On Definability for Model Counting

Jean-Marie Lagniez\textsuperscript{1}, Emmanuel Lonca\textsuperscript{1} and Pierre Marquis\textsuperscript{1,2}

\textsuperscript{1}CRIL, U. Artois & CNRS
\textsuperscript{2}Institut Universitaire de France

Meeting GT ALGA, GdR IM, Lille, October 15\textsuperscript{th}, 2018
Key idea: Leveraging the power of modern SAT solvers to tackle other intractable problems

Objective: Enlarging the sets of instances which can be solved in practice using "reasonable" resources

- Knowledge compilers
- MUS/MCS enumerators
- QBF solvers
- Model counters
- ...

beyondnp.org
Model Counting

\[ \Sigma \mapsto \| \Sigma \| = ? \]
Model Counting

- $\Sigma \leftrightarrow \|\Sigma\| = ?$

- $\Sigma = (x \lor y) \land (\neg y \lor z)$
\[
\Sigma \mapsto \|\Sigma\| = ?
\]
\[
\Sigma = (x \lor y) \land (\neg y \lor z)
\]
\[
The \text{models of } \Sigma \text{ over } \{x, y, z\} \text{ are:}
\]

- 011
- 100
- 101
- 111
Model Counting

\[
\Sigma \mapsto \|\Sigma\| = ?
\]

\[
\Sigma = (x \lor y) \land (\neg y \lor z)
\]

The models of $\Sigma$ over $\{x, y, z\}$ are:

- 011
- 100
- 101
- 111

\[
\|\Sigma\| = 4
\]
Counting the models of a propositional formula is a key task for a number of problems (especially in AI):

- probabilistic inference
- stochastic planning
- ...

However, counting is a computationally hard task: \#P-complete. Even for subsets of formulae for which counting is easy (e.g., monotone Krom formulae), the "power" of counting and its complexity are reflected by Toda's theorem: 

Seinosuke Toda (Gödel Prize 1998): \( \text{PH} \subseteq \text{P}^{\#P} \)
Counting the models of a propositional formula is a key task for a number of problems (especially in AI):

- probabilistic inference
- stochastic planning
- ...

However \#SAT is a computationally hard task: \#P-complete
Counting the models of a propositional formula is a key task for a number of problems (especially in AI):
- probabilistic inference
- stochastic planning
- ...

However \#\text{sat} is a computationally hard task: \#P-complete

Even for subsets of formulae for which \text{sat} is easy (e.g., monotone Krom formulae)!
Counting the models of a propositional formula is a key task for a number of problems (especially in AI):
- probabilistic inference
- stochastic planning
- ...

However \#\textsc{sat} is a computationally hard task: \#P-complete

Even for subsets of formulae for which \textsc{sat} is easy (e.g., monotone Krom formulae)!

The "power" of counting and its complexity are reflected by Toda’s theorem:

Seinosuke Toda (Gödel Prize 1998):

\[
\text{PH} \subseteq \text{P}^{\#\text{P}}
\]
Many model counters have been developed:

- Exact model counters:
  - search-based: Cachet, SharpSAT, DMC, etc.,
  - compilation-based: C2D, Dsharp, D4, etc.
- Approximate model counters (SampleCount, etc.)
Many model counters have been developed:
- Exact model counters:
  - search-based: Cachet, SharpSAT, DMC, etc.,
  - compilation-based: C2D, Dsharp, D4, etc.
- Approximate model counters (SampleCount, etc.)

In this talk: improving exact model counters by preprocessing the input

\[ \text{CNF} \rightarrow \text{CNF} \]
Objective: simplifying the input so that the task at hand can be achieved more efficiently from the input once preprocessed

Simplifying = ”reducing something”

Trade-off preprocessing cost / rest of the computation to be looked for

Using aggressive, computationally demanding preprocessing techniques can make sense when dealing with highly complex problems (like #SAT)

P-preprocessing vs. NP-preprocessing
Similarities: two off-line approaches for improving the model counting task
Knowledge Compilation vs. Preprocessing for \#SAT

- **Similarities:** two **off-line** approaches for improving the model counting task
- **Differences:**
  - computing a new representation in the same vs. a distinct language
  - "hard part" vs. "easy part"
Knowledge Compilation vs. Preprocessing for #SAT

- **Similarities**: two off-line approaches for improving the model counting task
- **Differences**:
  - computing a new representation in the same vs. a distinct language
  - "hard part" vs. "easy part"

- **knowledge compilation**

\[
\text{CNF } \Sigma \quad \xrightarrow{\text{compilation}} \quad \text{d-DNNF } \Psi \quad \xrightarrow{\text{model counting}} \quad ||\Sigma||
\]
Similarities: two off-line approaches for improving the model counting task

Differences:
- computing a new representation in the same vs. a distinct language
- "hard part" vs. "easy part"

**knowledge compilation**

\[
\text{CNF } \Sigma \xrightarrow{\text{compilation}} \text{d-DNNF } \Psi \xrightarrow{\text{model counting}} \| \Sigma \|
\]

**preprocessing**

\[
\text{CNF } \Sigma \xrightarrow{\text{preprocessing}} \text{CNF } \Phi \xrightarrow{\text{model counting}} \| \Sigma \|
\]
Knowledge Compilation vs. Preprocessing for \#SAT

- **Similarities:** two off-line approaches for improving the model counting task

- **Differences:**
  - computing a new representation in the same vs. a distinct language
  - "hard part" vs. "easy part"

- **knowledge compilation**
  \[
  \text{CNF } \Sigma \xrightarrow{\text{compilation}} \text{d-DNNF } \Psi \xrightarrow{\text{model counting}} ||\Sigma||
  \]

- **preprocessing**
  \[
  \text{CNF } \Sigma \xrightarrow{\text{preprocessing}} \text{CNF } \Phi \xrightarrow{\text{model counting}} ||\Sigma||
  \]

- The two approaches can be combined
Dozens of P-Preprocessings

- Vivification (VI) and a light form of it, called Occurrence Elimination (OE),
- Gate Detection and Replacement (GDR)
- Pure Literal Elimination (PLE)
- Variable Elimination (VE)
- Blocked Clause Elimination (BCE)
- Covered Clause Elimination (CCE)
- Failed Literal Elimination (FLE)
- Self-Subsuming Resolution (SSR)
- Hidden Literal Elimination (HLE)
- Subsumption Elimination (SE)
- Hidden Subsumption Elimination (HSE)
- Asymmetric Subsumption Elimination (ASE)
- Tautology Elimination (TE)
- Hidden Tautology Elimination (HTE)
- Asymmetric Tautology Elimination (ATE)
- ...

8/46 On Definability for Model Counting Meeting GT ALGA, GdR IM, Lille, October 15th, 2018
Use in State-of-the-Art SAT Solvers

- Glucose (exploits the SatELite preprocessor)
- Lingeling (has an internal preprocessor)
- Riss (use of the Coprocessor preprocessor)
- ...
CNF $\Sigma \mapsto$ CNF $p(\Sigma)$

- What are the connections between $\Sigma$ and $p(\Sigma)$?
- Removing clauses from $\Sigma$
- Removing literals in the clauses of $\Sigma$
- ...
A clause $\delta$ of a CNF $\Sigma$ is **redundant** if and only if $\Sigma \setminus \{\delta\} \models \delta$

A CNF $\Sigma$ is **irredundant** if and only if it does not contain any redundant clause

A subset $\Sigma'$ of a CNF $\Sigma$ is an **irredundant equivalent subset (IES)** of $\Sigma$ if and only if $\Sigma'$ is irredundant and $\Sigma' \equiv \Sigma$

Deciding whether a CNF $\Sigma$ is irredundant is NP-complete

Deciding whether a CNF $\Sigma'$ is an irredundant equivalent subset (IES) of a CNF $\Sigma$ is $\mathcal{D}^p$-complete

Given an integer $k$, deciding whether a CNF $\Sigma$ has an IES of size at most $k$ is $\Sigma^p_2$-complete

Given an integer $k$, deciding whether there exists a CNF formula $\Sigma'$ with at most $k$ literals (or with at most $k$ clauses) equivalent to a given CNF $\Sigma$ is $\Sigma^p_2$-complete
Preserving What?

- Logical equivalence
- Queries over the input alphabet
- **Number of models**
- Satisfiability
- ...
Several measures for the reduction achieved can be considered:

- The number of variables in the input CNF $\Sigma$
- The size of $\Sigma$ (the number of literals or the number of clauses in it)
- The value of some structural parameters for $\Sigma$
- ...
A clause $\delta_1$ subsumes a clause $\delta_2$
if every literal of $\delta_1$ is a literal of $\delta_2$

\[ SE : (x_1 \lor x_2) \land (x_1 \lor x_2 \lor \overline{x_3}) \mapsto x_1 \lor x_2 \]

- P-preprocessing
- Preserves logical equivalence
- Hence preserves the number of models of the input (over the original alphabet), its queries and its satisfiability

- $\#\text{var}(\Sigma) \geq \#\text{var}(SE(\Sigma))$
- $\#\text{lit}(\Sigma) \geq \#\text{lit}(SE(\Sigma))$
A gate of $\Sigma$ is a circuit $\ell \iff \beta$ such that $\Sigma \models \ell \iff \beta$
$\Sigma$ and $\Sigma[\ell \leftarrow \beta]$ have the same number of models (but are not logically equivalent in general)

$$\Sigma = \overline{x} \lor u \lor v$$
$$\overline{x} \lor \overline{y} \lor u$$
$$\overline{x} \lor \overline{z} \lor u$$
$$x \lor \overline{u}$$
$$y \lor z \lor \overline{u}$$
A **gate** of $\Sigma$ is a circuit $\ell \Leftrightarrow \beta$ such that $\Sigma \models \ell \Leftrightarrow \beta$

$\Sigma$ and $\Sigma[\ell \leftarrow \beta]$ have the same number of models (but are not logically equivalent in general)

\[
\Sigma = \overline{x} \lor u \lor v \\
\overline{x} \lor \overline{y} \lor u \\
\overline{x} \lor \overline{z} \lor u \\
x \lor \overline{u} \\
y \lor z \lor \overline{u} \\
u \Leftrightarrow (x \land (y \lor z))
\]
A gate of $\Sigma$ is a circuit $\ell \leftrightarrow \beta$ such that $\Sigma \models \ell \leftrightarrow \beta$

$\Sigma$ and $\Sigma[\ell \leftarrow \beta]$ have the same number of models (but are not logically equivalent in general)

$$\Sigma = \overline{x} \lor u \lor v$$

$$\overline{x} \lor \overline{y} \lor u$$

$$\overline{x} \lor \overline{z} \lor u$$

$$x \lor \overline{u}$$

$$y \lor z \lor \overline{u}$$

$$u \leftrightarrow (x \land (y \lor z))$$

$$\Sigma \equiv (\overline{x} \lor u \lor v) \land (u \leftrightarrow (x \land (y \lor z)))$$

detection
A gate of $\Sigma$ is a circuit $\ell \iff \beta$ such that $\Sigma \models \ell \iff \beta$.

$\Sigma$ and $\Sigma[\ell \leftarrow \beta]$ have the same number of models (but are not logically equivalent in general)

\[
\Sigma = \begin{align*}
\overline{x} \lor u \lor v \\
\overline{x} \lor \overline{y} \lor u \\
x \lor \overline{u} \\
y \lor z \lor \overline{u}
\end{align*}
\]

$\Sigma \equiv \begin{align*}
(\overline{x} \lor u \lor v) \land (u \iff (x \land (y \lor z))) \\
(\overline{x} \lor (x \land (y \lor z)) \lor v) \land (u \iff (x \land (y \lor z)))
\end{align*}$

- detection
- replacement
A gate of $\Sigma$ is a circuit $\ell \leftrightarrow \beta$ such that $\Sigma \models \ell \leftrightarrow \beta$ and $\Sigma[\ell \leftarrow \beta]$ have the same number of models (but are not logically equivalent in general)

$$
\Sigma =
\overline{x} \lor u \lor v \\
\overline{x} \lor \overline{y} \lor u \\
x \lor \overline{u} \\
y \lor z \lor \overline{u}
$$

$$
\Sigma \equiv
(\overline{x} \lor u \lor v) \land (u \leftrightarrow (x \land (y \lor z)))
$$

detection

$$
(\overline{x} \lor (x \land (y \lor z)) \lor v) \land (u \leftrightarrow (x \land (y \lor z)))
$$

replacement

$$
(\overline{x} \lor y \lor z \lor v) \land (u \leftrightarrow (x \land (y \lor z)))
$$

normalization
A gate of $\Sigma$ is a circuit $\ell \Leftrightarrow \beta$ such that $\Sigma \models \ell \Leftrightarrow \beta$

$\Sigma$ and $\Sigma[\ell \leftarrow \beta]$ have the same number of models (but are not logically equivalent in general)

$$\Sigma = \begin{array}{c}
\overline{x} \lor u \lor v \\
\overline{x} \lor \overline{y} \lor u \\
x \lor \overline{\overline{u}} \\
y \lor z \lor \overline{\overline{u}}
\end{array}$$

$u \leftrightarrow (x \land (y \lor z))$

$$\Sigma \equiv \begin{array}{l}
(\overline{x} \lor u \lor v) \land (u \leftrightarrow (x \land (y \lor z))) \quad \text{detection} \\
(\overline{x} \lor (x \land (y \lor z)) \lor v) \land (u \leftrightarrow (x \land (y \lor z))) \quad \text{replacement} \\
(\overline{x} \lor y \lor z \lor v) \land (u \leftrightarrow (x \land (y \lor z))) \quad \text{normalization}
\end{array}$$

$$\|\Sigma\| = \|\Sigma[u \leftarrow (x \land (y \lor z))]\| = \|\overline{x} \lor y \lor z \lor v\| = 15$$
Gate detection and replacement proves to be a valuable preprocessing.

Specific gates are typically sought for (literal equivalence, AND/OR gates, XOR gates) for complexity reasons.

The replacement $\Sigma[\ell \leftarrow \beta]$ requires to turn the resulting formula into CNF.

It is implemented only if it does not lead to increase the size of the input (a "small" increase can also be accepted).

BCP (instead of a "full" SAT solver) is often used for efficiency reasons (P-preprocessing).
Literal Equivalence (LE)

- **Literal equivalence** aims to detect equivalences between literals using BCP.
- P-preprocessing
- For each literal $\ell$, all the literals $\ell'$ which can be found equivalent to $\ell$ using BCP are replaced by $\ell$ in $\Sigma$.
- Taking advantage of BCP makes it more efficient than a "syntactic detection" (if two binary clauses stating an equivalence between two literals $\ell$ and $\ell'$ occur in $\Sigma$, then those literals are found equivalent using BCP, but the converse does not hold).
Literal Equivalence (LE)

Algorithm 1: LE

input: a CNF formula $\Sigma$

output: a CNF formula $\Phi$ such that $||\Phi|| = ||\Sigma||$

1. $\Phi \leftarrow \Sigma$; Unmark all variables of $\Phi$;

2. while $\exists \ell \in \text{Lit}(\Phi)$ s.t. $\text{var}(\ell)$ is not marked do

   // detection
   3. mark $\text{var}(\ell)$;
   4. $\mathcal{P}_\ell \leftarrow \text{BCP}(\Phi \cup \{\ell\})$;
   5. $\mathcal{N}_\ell \leftarrow \text{BCP}(\Phi \cup \{\neg \ell\})$;
   6. $\Gamma \leftarrow \{\ell \leftrightarrow \ell' | \ell' \neq \ell \text{ and } \ell' \in \mathcal{P}_\ell \text{ and } \neg \ell' \in \mathcal{N}_\ell\}$;

   // replacement
   7. foreach $\ell \leftrightarrow \ell' \in \Gamma$ do
      8. replace $\ell$ by $\ell'$ in $\Phi$;

9. return $\Phi$
Literal Equivalence (LE): Example

\[ \Sigma = \]
\[ a \lor b \lor c \lor \neg d \lor \neg a \lor \neg b \lor \neg c \lor d \]
\[ a \lor b \lor \neg c \lor \neg a \lor \neg b \lor c \]
\[ \neg a \lor b \lor \neg a \lor \neg b \lor c \]
\[ \neg e \lor \neg f \lor h \lor e \lor f \lor g \]
\[ e \lor \neg g \lor \neg e \lor \neg h \]

Assume that the variables of \( \Sigma \) are considered in the following ordering: \( a < b < c < d < e < f < g < h \)

The equivalences \((a \iff b) \land (b \iff c) \land (c \iff d) \land (e \iff \neg f)\) are detected

\[ \text{LE}(\Sigma) = \]
\[ \neg f \lor \neg g \lor f \lor \neg h \]
Properties of LE

- Preserves the number of models (but not logical equivalence)
- \(# \text{var}(\Sigma) \geq \# \text{var}(\text{LE}(\Sigma))\)
- \(# \text{lit}(\Sigma) \geq \# \text{lit}(\text{LE}(\Sigma))\)
LE: Reduction of the Number of Variables

**Figure** — Comparing \( \#\text{var}(\Sigma) \) with \( \#\text{var}(\text{LE}(\Sigma)) \).
LE: Reduction of the Size

**Figure** — Comparing \( \#\text{lit}(\Sigma) \) with \( \#\text{lit}(\text{LE}(\Sigma)) \).
The backbone of a CNF formula $\Sigma$ is the set of all literals which are implied by $\Sigma$ when $\Sigma$ is satisfiable, and is the empty set otherwise.

The purpose of the BI preprocessing is to make the backbone $B$ of the input CNF formula $\Sigma$ explicit, to conjoin it to $\Sigma$, and to use BCP (Boolean Constraint Propagation) on the resulting set of clauses.

NP-preprocessing
Algorithm 2: BI Backbone Identification

**input**: a CNF formula $\Sigma$

**output**: the CNF $\text{BCP}(\Sigma \cup B)$, where $B$ is the backbone of $\Sigma$

1. $B \leftarrow \emptyset$
2. $I \leftarrow \text{solve}(\Sigma)$
3. **while** $\exists \ell \in I$ s.t. $\ell \notin B$ **do**
4. 4.1 $I' \leftarrow \text{solve}(\Sigma \cup \{\neg \ell\})$
5. 4.2 **if** $I' = \emptyset$ **then** $B \leftarrow B \cup \{\ell\}$ **else** $I \leftarrow I \cap I'$
6. return $\text{BCP}(\Sigma \cup B)$
Backbone Identification (BI): Example

\[ \Sigma = \]
\[ a \lor b \]
\[ \neg a \lor b \]
\[ \neg b \lor c \]
\[ c \lor d \]
\[ \neg c \lor e \lor f \]
\[ f \lor \neg g \]

The backbone of \( \Sigma \) is equal to \( B = \{b, c\} \)

\[ BI(\Sigma) = \]
\[ b \]
\[ c \]
\[ e \lor f \]
\[ f \lor \neg g \]
Properties of BI

- Preserves logical equivalence

\[ \#\text{var}(\Sigma) \geq \#\text{var}(\text{BI}(\Sigma)) \]
\[ \#\text{lit}(\Sigma) \geq \#\text{lit}(\text{BI}(\Sigma)) \]
Figure – Comparing $\#\text{var}(\Sigma)$ with $\#\text{var}(\text{BI}(\Sigma))$. 
Figure – Comparing $\# \text{lit}(\Sigma)$ with $\# \text{lit}(\text{BI}(\Sigma))$. 
Limitations of the Basic Gate Detection and Replacement Preprocessings

- The replacement phase requires gates to be detected
  - The search space for gates is huge
  - The size of a gate can be huge as well
Limitations of the Basic Gate Detection and Replacement Preprocessings

- The replacement phase requires gates to be detected
  - The search space for gates is huge
  - The size of a gate can be huge as well

- Identifying "complex gates" is incompatible with the efficiency expected for a preprocessing:
  only "simple" gates are targeted

  literal equivalences \( y \leftrightarrow x_1 \)
  AND/OR gates \( y \leftrightarrow (x_1 \land \bar{x}_2 \land x_3) \)
  XOR gates \( y \leftrightarrow (x_1 \oplus \bar{x}_2) \)
The (explicit) identification phase can be replaced by an implicit identification phase.

Stated otherwise, there is no need to identify $f$ to determine that a gate of the form $y \leftrightarrow f(x_1, \ldots, x_n)$ exists in $\Sigma$. 

Let us ask Evert and Alessandro for some help...
The (explicit) identification phase can be replaced by an implicit identification phase.

Stated otherwise, there is no need to identify $f$ to determine that a gate of the form $y \leftrightarrow f(x_1, \ldots, x_n)$ exists in $\Sigma$.

Let us ask Evert and Alessandro for some help ...
Explicitly defines $y$ in terms of $X = \{x_1, \ldots, x_n\}$ iff there exists a formula $f(x_1, \ldots, x_n)$ over $X$ such that

\[ \Sigma | = y \iff f(x_1, \ldots, x_n) \]

Implicitly defines $y$ in terms of $X = \{x_1, \ldots, x_n\}$ iff for every canonical term $\gamma_X$ over $X$,

\[ \Sigma \land \gamma_X | = y \text{ or } \Sigma \land \gamma_X | = y \]

Beth's theorem: $\Sigma$ explicitly defines $y$ in terms of $X$ iff $\Sigma$ implicitly defines $y$ in terms of $X$. 

On Definability for Model Counting Meeting GT ALGA, GdR IM, Lille, October 15th, 2018


- \( \Sigma \) **explicitly defines** \( y \) in terms of \( X = \{x_1, \ldots, x_n\} \) iff there exists a formula \( f(x_1, \ldots, x_n) \) over \( X \) such that

\[
\Sigma \models y \iff f(x_1, \ldots, x_n)
\]

- \( \Sigma \) **implicitly defines** \( y \) in terms of \( X = \{x_1, \ldots, x_n\} \) iff for every canonical term \( \gamma_X \) over \( X \), we have

\[
\Sigma \land \gamma_X \models y \lor \Sigma \land \gamma_X \models \overline{y}
\]

**Beth’s theorem:** \( \Sigma \) explicitly defines \( y \) in terms of \( X \) iff \( \Sigma \) implicitly defines \( y \) in terms of \( X \)
Evert Willem Beth (1908–1964)

- **Σ explicitly defines** $y$ in terms of $X = \{x_1, \ldots, x_n\}$ **iff** there exists a formula $f(x_1, \ldots, x_n)$ over $X$ such that

$$
\Sigma \models y \iff f(x_1, \ldots, x_n)
$$

- **Σ implicitly defines** $y$ in terms of $X = \{x_1, \ldots, x_n\}$ **iff** for every canonical term $\gamma_X$ over $X$, we have

$$
\Sigma \land \gamma_X \models y \text{ or } \Sigma \land \gamma_X \models \overline{y}
$$
Evert Willem Beth (1908–1964)

- **Σ explicitly defines** $y$ in terms of $X = \{x_1, \ldots, x_n\}$ iff there exists a formula $f(x_1, \ldots, x_n)$ over $X$ such that

$$\Sigma \models y \iff f(x_1, \ldots, x_n)$$

- **Σ implicitly defines** $y$ in terms of $X = \{x_1, \ldots, x_n\}$ iff for every canonical term $\gamma_X$ over $X$, we have $\Sigma \land \gamma_X \models y$ or $\Sigma \land \gamma_X \models \neg y$

- **Beth’s theorem**: $Σ$ explicitly defines $y$ in terms of $X$ iff $Σ$ implicitly defines $y$ in terms of $X$
Padoa’s theorem:

Let $\Sigma'_X$ be equal to $\Sigma$ where each variable but those of $X$ have been renamed in a uniform way. If $y \not\in X$, then $\Sigma$ (implicitly) defines $y$ in terms of $X$ iff $\Sigma \land \Sigma'_X \land y \land \overline{y}'$ is inconsistent.
Padoa’s theorem:

Let $\Sigma'_X$ be equal to $\Sigma$ where each variable but those of $X$ have been renamed in a uniform way.

If $y \not\in X$, then $\Sigma$ (implicitly) defines $y$ in terms of $X$ iff $\Sigma \land \Sigma'_X \land y \land \overline{y'}$ is inconsistent.

Deciding whether $\Sigma$ (implicitly) defines $y$ in terms of $X$ is "only" coNP-complete.
There is no need to identify \( f \) to determine that a gate of the form \( y \leftrightarrow f(x_1, \ldots, x_n) \) exists in \( \Sigma \)
Overcoming the Limitations (2)

- There is no need to identify $f$ to determine that a gate of the form $y \leftrightarrow f(x_1, \ldots, x_n)$ exists in $\Sigma$
  - Gate identification = Explicit definability

\[ \text{There is no need to identify } f \text{ to compute } \Sigma[y \leftarrow f(x_1, \ldots, x_n)] \]

- The replacement phase can be replaced by an output variable elimination phase: if $y \leftrightarrow f(x_1, \ldots, x_n)$ is a gate of $\Sigma$, then
  \[ \Sigma[y \leftarrow f(x_1, \ldots, x_n)] \equiv \exists y. \Sigma \]
There is **no need to identify** $f$ to determine that a gate of the form $y \leftrightarrow f(x_1, \ldots, x_n)$ exists in $\Sigma$

- Gate identification = Explicit definability
- Explicit definability = Implicit definability (Beth’s theorem)
There is no need to identify $f$ to determine that a gate of the form $y \leftrightarrow f(x_1, \ldots, x_n)$ exists in $\Sigma$

- Gate identification $=$ Explicit definability
- Explicit definability $=$ Implicit definability (Beth’s theorem)
- One call to a SAT solver is enough to decide whether $\Sigma$ defines $y$ in terms of $\{x_1, \ldots, x_n\}$ (thanks to Padoa’s theorem)
There is no need to identify $f$ to determine that a gate of the form $y \leftrightarrow f(x_1, \ldots, x_n)$ exists in $\Sigma$

- Gate identification = Explicit definability
- Explicit definability = Implicit definability (Beth’s theorem)
- One call to a SAT solver is enough to decide whether $\Sigma$ defines $y$ in terms of $\{x_1, \ldots, x_n\}$ (thanks to Padoa’s theorem)

There is no need to identify $f$ to compute $\Sigma[y \leftarrow f(x_1, \ldots, x_n)]$
There is no need to identify $f$ to determine that a gate of the form $y \leftrightarrow f(x_1, \ldots, x_n)$ exists in $\Sigma$.

- Gate identification = Explicit definability
- Explicit definability = Implicit definability (Beth’s theorem)
- One call to a SAT solver is enough to decide whether $\Sigma$ defines $y$ in terms of $\{x_1, \ldots, x_n\}$ (thanks to Padoa’s theorem)

There is no need to identify $f$ to compute $\Sigma[y \leftarrow f(x_1, \ldots, x_n)]$.

- The replacement phase can be replaced by an output variable elimination phase: if $y \leftrightarrow f(x_1, \ldots, x_n)$ is a gate of $\Sigma$, then

$$\Sigma[y \leftarrow f(x_1, \ldots, x_n)] \equiv \exists y. \Sigma$$
A two-step preprocessing

- "Identification = Bipartition":
  compute a definability bipartition \( \langle I, O \rangle \) of \( \Sigma \) such that 
  \( I \cup O = \text{Var}(\Sigma) \), \( I \cap O = \emptyset \), and \( \Sigma \) defines every variable 
  \( o \in O \) in terms of \( I \)
A two-step preprocessing

- "Identification = Bipartition": compute a definability bipartition \( \langle I, O \rangle \) of \( \Sigma \) such that
  \( I \cup O = \text{Var}(\Sigma) \), \( I \cap O = \emptyset \), and \( \Sigma \) defines every variable \( o \in O \) in terms of \( I \)

- "Replacement = Elimination": compute \( \exists E.\Sigma \) for \( E \subseteq O \)
A two-step preprocessing

- "Identification = Bipartition":
  compute a definability bipartition $\langle I, O \rangle$ of $\Sigma$ such that $I \cup O = \text{Var}(\Sigma)$, $I \cap O = \emptyset$, and $\Sigma$ defines every variable $o \in O$ in terms of $I$

- "Replacement = Elimination":
  compute $\exists E. \Sigma$ for $E \subseteq O$

- Steps B and E of B + E can be tuned in order to keep the preprocessing phase light from a computational standpoint (NP-preprocessing)
Identifying $u$ as an Output Variable and Eliminating it

**Identification:**

$$\Sigma \land \Sigma'_{\{x,y,z\}} \land u \land \overline{u}'$$

is inconsistent

- $\overline{x} \lor u \lor v$
- $\overline{x} \lor \overline{y} \lor u$
- $\overline{x} \lor \overline{z} \lor u$
- $x \lor \overline{u}$
- $y \lor z \lor \overline{u}$
- $\overline{x} \lor u' \lor v'$
- $\overline{x} \lor \overline{y} \lor u'$
- $\overline{x} \lor \overline{z} \lor u'$
- $x \lor \overline{u}'$
- $y \lor z \lor \overline{u}'$
- $u$
- $\overline{u}'$
Identifying $u$ as an Output Variable and Eliminating it

**Identification:**

\[ \Sigma \land \Sigma'_{\{x,y,z\}} \land u \land \overline{u}' \text{ is inconsistent} \]

\begin{align*}
\overline{x} \lor u \lor v \\
\overline{x} \lor \overline{y} \lor u \\
\overline{x} \lor z \lor u \\
x \lor \overline{u} \\
y \lor z \lor \overline{u} \\
\overline{x} \lor u' \lor v' \\
\overline{x} \lor \overline{y} \lor u' \\
\overline{x} \lor z \lor u' \\
x \lor \overline{u}' \\
y \lor z \lor \overline{u}' \\
u \\
\overline{u}'
\end{align*}

**Elimination:**

computing resolvents over $u$

\begin{align*}
\overline{x} \lor v \lor x & \quad \text{valid} \\
\overline{x} \lor v \lor y \lor z & \quad \text{valid} \\
\overline{x} \lor \overline{y} \lor x & \quad \text{valid} \\
\overline{x} \lor \overline{y} \lor y \lor z & \quad \text{valid} \\
\overline{x} \lor z \lor x & \quad \text{valid} \\
\overline{x} \lor z \lor y \lor z & \quad \text{valid}
\end{align*}
Identifying \( u \) as an Output Variable and Eliminating it

Identification:
\[
\Sigma \land \Sigma'_{\{x,y,z\}} \land u \land \overline{u}' \text{ is inconsistent}
\]

\[
\begin{align*}
\overline{x} \lor u \lor v \\
\overline{x} \lor \overline{y} \lor u \\
\overline{x} \lor \overline{z} \lor u \\
x \lor \overline{u} \\
y \lor z \lor \overline{u} \\
\overline{x} \lor u' \lor v' \\
\overline{x} \lor \overline{y} \lor u' \\
\overline{x} \lor \overline{z} \lor u' \\
x \lor \overline{u}' \\
y \lor z \lor \overline{u}' \\
u \\
\overline{u}'
\end{align*}
\]

Elimination:
computing resolvents over \( u \)

\[
\begin{align*}
\overline{x} \lor v \lor x & \quad \text{valid} \\
\overline{x} \lor v \lor y \lor z & \quad \text{valid} \\
\overline{x} \lor \overline{y} \lor x & \quad \text{valid} \\
\overline{x} \lor \overline{y} \lor y \lor z & \quad \text{valid} \\
\overline{x} \lor \overline{z} \lor x & \quad \text{valid} \\
\overline{x} \lor \overline{z} \lor y \lor z & \quad \text{valid}
\end{align*}
\]

\[
\|\Sigma\| = \|\overline{x} \lor v \lor y \lor z\| = 15
\]
Both steps $B$ and $E$ of $B + E$ can be tuned in order to keep the preprocessing phase **light from a computational standpoint**

- It is not necessary to determine a definability bipartition $\langle I, O \rangle$ with $|I|$ minimal
  - $\Rightarrow B$ is a **greedy algorithm** (one definability test per variable)
  - $\Rightarrow$ Only the minimality of $I$ for $\subseteq$ is guaranteed
Both steps $B$ and $E$ of $B + E$ can be tuned in order to keep the preprocessing phase light from a computational standpoint

- It is not necessary to determine a definability bipartition $\langle I, O \rangle$ with $|I|$ minimal
  $\Rightarrow$ $B$ is a greedy algorithm (one definability test per variable)
  $\Rightarrow$ Only the minimality of $I$ for $\subseteq$ is guaranteed

- It is not necessary to eliminate in $\Sigma$ every variable of $O$ but focusing on a subset $E \subseteq O$ is enough
  $\Rightarrow$ Eliminating every output variable could lead to an exponential blow up
  $\Rightarrow$ The elimination of $y \in O$ is committed only if $|\Sigma|$ after the elimination step and some additional preprocessing techniques (occurrence simplification and vivification) remains small enough
Experiments

Objectives:
- Evaluating the computational benefits offered by $B + E$ when used upstream to state-of-the-art model counters:
  - the search-based model counter Cachet
  - the search-based model counter SharpSAT
  - the compilation-based model counter C2D (used with `-count -in_memory -smooth_all`)
  - the compilation-based model counter D4
Objectives:

- Evaluating the computational benefits offered by $B + E$ when used upstream to state-of-the-art model counters:
  - the search-based model counter Cachet
  - the search-based model counter SharpSAT
  - the compilation-based model counter C2D (used with `-count -in_memory -smooth_all`)
  - the compilation-based model counter D4

- Comparing the benefits offered by $B + E$ with those offered by our previous preprocessor pmc (based on gate identification and replacement) or with no preprocessing
Empirical Setting

- 703 CNF instances from the SAT LIBrary
- 8 data sets: BN (Bayesian networks) (192), BMC (Bounded Model Checking) (18), Circuit (41), Configuration (35), Handmade (58), Planning (248), Random (104), Qif (7) (Quantitative Information Flow analysis - security)

- Cluster of Intel Xeon E5-2643 (3.30 GHz) processors with 32 GiB RAM on Linux CentOS
- Time-out = 1h
- Memory-out = 7.6 GiB
Empirical Results: Reduction Achieved

**Figure** – Reduction achieved by $B + E$

(a) #var reduction

(b) #lit reduction
Empirical Results: Time Saving

(a) Cachet vs. $B+E+Cachet$

(b) SharpSAT vs. $B+E+SharpSAT$

**Figure** – Time saved by using $B+E$ upstream
Empirical Results: Time Saving

Figure – Time saved by using $B + E$ upstream
Empirical Results

Figure – Cachet depending on the preprocessing used
Empirical Results

\[ \sum \text{pmc}(\Sigma) \]

\[ b + e(\Sigma) \]

\begin{figure}
\centering
\includegraphics[width=\textwidth]{chart.png}
\caption{SharpSAT depending on the preprocessing used}
\end{figure}
Empirical Results

Figure – C2D depending on the preprocessing used
Empirical Results

\[ \sum \text{pmc}(\Sigma) + b + e(\Sigma) \]

Figure – D4 depending on the preprocessing used
Conclusion

- Design and implementation of the $B + E$ preprocessor
- Empirical evaluation of $B + E$: for several model counters $mc$, $mc(B + E(.))$ proves computationally more efficient than $mc(.)$
- "Real" instances are structured ones
Conclusion

▶ Design and implementation of the B + E preprocessor
▶ Empirical evaluation of B + E: for several model counters mc, mc(B + E(.)) proves computationally more efficient than mc(.)
▶ "Real" instances are structured ones

Perspectives

▶ Developing other ordering heuristics for B
▶ Investigating the connections to projected model counting: computing $\left\| \exists E. \Sigma \right\|$ given a set $E$ of variables and a formula $\Sigma$