Set Family Decision Diagrams

Dimitri Racordon Didier Buchs

Centre Universitaire d'Informatique, Université de Genève

December 6, 2018

How to compute efficiently on sets ?

- represent sets in a compact way
- compute on a whole set instead on a single element
  - aka SIMD or *graphic card computing*
- respect union : set homomorphism

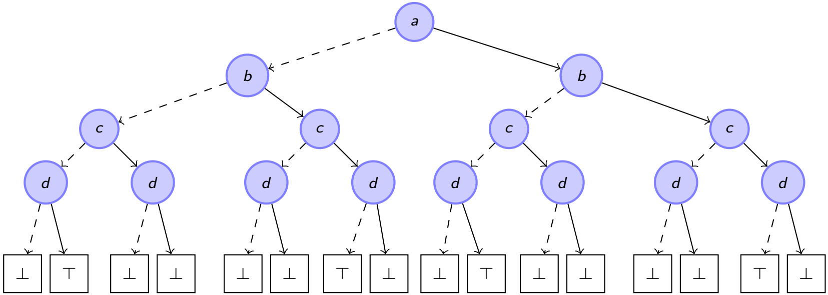
various approaches based on decision diagrams.

A SFDD is a directed acyclic graph where

- each node represent a term
- each node has two children, indicating whether or not the term is contained
- each path from the root to an accepting terminal represents a set of terms
- terms are totally ordered

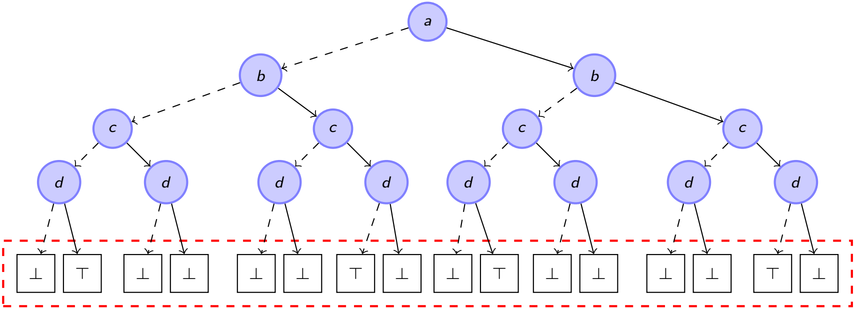
Encodes with the order $a < b < c < d$ the sets:

- $\{a, b, c\}$
- $\{a, d\}$
- $\{b, c\}$
- $\{d\}$



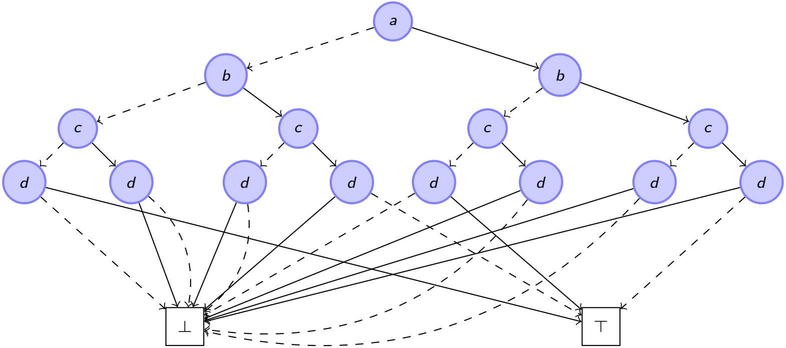
Encodes with the order $a < b < c < d$ the sets:

- $\{a, b, c\}$
- $\{a, d\}$
- $\{b, c\}$
- $\{d\}$



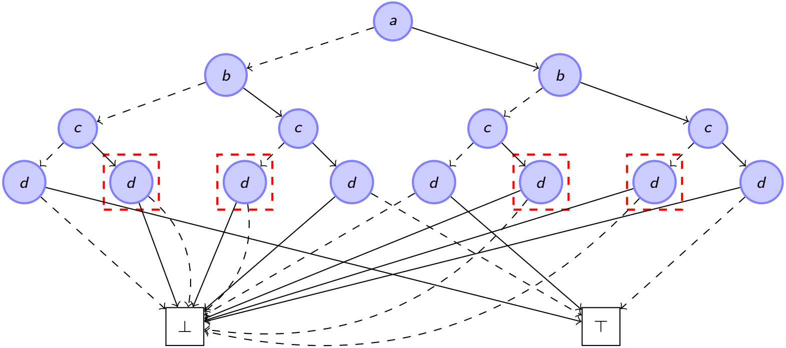
Encodes with the order $a < b < c < d$ the sets:

- $\{a, b, c\}$
- $\{a, d\}$
- $\{b, c\}$
- $\{d\}$



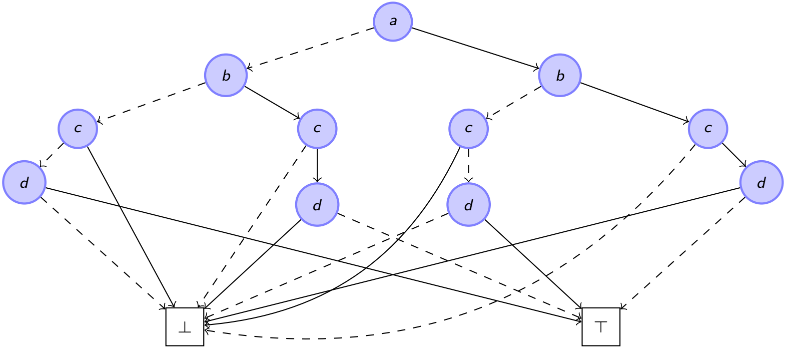
Encodes with the order $a < b < c < d$ the sets:

- $\{a, b, c\}$
- $\{a, d\}$
- $\{b, c\}$
- $\{d\}$



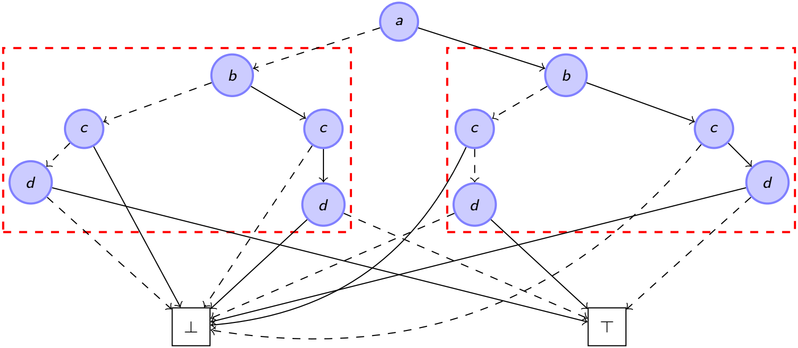
Encodes with the order $a < b < c < d$ the sets:

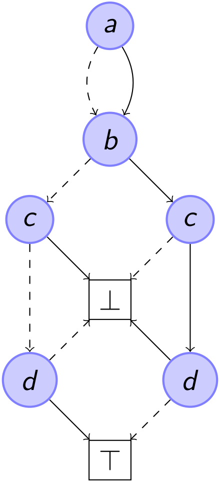
- $\{a, b, c\}$
- $\{a, d\}$
- $\{b, c\}$
- $\{d\}$



Encodes with the order $a < b < c < d$ the sets:

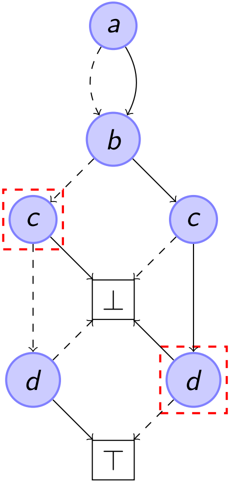
- $\{a, b, c\}$
- $\{a, d\}$
- $\{b, c\}$
- $\{d\}$





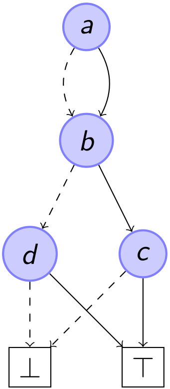
Encodes with the order $a < b < c < d$ the sets:

- $\{a, b, c\}$
- $\{a, d\}$
- $\{b, c\}$
- $\{d\}$



Encodes with the order $a < b < c < d$ the sets:

- $\{a, b, c\}$
- $\{a, d\}$
- $\{b, c\}$
- $\{d\}$



Encodes with the order $a < b < c < d$ the sets:

- $\{a, b, c\}$
- $\{a, d\}$
- $\{b, c\}$
- $\{d\}$

A set S over terms $T = \{t_1, t_2, ..., t_m\}$ is represented as a function f from T to $\mathbb{B}$, such as:

$$\begin{array}{ll} \forall s \in S, & f_S(s) = t \\ \forall s \in T - S, & f_S(s) = f \end{array}$$

A familly of sets $F = \{S_1, S_2, ..., S_n\}$, is defined as the following boolean function of arity m: $F : \mathbb{B} \times \mathbb{B} \times ... \times \mathbb{B} \rightarrow \mathbb{B}$ such as

$$\begin{array}{ll} \forall i \in 1...n, F(f_{S_i}(t_1), f_{S_i}(t_2), ..., f_{S_i}(t_m)) & = t \\ otherwise & = f \end{array}$$

This shows the correspondance between SFDD and BDD if we provide a total order over elements of T.

Although they are structurally similar, they benefit from different operations.

Moreover SFDD can be extended to other decision diagrams such as MFDD (encoding set of <KEY,VALUE>) and ΣDD (encoding set of Σ Terms) seamlessly.

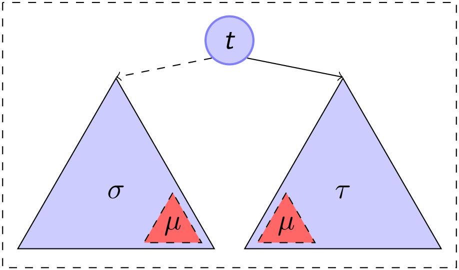
Definition (Formal definition)

Let T be a set of terms. The set of SFDDs $\mathbb{S}$ is inductively defined by:

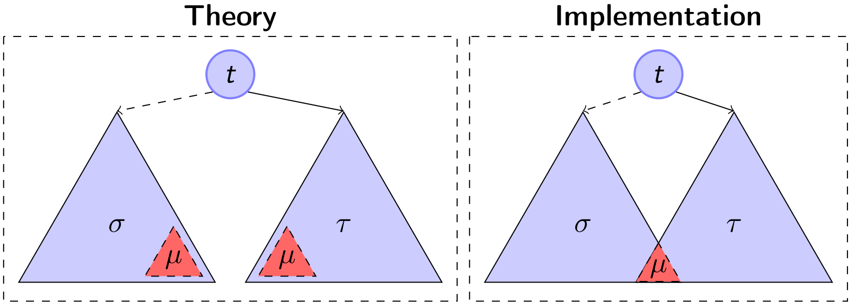
- $\bot \in \mathbb{S}$ is the rejecting terminal
- $\top \in \mathbb{S}$ is the accepting terminal
- $\langle t, \tau, \sigma \rangle \in \mathbb{S}$ if and only if $t \in T \wedge \tau, \sigma \in \mathbb{S}$

Theory VS Implementation

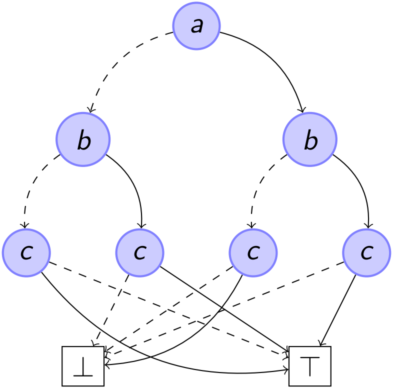
Theory



Theory VS Implementation

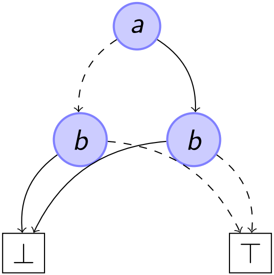


$$S = \{\emptyset, \{c\}, \{c,b\}, \{c,b,a\}\}$$



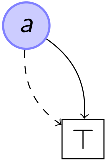
It is not optimal (neither unique, in fact depends on the constraint) as there is no common part (except the terminals) and several representation for the same set.

$S = \{\emptyset, \{a\}\}$



Representation uniqueness ?

$S = \{\emptyset, \{a\}\}$



From the brute ordered shape, we can reduce slightly the unnecessary nodes:

- remove negative nodes, i.e nodes with accept branch pointing to $\perp$, they are not providing any information.
- share common sub trees (not expressed in this formal definition)

clean : $\mathbb{S} \rightarrow \mathbb{S}$ removes a negative node from all sets that contain it:

$$\mathit{clean}(\perp) = \perp$$
$$\mathit{clean}(\top) = \top$$
$$\mathit{clean}(\langle t, \tau, \sigma \rangle) = \begin{cases} \mathit{clean}(\sigma) & \text{if } \tau = \perp \\ \langle t, \mathit{clean}(\tau), \mathit{clean}(\sigma) \rangle & \text{if } \mathit{otherwise} \end{cases}$$

NB: *clean* is an homomorphism.

Let $S \in \mathbb{S}$ be the SFDD $\langle t, \tau, \sigma \rangle$, we call τ its take node and σ its skip node.

S is canonical if for all its nodes, the skip node and take node represent greater terms or terminals, and no take node is the rejecting terminal. (sharing ?)

Definition (Canonical form)

Let T be a set of terms, and $< \in T \times T$ a total ordering on T. A SFDD $S \in \mathbb{S}$ is canonical if and only if

- S is the rejecting terminal $\bot$
- S is the accepting terminal $\top$
- $S = \langle t, \tau, \sigma \rangle$ where
  - $\tau = \langle t_\tau, \tau_\tau, \sigma_\tau \rangle \implies t < t_\tau$ and $\tau \neq \bot$
  - $\sigma = \langle t_\sigma, \tau_\sigma, \sigma_\sigma \rangle \implies t < t_\sigma$
  - τ and σ are canonical

From the brute ordered shape, we can reduce by the *clean* operation. Shared trees are themselves describbed by the fact that equivalent subtrees are collapsed by an equivalence relation. $\equiv \subseteq \mathbb{S} \times \mathbb{S}$ identify similar sets:

$$\begin{aligned}
\bot &\equiv \bot \\
\top &\equiv \top \\
\langle t, \tau, \sigma \rangle &\equiv \langle t, \tau', \sigma' \rangle && \text{if } \tau \equiv \tau' \wedge \sigma \equiv \sigma'
\end{aligned}$$

The structure which is implemented is then $\mathbb{S} = \mathit{clean}(\mathbb{S}_{\mathit{brute}}) / \equiv$. Implementations share same subtrees and memorization can be used due to the functional nature of operations (no side effects).

We give some basic examples of SFDD for a given set of sets from S and a total order $a < b < c$:

$$S = \{a, b, c\}$$

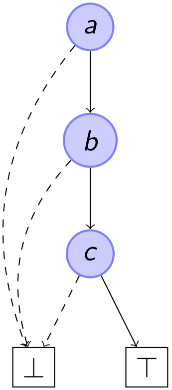
$$\wp(S) = \{\{a, b, c\}, \{a, b\}, \{b, c\}, \{a, c\}, \{a\}, \{b\}, \{c\}, \varnothing\}$$

$$\wp(S) - S = \{\{a, b\}, \{b, c\}, \{a, c\}, \{a\}, \{b\}, \{c\}, \varnothing\}$$

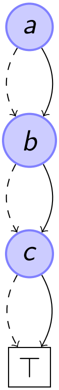
$$\wp(S) - \varnothing = \{\{a, b, c\}, \{a, b\}, \{b, c\}, \{a, c\}, \{a\}, \{b\}, \{c\}\}$$

Set Family Decision Diagrams
Example

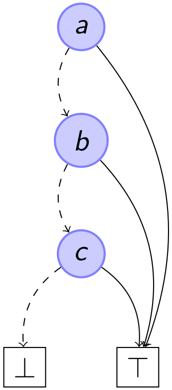
$enc(\{\{a, b, c\}\})$



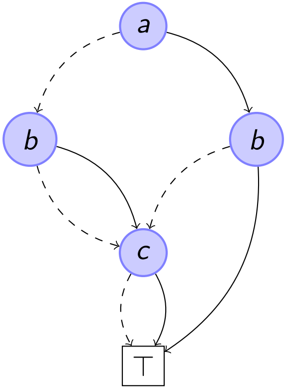
$enc(\wp(\{a, b, c\}))$



$enc(\{\{a\},\{b\},\{c\}\})$

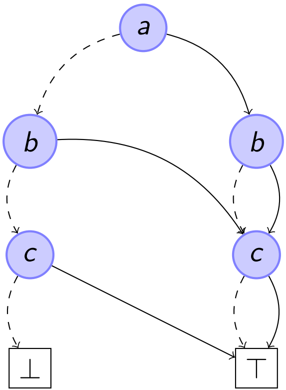


$enc(\wp(\{a, b, c\}) - \{a, b, c\})$



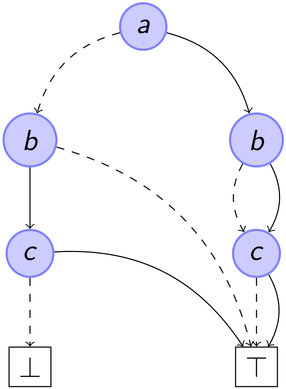
It is not a good case of encoding.

$enc(\wp(\{a, b, c\}) - \emptyset)$



It is one of the bad case we can expect, comparable to the previous one. But we can do worse.

$enc(\{\{a, b, c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a\}, \emptyset\})$



It is one of the worst case we can expect if we remove also the non singleton sets, comparable to the previous one, we need $2^{|S|} - k$ nodes to encode $\wp(S) - \emptyset$.

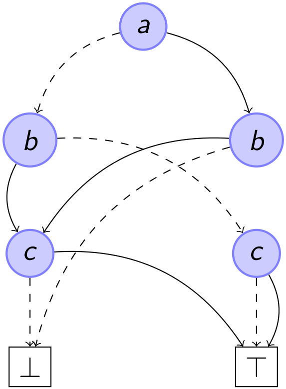
$$S_{i+1} = \{\emptyset\} \cup (S_i \oplus \{e_{i+1}\}), 0 \leq i \leq n-1$$

$$S_0 = \{\emptyset\}$$

$$S1 = \{\emptyset, \{a\}\}$$

$$S2 = \{\emptyset, \{b\}, \{a, b\}\}$$

$$S3 = \{\emptyset, \{c\}, \{c, b\}, \{c, b, a\}\}$$



It is one of the worst case we can expect if we remove also the non singleton sets, comparable to the previous one, we need $2^{|S|} - k$ nodes to encode $\wp(S) - \emptyset$.

The union of two SFDDs is given by:

$$
\begin{aligned}
A \cup B &= B \cup A \\
A \cup A &= A \\
\bot \cup A &= A \\
\top \cup \langle t, \tau, \sigma \rangle &= \langle t, \top \cup \tau, \top \cup \sigma \rangle \\
\langle t, \tau, \sigma \rangle \cup \langle t', \tau', \sigma' \rangle &= \begin{cases} \langle t, \tau, \sigma \cup \langle t', \tau', \sigma' \rangle \rangle & \text{if } t < t' \\ \langle t, \tau \cup \tau', \sigma \cup \sigma' \rangle & \text{if } t = t' \\ \langle t', \tau', \sigma' \cup \langle t, \tau, \sigma \rangle \rangle & \text{if } t > t' \end{cases}
\end{aligned}
$$

The intersection of two SFDDs is given by:

$$
\begin{aligned}
A \cap B &= B \cap A \\
A \cap A &= A \\
\bot \cap A &= \bot \\
\top \cap \langle t, \tau, \sigma \rangle &= \top \cap \sigma \\
\langle t, \tau, \sigma \rangle \cap \langle t', \tau', \sigma' \rangle &= \begin{cases} \sigma \cap \langle t', \tau', \sigma' \rangle & \text{if } t < t' \\ \langle t, \tau \cap \tau', \sigma \cap \sigma' \rangle & \text{if } t = t' \\ \langle t, \tau, \sigma \rangle \cap \sigma' & \text{if } t > t' \end{cases}
\end{aligned}
$$

The encoding of a set into a SFDD is given by:

$$
\begin{aligned}
\mathsf{enc}(\varnothing) &= \bot \\
\mathsf{enc}(\{\varnothing\}) &= \top \\
\mathsf{enc}(S \cup \{s\}) &= \mathsf{enc}(S) \cup \mathsf{enc}(\{s\}) \\
t < \min(s) \implies \mathsf{enc}(\{s \cup \{t\}\}) &= \langle t, \mathsf{enc}(\{s\}), \bot \rangle
\end{aligned}
$$

The decoding of one SFDD is given by:

$$\begin{aligned}
\mathrm{dec}(\bot) &= \varnothing \\
\mathrm{dec}(\top) &= \{\varnothing\} \\
\mathrm{dec}(\langle t, \tau, \sigma \rangle) &= (\mathrm{dec}(\tau) \oplus t) \cup \mathrm{dec}(\sigma)
\end{aligned}$$

Where $\oplus$ is defined as follows:

$$\bigcup_{s \in S} \{s\} \oplus t = \bigcup_{s \in S} \{s \cup \{t\}\}$$

The decoding/encoding of one set is the identity (and the reverse):

$$\forall S \subseteq \mathcal{P}(T), \mathrm{dec}(\mathrm{enc}(S)) = S$$
$$\forall S \in \mathbb{S}, \mathrm{enc}(\mathrm{dec}(S)) = S$$

We write as index the reference set T for the encoding : enc_T

- Extending the reference set from T to T' ($T \subseteq T'$) does not imply changing the representation:
  $\forall S \subseteq \mathcal{P}(T) \Rightarrow \mathrm{enc}_T(S) = \mathrm{enc}_{T'}(S)$
- Under some constraint we can reduce the reference set $T' \subseteq T$, with or without change,$\forall S \subseteq \mathcal{P}(T)$:
  - case 1: $S \cap (T - T') = \emptyset \Rightarrow$
    $\mathrm{enc}_T(S) = \mathrm{enc}_{T'}(S)$
  - case 2: $S \cap (T - T') \neq \emptyset \Rightarrow$
    $\mathrm{enc}_T(S \cap T') = \mathrm{enc}_{T'}(S \cap T') = \mathrm{enc}_T(S) \ominus (T - T')$

Where $\cup, -$ and $\cap$ are defined as extension of set operation on family of sets, $\ominus$ is defined later on SFDD.

Homomorphisms are operations that preserve union:

$$\phi(S \cup S') = \phi(S) \cup \phi(S')$$

They also support operations that are themselves homomorphisms:

$$\forall S, (\phi_1 + \phi_2)(S) = \phi_1(S) \cup \phi_2(S)$$
$$\forall S, (\phi_1 \times \phi_2)(S) = \phi_1(S) \cap \phi_2(S)$$
$$\forall S, (\phi_1 \circ \phi_2)(S) = \phi_1(\phi_2(S))$$

$\oplus : \mathbb{S}, T \to \mathbb{S}$ inserts a term $t \in T$ into all sets of a SFDD:

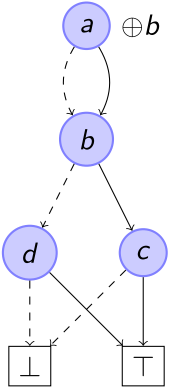
$$\bot \oplus a = \bot$$

$$\top \oplus a = \langle a, \top, \bot \rangle$$

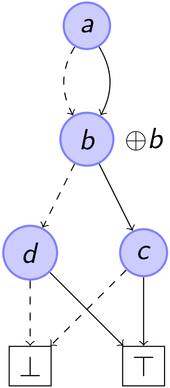
$$\langle t, \tau, \sigma \rangle \oplus a = \begin{cases} \langle t, \tau \oplus a, \sigma \oplus a \rangle & \text{if } t < a \\ \langle t, \tau \cup \sigma, \bot \rangle & \text{if } t = a \\ \langle a, \langle t, \tau, \sigma \rangle, \bot \rangle & \text{if } t > a \end{cases}$$

NB: $\oplus$ is an homomorphism.

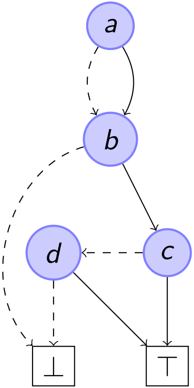
Example: $\text{enc}(\{\{a, b, c\}, \{a, d\}, \{b, c\}, \{d\}\}) \oplus b$



Example: $\mathrm{enc}(\{\{a, b, c\}, \{a, d\}, \{b, c\}, \{d\}\}) \oplus b$



Example: $\mathrm{enc}(\{\{a, b, c\}, \{a, d\}, \{b, c\}, \{d\}\}) \oplus b$



Encodes the sets:

- $\{a, b, c\}$
- $\{a, b, d\}$
- $\{b, c\}$
- $\{b, d\}$

$\ominus : \mathbb{S}, T \to \mathbb{S}$ removes a term $t \in T$ from all sets that contain it:

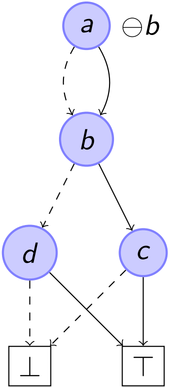
$$\bot \ominus a = \bot$$

$$\top \ominus a = \top$$

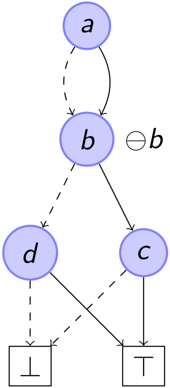
$$\langle t, \tau, \sigma \rangle \ominus a = \begin{cases} \langle t, \tau \ominus a, \sigma \ominus a \rangle & \text{if } t < a \\ \sigma \cup \tau & \text{if } t = a \\ \langle t, \tau, \sigma \rangle & \text{if } t > a \end{cases}$$

NB: $\ominus$ is an homomorphism.

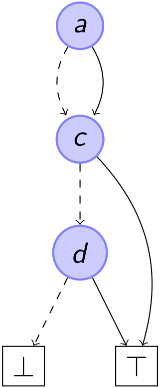
Example: $\mathrm{enc}(\{\{a, b, c\}, \{a, d\}, \{b, c\}, \{d\}\}) \ominus b$



Example: $\mathrm{enc}(\{\{a,b,c\},\{a,d\},\{b,c\},\{d\}\}) \ominus b$



Example: $\text{enc}(\{\{a, b, c\}, \{a, d\}, \{b, c\}, \{d\}\}) \ominus b$



Encodes the sets:

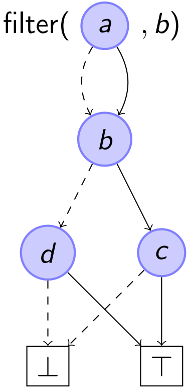
- $\{a, c\}$
- $\{a, d\}$
- $\{c\}$
- $\{d\}$

filter : $\mathbb{S}, T \to \mathbb{S}$ filters out the sets that don't contain a term $t \in T$:

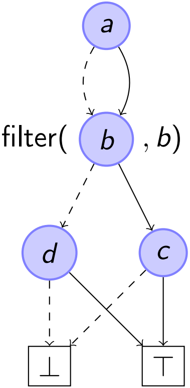
$$\begin{aligned}
\text{filter}(\bot, a) &= \bot \\
\text{filter}(\top, a) &= \bot \\
\text{filter}(\langle t, \tau, \sigma\rangle, a) &= \begin{cases} \langle t, \text{filter}(\tau, a), \text{filter}(\sigma, a)\rangle & \text{if } t < a \\ \langle t, \tau, \bot\rangle & \text{if } t = a \\ \bot & \text{if } t > a \end{cases}
\end{aligned}$$

NB1: filter is an homomorphism.

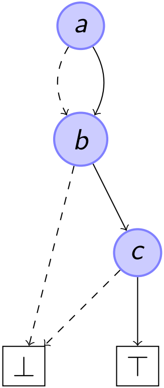
Example: $\mathrm{filter}(\mathrm{enc}(\{\{a, b, c\}, \{a, d\}, \{b, c\}, \{d\}\}), b)$



Example: $\text{filter}(\text{enc}(\{\{a, b, c\}, \{a, d\}, \{b, c\}, \{d\}\}), b)$



Example: $\mathrm{filter}(\mathrm{enc}(\{\{a, b, c\}, \{a, d\}, \{b, c\}, \{d\}\}), b)$



Encodes the sets:

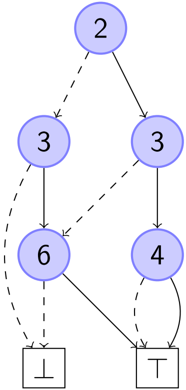
- $\{a, b, c\}$
- $\{b, c\}$

An inductive homomorphism is a tuple $\phi = \langle S, i \rangle$ where:

- $S \in \mathbb{S}$
- $i(A) = \langle \phi_\tau, \phi_\sigma \rangle$ where ϕ_τ, ϕ_σ are homomorphisms and $A \in \mathbb{S} \setminus \{\perp, \top\}$

Let $\phi = \langle S, i \rangle$, its application on $A \in \mathbb{S}$ is given by:

$$\phi(A) = \begin{cases} \perp & \text{if } A = \perp \\ S & \text{if } A = \top \\ \langle t, \phi_\tau(\tau), \phi_\sigma(\sigma) \rangle & \text{if } A = \langle t, \tau, \sigma \rangle, i(A) = \langle \phi_\tau, \phi_\sigma \rangle \end{cases}$$

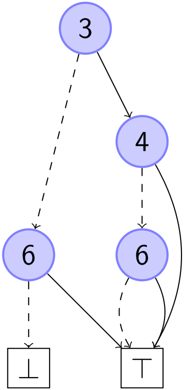


Example: removing values smaller than of 4

$$\phi = \langle \top, i \rangle$$

$$i(\langle t, \tau, \sigma \rangle) = \begin{cases} \langle h[\bot], \phi \circ (h[\tau] + \mathrm{id}) \rangle & \text{if } t < 4 \\ \langle \mathrm{id}, \mathrm{id} \rangle & \text{otherwise} \end{cases}$$

where $\forall S, \mathrm{id}(S) = S$ and $\forall S, h[K](S) = K$.

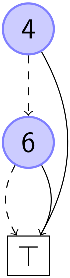


Example: removing values smaller than of 4

$$\phi = \langle \top, i \rangle$$

$$i(\langle t, \tau, \sigma \rangle) = \begin{cases} \langle h[\perp], \phi \circ (h[\tau] + \mathrm{id}) \rangle & \text{if } t < 4 \\ \langle \mathrm{id}, \mathrm{id} \rangle & \text{otherwise} \end{cases}$$

where $\forall S, \mathrm{id}(S) = S$ and $\forall S, h[K](S) = K$.



Example: removing values smaller than of 4

$$\phi = \langle \top, i \rangle$$

$$i(\langle t, \tau, \sigma \rangle) = \begin{cases} \langle h[\bot], \phi \circ (h[\tau] + \mathrm{id}) \rangle & \text{if } t < 4 \\ \langle \mathrm{id}, \mathrm{id} \rangle & \text{otherwise} \end{cases}$$

where $\forall S, \mathrm{id}(S) = S$ and $\forall S, h[K](S) = K$.

The count of members in a family is the operation size:

$$\mathrm{size}(S) = \begin{cases} 0 & \text{if } S = \bot \\ 1 & \text{if } S = \top \\ \mathrm{size}(\tau) + \mathrm{size}(\sigma) & \text{if } S = \langle t, \tau, \sigma \rangle \end{cases}$$

NB1: size is not an homomorphism.

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 ← {s_0} ; s ← {};
    repeat
        m ← choose(s_rem) ;
        s_rem ← s_rem/{m} ;
        foreach t ∈ T do
            if fireable(t,m) then
                mt ← t(m) ;
                if m ∉ s then s ← s ∪ {mt}; s_rem ← s_rem ∪ {mt};
    until s_rem = ∅;
    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 ;
    $s \leftarrow \{s_0\}$;
    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;

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:

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

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):

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

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

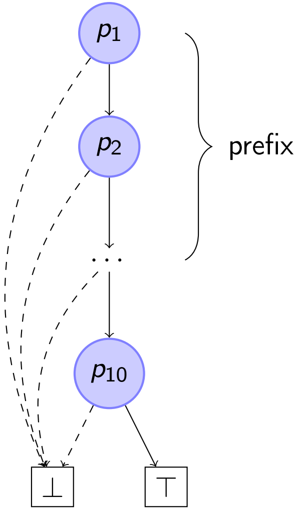
$$S_M = \bigcup_{p \in P, M(p)=1} \{p\}$$

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

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

Homomorphisms may involve uncessary operations on large prefixes:

$$\text{filter}(p_{10})(\text{enc}(\bigcup_{1 \leq i \leq 10} p_i))$$



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

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

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:

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

As in BDD 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

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

Definition

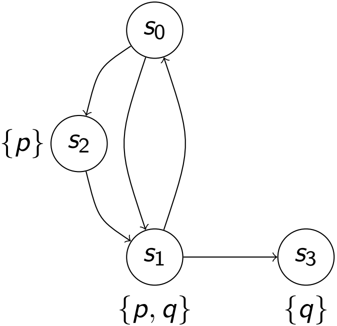
A Kripke structure of a set of atomic propositions AP is a tuple $K = \langle S, S_0, R, L \rangle$ where:

- $\mathbf{S}$ 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

Definition

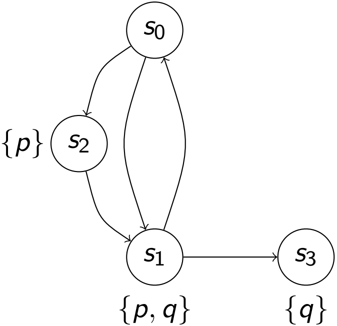
A Kripke structure of a set of atomic propositions AP is a tuple $K = \langle S, S_0, R, L \rangle$ where:

- $\mathbf{S}$ 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
- $\mathbf{L} : \mathbf{S} \to \mathcal{P}(\mathbf{AP})$ labels each state with a set of atomic propositions that hold on that state



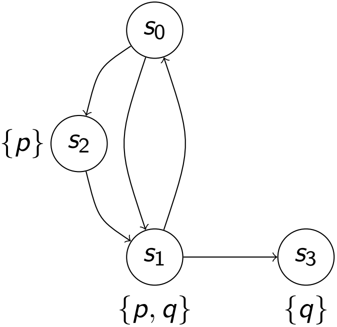
$K = \langle S, S_0, R, L \rangle$ where:

Kripke Structure
example



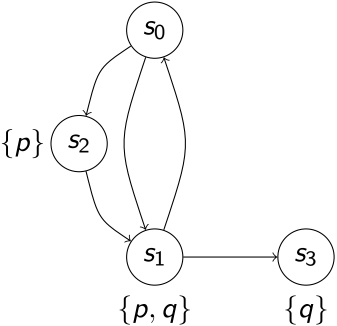
$K = \langle S, S_0, R, L \rangle$ where:

- $S = \{s_0, s_1, s_2, s_3\}$



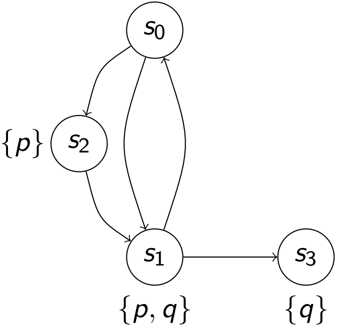
$K = \langle S, S_0, R, L \rangle$ where:

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



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

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



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

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

From Kripke Structure to SFDD

- Given AP, we create a sibling set AP' different from AP: $AP \cap AP' = \emptyset$ and a bijective function $sib : AP \rightarrow AP'$
- 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 $s_b \in AP$:
$s_a < s_b \Leftrightarrow s_a <' s_b \Leftrightarrow sib(s_a) <' sib(s_b)$

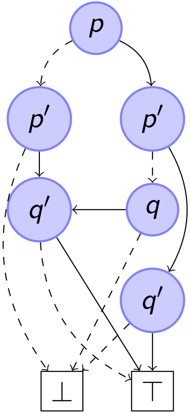
Example

$AP = \{p, q\}$, $AP' = \{p', q'\}$ 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 $sib : \mathcal{P}(AP) \rightarrow \mathcal{P}(AP')$ and $sib^{-1} : AP' \rightarrow AP$

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

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



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

Shannon decomposition on sets

Let's do Shannon decomposition

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

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

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

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

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

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

$EX(\phi)$

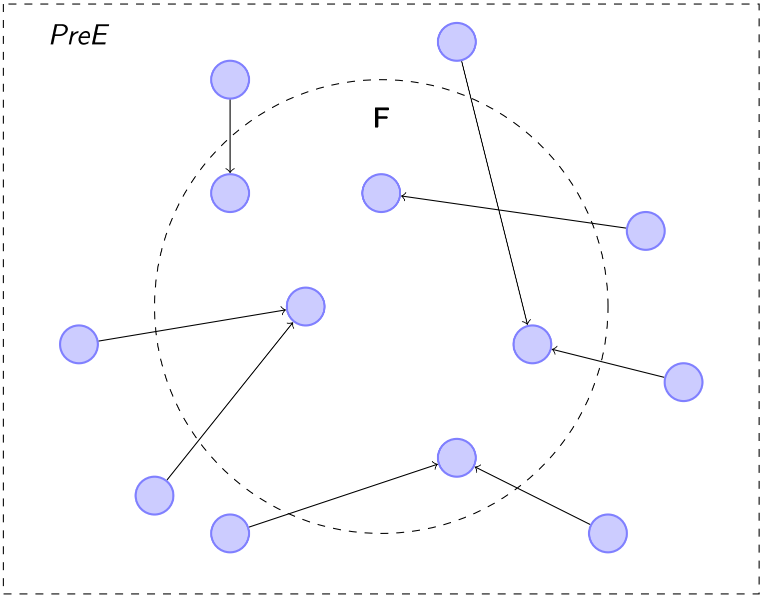
- Let F be the set of states ($\in$ *SFDD*) satisfying ϕ:
  - S := *PreE*(F)
  - Return S

Function PreE

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

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

$PreE(F)$



$E(\phi Until \theta)$

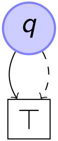
- Let F(resp. G) be the set of states ($\in SFDD$) satisfying ϕ (resp. θ):
  - S := G
  - N := enc($\emptyset$)
  - While(N $\neq$ S)
  - do
    - N := S;
    - S := $S \cup (F \cap PreE$(S))
  - done
  - Return S

- Let F be the set of states ($\in$ *SFDD*) satisfying ϕ:
    - S := F
    - N := enc($\emptyset$)
    - While(N $\neq$ S)
    - do
        - N := S;
        - S := S $\cap$ *preE*(S)
    - done
    - Return S

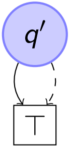
$EX(\neg p)$

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

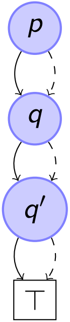
The states satisfying $\neg p$ are: s_0, s_3 which are the states where the $\{\emptyset, \{q\}\}$ atomic propositions are valid



transformed by: sib

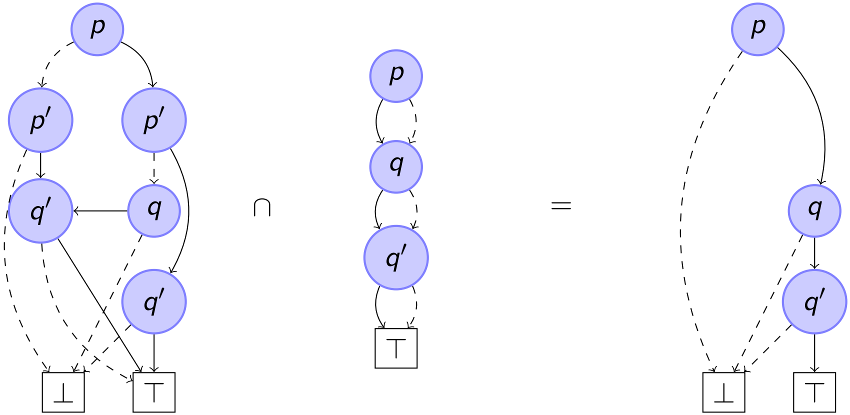


and extend by: $enc\mathcal{P}(AP)$



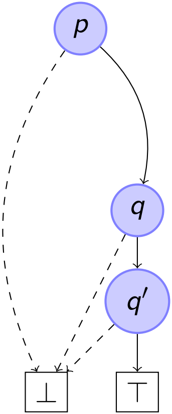
$EX(\neg p)$

- Let's compute the intersection:

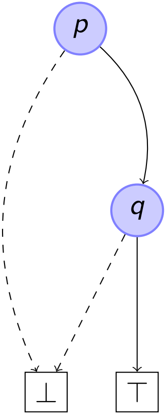


$EX(\neg p)$

- Let's compute the reduction to AP:



reduced into



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

- 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