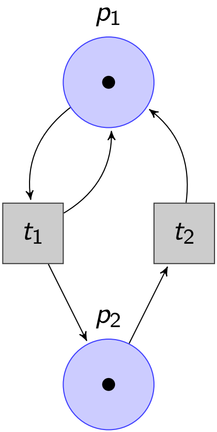
$$\begin{aligned} t(s \cup \{m\}) &= t(s) \cup \{post(t, pre(t, m))\} \\ t(\varnothing) &= \varnothing \end{aligned}$$

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

Set Family Decision Diagrams

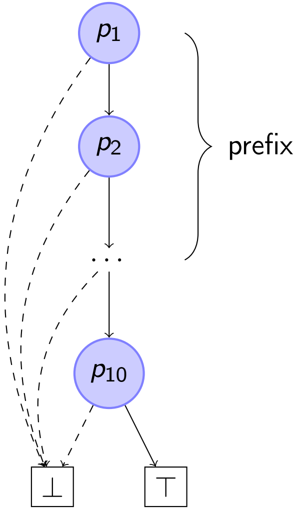
Encoding safe PN pre and post conditions

$$
\begin{aligned}
t &= post(t) \circ pre(t) \\
pre(t) &= pre(t, p_1) \circ pre(t, p_2) \circ \cdots \circ pre(t, p_n) \\
post(t) &= post(t, p_1) \circ post(t, p_2) \circ \cdots \circ post(t, p_1) \\
pre(t, p_i) &= \begin{cases} \ominus(p_i) \circ \mathrm{filter}(p_i) & \text{if } Pre(t, p_i) \neq 0 \\ (id) & \text{otherwise} \end{cases} \\
post(t, p_i) &= \begin{cases} \oplus(p_i) & \text{if } Post(t, p_i) \neq 0 \\ (id) & \text{otherwise} \end{cases}
\end{aligned}
$$



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

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



Homomorphisms may involve uncessary operations on large prefixes:

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

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

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

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

Some homomorphism may be reordered so they can be grouped:

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

We need to proceed as:

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

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

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

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

Kripke Structure
definition

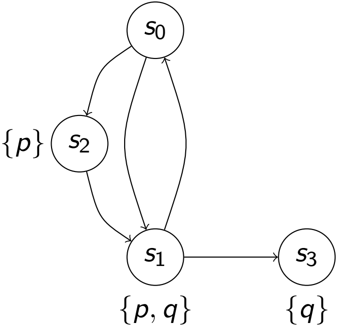
Definition

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
- $\mathrm{S}_0 \subseteq \mathrm{S}$ is a non-empty set of initial states
- $\mathrm{R} \subseteq \mathrm{S} \times \mathrm{S}$ is a left-total binary relation on S representing the transitions
- $\mathrm{L} : \mathrm{S} \rightarrow \mathcal{P}(\mathrm{AP})$ 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:

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

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.

From Kripke Structure to SFDD

- Given AP, we create a sibling set AP' different from AP: $AP \cap AP' = \emptyset$ and a bijective function $sib : AP \to AP'$
- We create also an order on $AP \cup AP'$ $<'$ 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

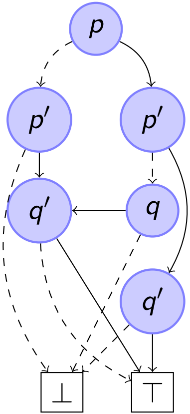
$AP = \{p, q\}$, $AP' = \{p', q'\}$ and $sib(p) = p'$ and $sib(q) = q'$.[a]
We also have the orders:
$p < q$ and $p <' p' <' q <' q'$

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

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:

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

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

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

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

Two sets prefixed by p' or by $\overline{p'}$

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

Two sets prefixed by q or by $\overline{q}$

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

CTL

- Only need algorithms for EX, EU, EG since:
  - $AX\phi \iff \neg EX(\neg\phi)$
  - $AF\phi \iff \neg EG(\neg\phi)$
  - $AG\phi \iff \neg EF(\neg\phi)$
  - $EF\phi \iff E[true \cup \phi]$
  - $A[\phi \cup \theta] \iff \neg E[\neg\theta \cup (\neg\phi \wedge \neg\theta)] \wedge \neg EG(\neg\theta)$

$EX(\phi)$

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

Algorithm 3: temporal logic computations

Input: F : ϕ satisfying states.
Result: set of states satisfying $EX(\phi)$
begin
  s : set of states ;
  $s \leftarrow PreE(F)$;
  return s;

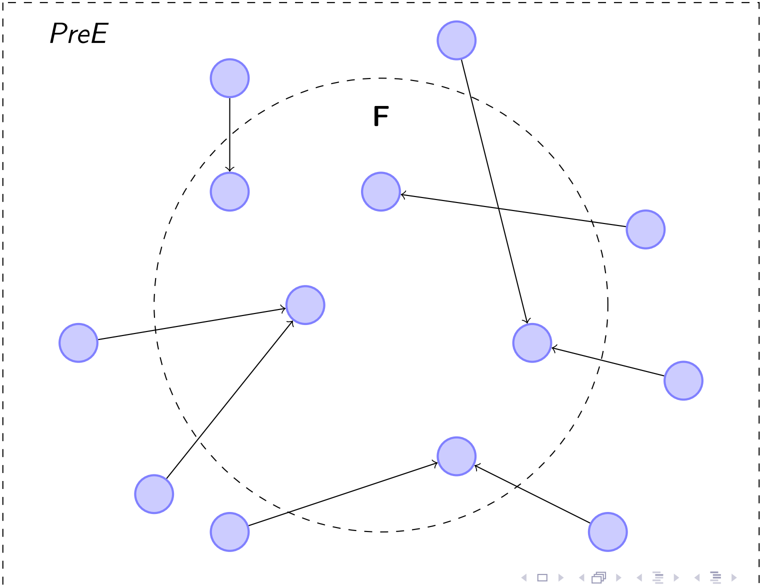
Function PreE

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

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

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.

$PreE(F)$



$E(\phi Until \Phi)$

- 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;

$EG(\phi)$

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

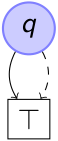
Algorithm 5: temporal logic computations

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;

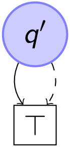
$EX(\neg p)$

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

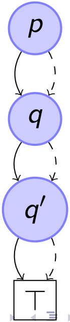
The states satisfying $\neg p$ are: s_0, s_3 which are the states where the $\{\emptyset, \{q\}\}$ atomic propositions are valid



transformed by: sib

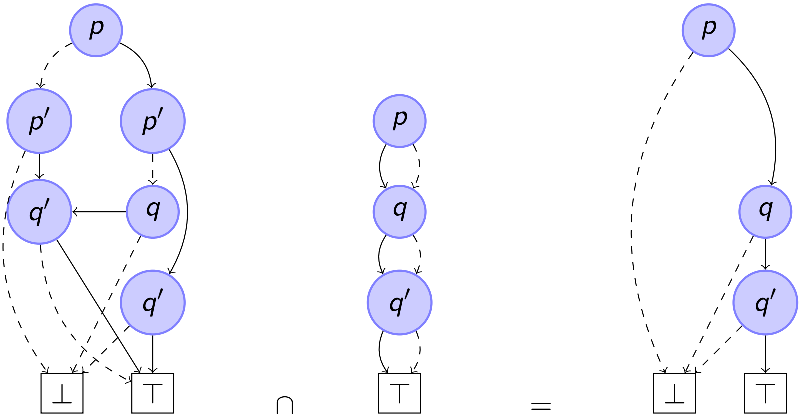


extended by: $enc\mathcal{P}(AP)$

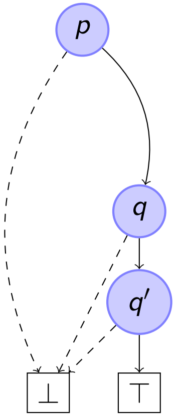


$EX(\neg p)$

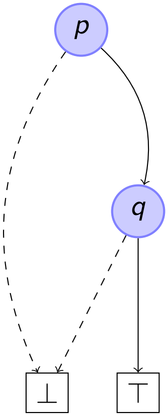
- Let's compute the intersection:



- Let's compute the reduction to AP:



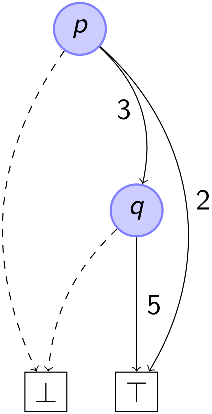
reduced into

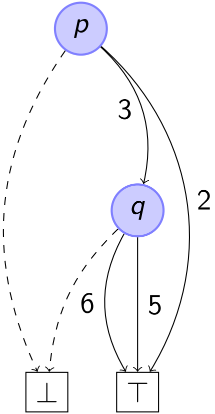


which means that $\{p, q\}$ is the only state s_1 satisfying $EX(\neg p)$.

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:





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

- 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