On Definability for Model Counting

Jean-Marie Lagniez[1], Emmanuel Lonca[1] and Pierre Marquis[1,2]

[1]CRIL, U. Artois & CNRS
[2] Institut Universitaire de France

Meeting GT ALGA, GdR IM, Lille, October 15th, 2018

UNIVERSITÉ D'ARTOIS

The Beyond NP Era

2

- 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

3

- $\Sigma \mapsto \|\Sigma\| \ = ?$

Model Counting

3

- $\Sigma \mapsto \|\Sigma\| = ?$
- $\Sigma = (x \vee y) \wedge (\neg y \vee z)$

Model Counting

3

- $\Sigma \mapsto \|\Sigma\| = ?$
- $\Sigma = (x \vee y) \wedge (\neg y \vee z)$
- The models of Σ over $\{x, y, z\}$ are :

  011
  100
  101
  111

Model Counting

3

- $\Sigma \mapsto \|\Sigma\| \ = ?$
- $\Sigma = (x \vee y) \wedge (\neg y \vee z)$
- The models of Σ over $\{x, y, z\}$ are :

  011
  100
  101
  111

- $\|\Sigma\| = 4$

Model Counting

- Counting the models of a propositional formula is a key task for a number of problems (especially in AI):
  - probabilistic inference
  - stochastic planning
  - ...

Model Counting

- 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

Model Counting

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 #SAT is a computationally hard task: #P-complete
- Even for subsets of formulae for which SAT is easy (e.g., monotone Krom formulae)!

Model Counting

- 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
- Even for subsets of formulae for which 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):

$$\mathrm{PH} \subseteq \mathrm{P}^{\#\mathrm{P}}$$

Model Counting

- **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.)
  - ...

Model Counting

5

- **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

$$\texttt{CNF} \rightarrow \texttt{CNF}$$

Preprocessings

6

- 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

Knowledge Compilation vs. Preprocessing for #SAT

7

- 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**

CNF Σ → compilation → d-DNNF Ψ → model counting → $\|\Sigma\|$

Knowledge Compilation vs. Preprocessing for #SAT

7

- 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**

  CNF Σ → [compilation] → d-DNNF Ψ → [model counting] → $\|\Sigma\|$

- **preprocessing**

  CNF Σ → [preprocessing] → CNF Φ → [model counting] → $\|\Sigma\|$

Knowledge Compilation vs. Preprocessing for #SAT

7

- 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**

  CNF Σ — [compilation] → d-DNNF Ψ — [model counting] → $\|\Sigma\|$

- **preprocessing**

  CNF Σ — [preprocessing] → CNF Φ — [model counting] → $\|\Sigma\|$

- The two approaches can be **combined**

Dozens of P-Preprocessings

8

- 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)
- ...

Use in State-of-the-Art SAT Solvers

9

- `Glucose` (exploits the `SatELite` preprocessor)
- `Lingeling` (has an internal preprocessor)
- `Riss` (use of the `Coprocessor` preprocessor)
- ...

Reducing What?

10

CNF $\Sigma \mapsto$ CNF $p(\Sigma)$

- What are the connections between Σ and $p(\Sigma)$?
- Removing clauses from Σ
- Removing literals in the clauses of Σ
- ...

Looking for IES or Minimal CNF is often too Expensive

- A clause δ of a CNF Σ is redundant if and only if $\Sigma \setminus \{\delta\} \models \delta$
- A CNF Σ is irredundant if and only if it does not contain any redundant clause
- A subset Σ' of a CNF Σ is an irredundant equivalent subset (IES) of Σ if and only if Σ' is irredundant and $\Sigma' \equiv \Sigma$

- Deciding whether a CNF Σ is irredundant is NP-complete
- Deciding whether a CNF Σ' is an irredundant equivalent subset (IES) of a CNF Σ is D^p-complete
- Given an integer k, deciding whether a CNF Σ has an IES of size at most k is Σ_2^p-complete
- Given an integer k, deciding whether there exists a CNF formula Σ' with at most k literals (or with at most k clauses) equivalent to a given CNF Σ is Σ_2^p-complete

Preserving What?

12

- Logical equivalence
- Queries over the input alphabet
- Number of models
- Satisfiability
- ...

Measuring the Impact of a Preprocessing

Several measures for the reduction achieved can be considered:

- The number of variables in the input CNF Σ
- The size of Σ (the number of literals or the number of clauses in it)
- The value of some structural parameters for Σ
- ...

Example: Subsumption Elimination

14

A clause δ_1 subsumes a clause δ_2
if every literal of δ_1 is a literal of δ_2

$SE : (x_1 \lor x_2) \land (x_1 \lor x_2 \lor \bar{x}_3) \mapsto x_1 \lor x_2$

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

- $\#var(\Sigma) \geq \#var(\mathtt{SE}(\Sigma))$
- $\#lit(\Sigma) \geq \#lit(\mathtt{SE}(\Sigma))$

The Gate Detection and Replacement Family

15

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

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

$$\Sigma = \begin{array}{l} \overline{x} \vee u \vee v \\ \overline{x} \vee \overline{y} \vee u \\ \overline{x} \vee \overline{z} \vee u \\ x \vee \overline{u} \\ y \vee z \vee \overline{u} \end{array}$$

The Gate Detection and Replacement Family

15

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

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

$$\Sigma = \begin{array}{l} \overline{x} \vee u \vee v \\ \overline{x} \vee \overline{y} \vee u \\ \overline{x} \vee \overline{z} \vee u \\ x \vee \overline{u} \\ y \vee z \vee \overline{u} \end{array} \qquad u \leftrightarrow (x \wedge (y \vee z))$$

The Gate Detection and Replacement Family

15

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

$$\Sigma = \begin{array}{l} \overline{x} \vee u \vee v \\ \overline{x} \vee \overline{y} \vee u \\ \overline{x} \vee \overline{z} \vee u \\ x \vee \overline{u} \\ y \vee z \vee \overline{u} \end{array} \qquad u \leftrightarrow (x \wedge (y \vee z))$$

$\Sigma \equiv$
$(\overline{x} \vee u \vee v) \wedge (u \leftrightarrow (x \wedge (y \vee z)))$ detection

The Gate Detection and Replacement Family

15

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

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

$$\Sigma = \begin{array}{l} \overline{x} \vee u \vee v \\ \overline{x} \vee \overline{y} \vee u \\ \overline{x} \vee \overline{z} \vee u \\ x \vee \overline{u} \\ y \vee z \vee \overline{u} \end{array} \qquad u \leftrightarrow (x \wedge (y \vee z))$$

$\Sigma \equiv$

$(\overline{x} \vee u \vee v) \wedge (u \leftrightarrow (x \wedge (y \vee z)))$ detection

$(\overline{x} \vee (x \wedge (y \vee z)) \vee v) \wedge (u \leftrightarrow (x \wedge (y \vee z)))$ replacement

15

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

$$\Sigma = \begin{array}{l} \overline{x} \vee u \vee v \\ \overline{x} \vee \overline{y} \vee u \\ \overline{x} \vee \overline{z} \vee u \\ x \vee \overline{u} \\ y \vee z \vee \overline{u} \end{array} \qquad u \leftrightarrow (x \wedge (y \vee z))$$

$\Sigma \equiv$

$(\overline{x} \vee u \vee v) \wedge (u \leftrightarrow (x \wedge (y \vee z)))$	detection
$(\overline{x} \vee (x \wedge (y \vee z)) \vee v) \wedge (u \leftrightarrow (x \wedge (y \vee z)))$	replacement
$(\overline{x} \vee y \vee z \vee v) \wedge (u \leftrightarrow (x \wedge (y \vee z)))$	normalization

The Gate Detection and Replacement Family

15

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

$$\Sigma = \begin{array}{l} \overline{x} \vee u \vee v \\ \overline{x} \vee \overline{y} \vee u \\ \overline{x} \vee \overline{z} \vee u \\ x \vee \overline{u} \\ y \vee z \vee \overline{u} \end{array} \qquad u \leftrightarrow (x \wedge (y \vee z))$$

$\Sigma \equiv$

$(\overline{x} \vee u \vee v) \wedge (u \leftrightarrow (x \wedge (y \vee z)))$	detection
$(\overline{x} \vee (x \wedge (y \vee z)) \vee v) \wedge (u \leftrightarrow (x \wedge (y \vee z)))$	replacement
$(\overline{x} \vee y \vee z \vee v) \wedge (u \leftrightarrow (x \wedge (y \vee z)))$	normalization

$$\|\Sigma\| = \|\Sigma[u \leftarrow (x \wedge (y \vee z))]\| = \|\overline{x} \vee y \vee z \vee v\| = 15$$

The Gate Detection and Replacement Family

16

- 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 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 ℓ, all the literals ℓ' which can be found equivalent to ℓ using BCP are replaced by ℓ in Σ
- Taking advantage of BCP makes it more efficient than a "syntactic detection" (if two binary clauses stating an equivalence between two literals ℓ and ℓ' occur in Σ, then those literals are found equivalent using BCP, but the converse does not hold)

Literal Equivalence (LE)

Algorithm 1: LE

input : a CNF formula Σ

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

$\Phi \leftarrow \Sigma$; Unmark all variables of Φ;

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

  // detection

  mark $\mathit{var}(\ell)$;

  $\mathcal{P}_\ell \leftarrow \mathtt{BCP}(\Phi \cup \{\ell\})$;

  $\mathcal{N}_\ell \leftarrow \mathtt{BCP}(\Phi \cup \{\sim\ell\})$;

  $\Gamma \leftarrow \{\ell \leftrightarrow \ell' | \ell' \neq \ell \text{ and } \ell' \in \mathcal{P}_\ell \text{ and } \sim\ell' \in \mathcal{N}_\ell\}$;

  // replacement

  foreach $\ell \leftrightarrow \ell' \in \Gamma$ **do**

    replace ℓ by ℓ' in Φ;

return Φ

Literal Equivalence (LE): Example

$\Sigma =$

$a \vee b \vee c \vee \neg d$	$\neg a \vee \neg b \vee \neg c \vee d$
$a \vee b \vee \neg c$	$\neg a \vee \neg b \vee c$
$\neg a \vee b$	$a \vee \neg b$
$\neg e \vee \neg f \vee h$	$e \vee f \vee g$
$e \vee \neg g$	$\neg e \vee \neg h$

Assume that the variables of Σ are considered in the following ordering: $a < b < c < d < e < f < g < h$

The equivalences $(a \Leftrightarrow b) \wedge (b \Leftrightarrow c) \wedge (c \Leftrightarrow d) \wedge (e \Leftrightarrow \neg f)$ are detected

$\texttt{LE}(\Sigma) =$

$\neg f \vee \neg g \quad f \vee \neg h$

Properties of LE

20

- Preserves the number of models (but not logical equivalence)
- $\#var(\Sigma) \geq \#var(\texttt{LE}(\Sigma))$
- $\#lit(\Sigma) \geq \#lit(\texttt{LE}(\Sigma))$

LE: Reduction of the Number of Variables

21

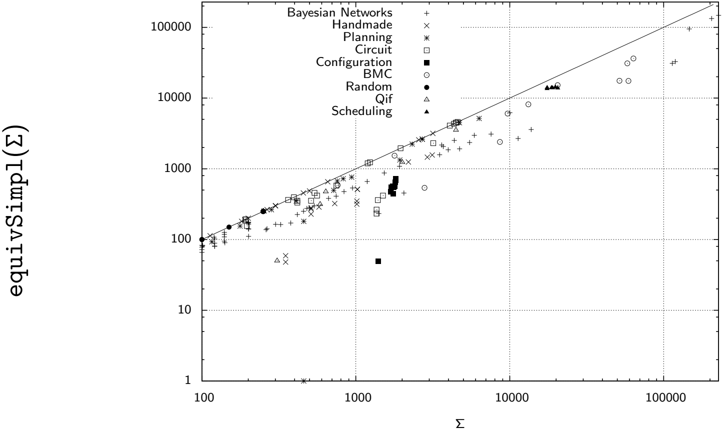


FIGURE – Comparing $\#var(\Sigma)$ with $\#var(\mathtt{LE}(\Sigma))$.

LE: Reduction of the Size

22

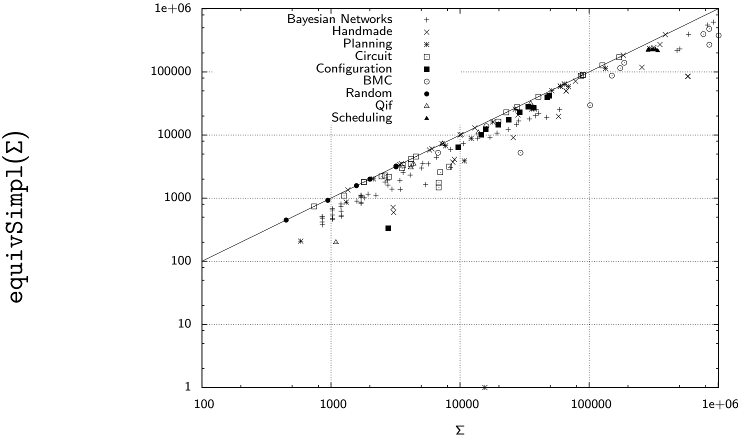


FIGURE – Comparing $\#lit(\Sigma)$ with $\#lit(\texttt{LE}(\Sigma))$.

Backbone Identification (BI)

- The backbone of a CNF formula Σ is the set of all literals which are implied by Σ when Σ 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 Σ explicit, to conjoin it to Σ, and to use BCP (Boolean Constraint Propagation) on the resulting set of clauses
- NP-preprocessing

Backbone Identification (BI)

Algorithm 2: BI Backbone Identification

input : a CNF formula Σ

output: the CNF $\texttt{BCP}(\Sigma \cup B)$, where $\mathcal{B}$ is the backbone of Σ

$\mathcal{B} \leftarrow \emptyset$;

$\mathcal{I} \leftarrow \texttt{solve}(\Sigma)$;

while $\exists \ell \in \mathcal{I}$ *s.t.* $\ell \notin \mathcal{B}$ **do**

  $\mathcal{I}' \leftarrow \texttt{solve}(\Sigma \cup \{\sim\ell\})$;

  if $\mathcal{I}' = \emptyset$ **then** $\mathcal{B} \leftarrow \mathcal{B} \cup \{\ell\}$**else** $\mathcal{I} \leftarrow \mathcal{I} \cap \mathcal{I}'$;

return $\texttt{BCP}(\Sigma \cup \mathcal{B})$

Backbone Identification (BI): Example

25

$\Sigma =$

$a \vee b$

$\neg a \vee b$

$\neg b \vee c$

$c \vee d$

$\neg c \vee e \vee f$

$f \vee \neg g$

The backbone of Σ is equal to $B = \{b, c\}$

$\texttt{BI}(\Sigma) =$

b

c

$e \vee f$

$f \vee \neg g$

Properties of BI

- Preserves logical equivalence
- $\#var(\Sigma) \geq \#var(\texttt{BI}(\Sigma))$
- $\#lit(\Sigma) \geq \#lit(\texttt{BI}(\Sigma))$

BI: Reduction of the Number of Variables

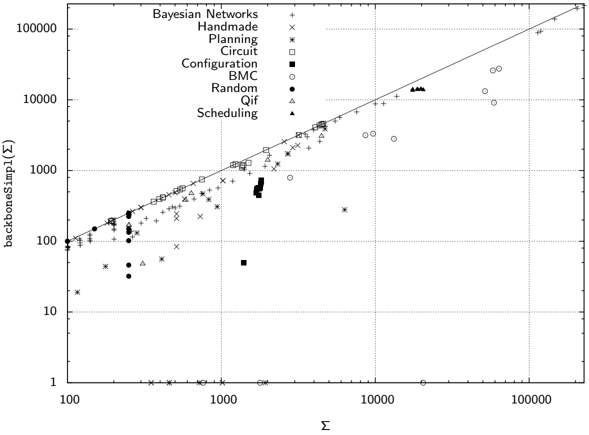


FIGURE – Comparing $\#var(\Sigma)$ with $\#var(\texttt{BI}(\Sigma))$.

BI: Reduction of the Size

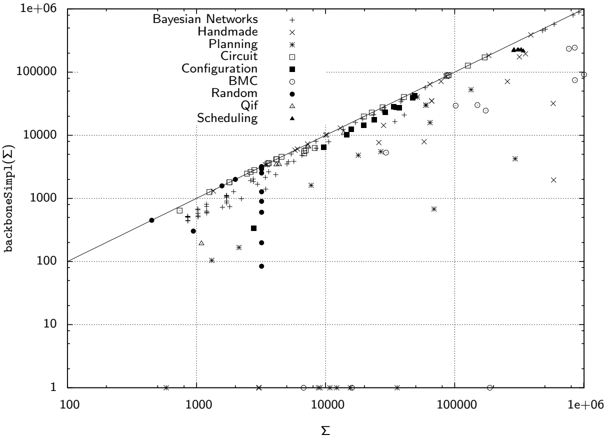


Figure – Comparing $\#lit(\Sigma)$ with $\#lit(\texttt{BI}(\Sigma))$.

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

29

- 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	$y \leftrightarrow (x_1 \land \overline{x_2} \land x_3)$
XOR gates	$y \leftrightarrow (x_1 \oplus \overline{x_2})$

Overcoming the Limitations (1)

30

- 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 Σ

Overcoming the Limitations (1)

30

- 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 Σ

- Let us ask Evert and Alessandro for some help ...

Evert Willem Beth (1908–1964)

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

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

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

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

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

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

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

- Σ **implicitly defines** y in terms of $X = \{x_1, \ldots, x_n\}$ iff for every canonical term γ_X over X, we have $\Sigma \wedge \gamma_X \models y$ or $\Sigma \wedge \gamma_X \models \overline{y}$
- Beth's theorem: Σ **explicitly defines** y in terms of X iff Σ **implicitly defines** y in terms of X

Alessandro Padoa (1868–1937)

32

Padoa's theorem:

Let Σ'_X be equal to Σ where each variable but those of X have been renamed in a uniform way
If $y \notin X$, then Σ (implicitly) defines y in terms of X iff $\Sigma \wedge \Sigma'_X \wedge y \wedge \overline{y'}$ is inconsistent

Alessandro Padoa (1868–1937)

Padoa's theorem:

Let Σ'_X be equal to Σ where each variable but those of X have been renamed in a uniform way
If $y \notin X$, then Σ (implicitly) defines y in terms of X iff $\Sigma \wedge \Sigma'_X \wedge y \wedge \overline{y'}$ is inconsistent

Deciding whether Σ (implicitly) defines y in terms of X is "only" coNP-complete

Overcoming the Limitations (2)

33

- 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 Σ

Overcoming the Limitations (2)

33

- 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 Σ
  - **Gate identification = Explicit definability**

Overcoming the Limitations (2)

33

- 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 Σ
  - **Gate identification = Explicit definability**
  - **Explicit definability = Implicit definability (Beth's theorem)**

Overcoming the Limitations (2)

33

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

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

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

Overcoming the Limitations (2)

33

- 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 Σ
  - **Gate identification = Explicit definability**
  - **Explicit definability = Implicit definability (Beth's theorem)**
  - **One call to a SAT solver** is enough to decide whether Σ 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 Σ, then

$$\Sigma[y \leftarrow f(x_1, \ldots, x_n)] \equiv \exists y.\Sigma$$

The B + E Preprocessing

A two-step preprocessing

- "Identification = Bipartition":
  compute a **definability bipartition** $\langle I, O \rangle$ of Σ such that $I \cup O = Var(\Sigma)$, $I \cap O = \emptyset$, and Σ 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 Σ such that $I \cup O = Var(\Sigma)$, $I \cap O = \emptyset$, and Σ 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 Σ such that $I \cup O = Var(\Sigma)$, $I \cap O = \emptyset$, and Σ defines every variable $o \in O$ in terms of I
- "Replacement = Elimination":
  compute $\exists E.\Sigma$ for $E \subseteq O$

- Steps $\mathtt{B}$ and $\mathtt{E}$ of $\mathtt{B} + \mathtt{E}$ 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

35

Identification:

$\Sigma \wedge \Sigma'_{\{x,y,z\}} \wedge u \wedge \overline{u'}$ is inconsistent

$\overline{x} \vee u \vee v$
$\overline{x} \vee \overline{y} \vee u$
$\overline{x} \vee \overline{z} \vee u$
$x \vee \overline{u}$
$y \vee z \vee \overline{u}$
$\overline{x} \vee u' \vee v'$
$\overline{x} \vee \overline{y} \vee u'$
$\overline{x} \vee \overline{z} \vee u'$
$x \vee \overline{u'}$
$y \vee z \vee \overline{u'}$
u
$\overline{u'}$

Identifying u as an Output Variable and Eliminating it

35

Identification:

$\Sigma \wedge \Sigma'_{\{x,y,z\}} \wedge u \wedge \overline{u'}$ is inconsistent

$\overline{x} \vee u \vee v$
$\overline{x} \vee \overline{y} \vee u$
$\overline{x} \vee \overline{z} \vee u$
$x \vee \overline{u}$
$y \vee z \vee \overline{u}$
$\overline{x} \vee u' \vee v'$
$\overline{x} \vee \overline{y} \vee u'$
$\overline{x} \vee \overline{z} \vee u'$
$x \vee \overline{u'}$
$y \vee z \vee \overline{u'}$
u
$\overline{u'}$

Elimination:

computing resolvents over u

$\overline{x} \vee v \vee x$	valid
$\overline{x} \vee v \vee y \vee z$	
$\overline{x} \vee \overline{y} \vee x$	valid
$\overline{x} \vee \overline{y} \vee y \vee z$	valid
$\overline{x} \vee \overline{z} \vee x$	valid
$\overline{x} \vee \overline{z} \vee y \vee z$	valid

Identifying u as an Output Variable and Eliminating it

35

Identification:

$\Sigma \wedge \Sigma'_{\{x,y,z\}} \wedge u \wedge \overline{u'}$ is inconsistent

$\overline{x} \vee u \vee v$
$\overline{x} \vee \overline{y} \vee u$
$\overline{x} \vee \overline{z} \vee u$
$x \vee \overline{u}$
$y \vee z \vee \overline{u}$
$\overline{x} \vee u' \vee v'$
$\overline{x} \vee \overline{y} \vee u'$
$\overline{x} \vee \overline{z} \vee u'$
$x \vee \overline{u'}$
$y \vee z \vee \overline{u'}$
u
$\overline{u'}$

Elimination:

computing resolvents over u

$\overline{x} \vee v \vee x$	valid
$\overline{x} \vee v \vee y \vee z$	
$\overline{x} \vee \overline{y} \vee x$	valid
$\overline{x} \vee \overline{y} \vee y \vee z$	valid
$\overline{x} \vee \overline{z} \vee x$	valid
$\overline{x} \vee \overline{z} \vee y \vee z$	valid

$\|\Sigma\| = \|\overline{x} \vee v \vee y \vee z\| = 15$

Tuning the Computational Effort

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

Tuning the Computational Effort

36

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 Σ 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**

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:

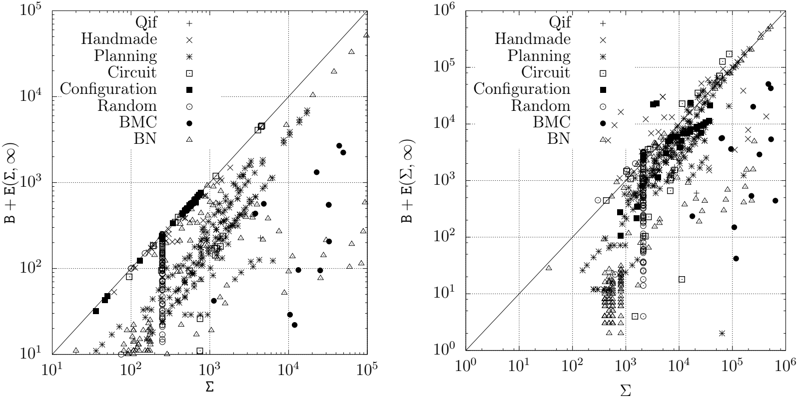
- Evaluating the computational benefits offered by $\mathtt{B} + \mathtt{E}$ when used upstream to state-of-the-art model counters:
  - the search-based model counter `Cachet`
  - the search-based model counter `SharpSAT`
  - the compilation-based model counter `C2D` (used with `-count -in_memory -smooth_all`)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

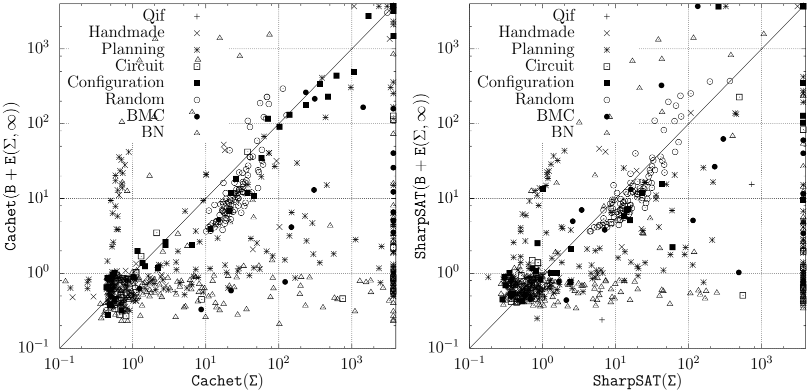
Empirical Results: Reduction Achieved



(a) #var reduction (b) #lit reduction

FIGURE – Reduction achieved by B + E

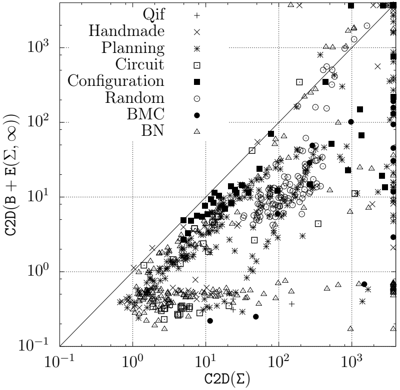
Empirical Results: Time Saving



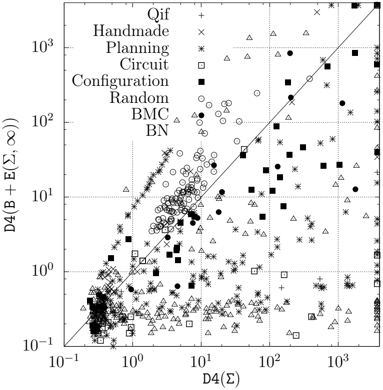
(a) Cachet vs. B + E+Cachet (b) SharpSAT vs. B + E+SharpSAT

Figure – Time saved by using B + E upstream

Empirical Results: Time Saving



(a) C2D vs. B + E+C2D



(b) D4 vs. B + E+D4

FIGURE – Time saved by using B + E upstream

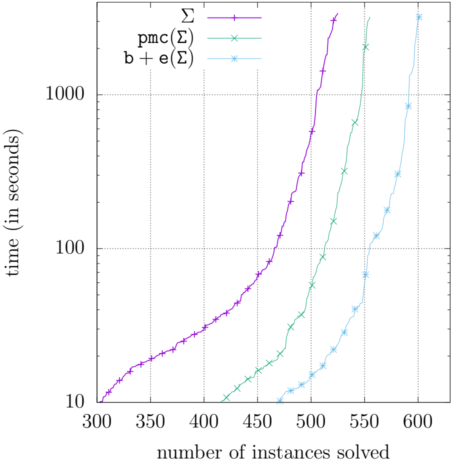


FIGURE – Cachet depending on the preprocessing used

Empirical Results

43

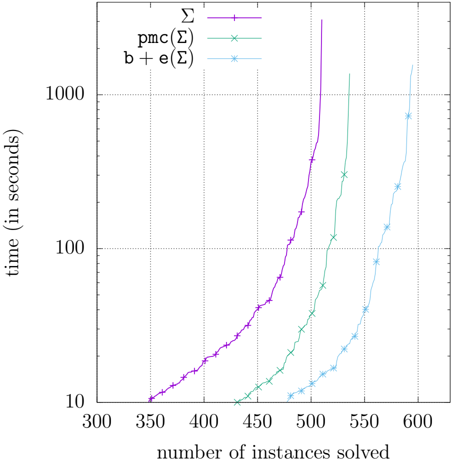


FIGURE – SharpSAT depending on the preprocessing used

Empirical Results

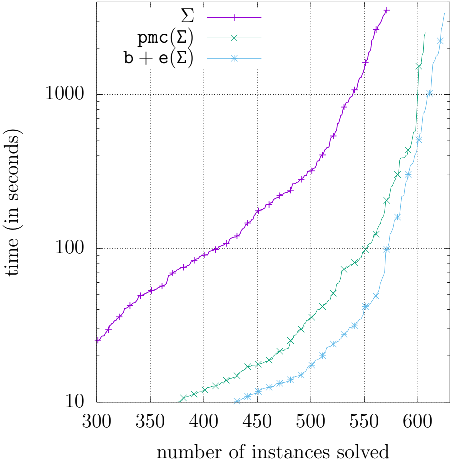


FIGURE – C2D depending on the preprocessing used

Empirical Results

45

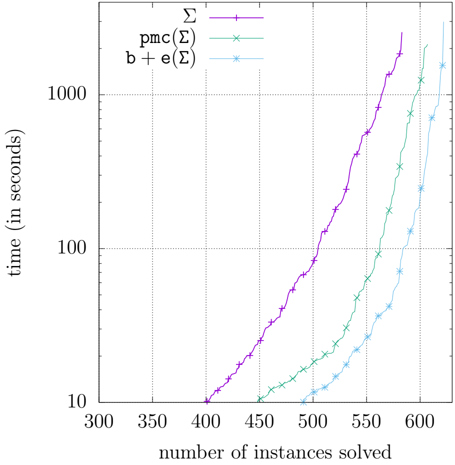


FIGURE – D4 depending on the preprocessing used

Conclusion and Perspectives

Conclusion

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

Conclusion and Perspectives

Conclusion

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

Perspectives

- Developing other ordering heuristics for $\texttt{B}$
- Investigating the connections to **projected model counting:** computing $\|\exists E.\Sigma\|$ given a set E of variables and a formula Σ