Knowledge Compilation with Empowerment

Lucas Bordeaux[1] and **Joao Marques-Silva**[2,3]

[1]Microsoft Research Cambridge

[2]University College Dublin

[3]IST/INESC-ID, TU Lisbon

SOFSEM, Spindleruv Mlyn, Czech Republic, January 2012

Motivation

- Start from concise propositional representation of a constraint
  - E.g. automatically encoded & without key redundancies added
  - E.g. $\varphi = (x \vee y \vee z) \wedge (x \vee y \vee \neg z)$ misses key redundant clause
    - $\varphi \wedge \neg x \vDash y$, but this is not derivable with unit propagation
    - φ is not propagation complete

Motivation

- Start from concise propositional representation of a constraint
  - E.g. automatically encoded & without key redundancies added
  - E.g. $\varphi = (x \vee y \vee z) \wedge (x \vee y \vee \neg z)$ misses key redundant clause
    - $\varphi \wedge \neg x \models y$, but this is not derivable with unit propagation
    - φ is not propagation complete

- Compute propagation-complete representation of the constraint
  - I.e. can use unit propagation to deduce all logically entailed literals
  - E.g. $\varphi' = (x \vee y \vee z) \wedge (x \vee y \vee \neg z) \wedge (x \vee y)$ is propagation complete
    - $\varphi' \wedge \neg x \models y$ is derivable with unit propagation
    - $\varphi' \wedge \neg y \models x$ is derivable with unit propagation

Motivation

- Start from concise propositional representation of a constraint
  - E.g. automatically encoded & without key redundancies added
  - E.g. $\varphi = (x \vee y \vee z) \wedge (x \vee y \vee \neg z)$ misses key redundant clause
    - $\varphi \wedge \neg x \vDash y$, but this is not derivable with unit propagation
    - φ is not propagation complete
- Compute propagation-complete representation of the constraint
  - I.e. can use unit propagation to deduce all logically entailed literals
  - E.g. $\varphi' = (x \vee y \vee z) \wedge (x \vee y \vee \neg z) \wedge (x \vee y)$ is propagation complete
    - $\varphi' \wedge \neg x \vDash y$ is derivable with unit propagation
    - $\varphi' \wedge \neg y \vDash x$ is derivable with unit propagation
- Make this representation compact.

Motivation

- Start from concise propositional representation of a constraint
  - E.g. automatically encoded & without key redundancies added
  - E.g. $\varphi = (x \vee y \vee z) \wedge (x \vee y \vee \neg z)$ misses key redundant clause
    - $\varphi \wedge \neg x \vDash y$, but this is not derivable with unit propagation
    - φ is not propagation complete
- Compute propagation-complete representation of the constraint
  - I.e. can use unit propagation to deduce all logically entailed literals
  - E.g. $\varphi' = (x \vee y \vee z) \wedge (x \vee y \vee \neg z) \wedge (x \vee y)$ is propagation complete
    - $\varphi' \wedge \neg x \vDash y$ is derivable with unit propagation
    - $\varphi' \wedge \neg y \vDash x$ is derivable with unit propagation
- Make this representation compact. How?

Motivation

- Start from concise propositional representation of a constraint
  - E.g. automatically encoded & without key redundancies added
  - E.g. $\varphi = (x \vee y \vee z) \wedge (x \vee y \vee \neg z)$ misses key redundant clause
    - $\varphi \wedge \neg x \vDash y$, but this is not derivable with unit propagation
    - φ is not propagation complete
- Compute propagation-complete representation of the constraint
  - I.e. can use unit propagation to deduce all logically entailed literals
  - E.g. $\varphi' = (x \vee y \vee z) \wedge (x \vee y \vee \neg z) \wedge (x \vee y)$ is propagation complete
    - $\varphi' \wedge \neg x \vDash y$ is derivable with unit propagation
    - $\varphi' \wedge \neg y \vDash x$ is derivable with unit propagation
- Make this representation compact. How?
  - Use empowering clauses

Outline

Outline

Notation

- Clause: c
  - E.g. $(x \vee y)$

Notation

- Clause: c
  - E.g. $(x \vee y)$
- CNF formula: φ
  - E.g. $(x \vee y) \wedge (x \vee \neg y)$

Notation

- Clause: c
  - E.g. $(x \vee y)$
- CNF formula: φ
  - E.g. $(x \vee y) \wedge (x \vee \neg y)$
- Entailment: $\models$
  - E.g. $(x \vee y) \wedge (x \vee \neg y) \models x$

Notation

- Clause: c
  - E.g. $(x \vee y)$
- CNF formula: φ
  - E.g. $(x \vee y) \wedge (x \vee \neg y)$
- Entailment: $\models$
  - E.g. $(x \vee y) \wedge (x \vee \neg y) \models x$
- φ inconsistent: $\varphi \models \bot$
  - E.g. $(x \vee y) \wedge (x \vee \neg y) \wedge (\neg x) \models \bot$

Notation

- Clause: c
  - E.g. $(x \vee y)$
- CNF formula: φ
  - E.g. $(x \vee y) \wedge (x \vee \neg y)$
- Entailment: $\models$
  - E.g. $(x \vee y) \wedge (x \vee \neg y) \models x$
- φ inconsistent: $\varphi \models \bot$
  - E.g. $(x \vee y) \wedge (x \vee \neg y) \wedge (\neg x) \models \bot$
- Unit propagation: $\vdash_1$
  - $(x \vee y) \wedge (x \vee \neg y) \wedge (\neg y) \vdash_1 x$
  - $(x \vee y) \wedge (x \vee \neg y) \wedge (\neg x) \vdash_1 \bot$

Propagation Completeness

- φ is propagation-complete if, for any set of literals $\{l_1, \ldots, l_k\} \neq \emptyset$,
  - If $\varphi \wedge l_1 \wedge \cdots \wedge l_k \models d$ then $\varphi \wedge l_1 \wedge \cdots \wedge l_k \vdash_1 d$
- And,
  - If $\varphi \wedge l_1 \wedge \cdots \wedge l_k \models \bot$ then $\varphi \wedge l_1 \wedge \cdots \wedge l_k \vdash_1 \bot$

Propagation Completeness

- φ is propagation-complete if, for any set of literals $\{l_1, \ldots, l_k\} \neq \emptyset$,
  - If $\varphi \wedge l_1 \wedge \cdots \wedge l_k \models d$ then $\varphi \wedge l_1 \wedge \cdots \wedge l_k \vdash_1 d$
- And,
  - If $\varphi \wedge l_1 \wedge \cdots \wedge l_k \models \bot$ then $\varphi \wedge l_1 \wedge \cdots \wedge l_k \vdash_1 \bot$

- $\varphi = (x \vee y) \wedge (x \vee \neg y)$ is propagation complete

Knowledge Compilation

- Compilation to achieve propagation completeness

Knowledge Compilation

- Compilation to achieve propagation completeness
- Given φ, compute a compiled formula φ_C such that,
  - φ_C equivalent to φ, $\varphi_C \equiv \varphi$
  - φ_C is propagation complete

Knowledge Compilation

- Compilation to achieve propagation completeness
- Given φ, compute a compiled formula φ_C such that,
  - φ_C equivalent to φ, $\varphi_C \equiv \varphi$
  - φ_C is propagation complete
- Equivalent to compilation for clausal entailment. Why?

Knowledge Compilation

- Compilation to achieve propagation completeness
- Given φ, compute a compiled formula φ_C such that,
  - φ_C equivalent to φ, $\varphi_C \equiv \varphi$
  - φ_C is propagation complete
- Equivalent to compilation for clausal entailment. Why?
  - To check whether $\varphi \models c$, with $c = (l_1 \vee \cdots \vee l_k)$, simply check whether $\varphi \wedge \neg l_1 \wedge \cdots \wedge \neg l_k \vdash_1 \bot$

Empowering Clauses

- Compilation by computing subset of formula's implicates

Empowering Clauses

- Compilation by computing subset of formula's implicates
- A clause c is empowering if,
  - $\varphi \models c$
  - $\varphi \wedge \bigwedge_{j \in 1..k,\ j \neq i} \neg l_j \quad \not\vdash_1 \quad \bot$; and
  - $\varphi \wedge \bigwedge_{j \in 1..k,\ j \neq i} \neg l_j \quad \not\vdash_1 \quad l_i$; where l_i is the empowering literal

Empowering Clauses

- Compilation by computing subset of formula's implicates
- A clause c is empowering if,
  - $\varphi \models c$
  - $\varphi \wedge \bigwedge_{j \in 1..k,\ j \neq i} \neg l_j \quad \not\vdash_1 \quad \bot$; and
  - $\varphi \wedge \bigwedge_{j \in 1..k,\ j \neq i} \neg l_j \quad \not\vdash_1 \quad l_i$; where l_i is the empowering literal
- Examples:
  - Given $\varphi = (x \vee z) \wedge (y \vee \neg z)$, $(x \vee y)$ is **not** empowering
    - $\varphi \wedge \neg x \vdash_1 y$
    - $\varphi \wedge \neg y \vdash_1 x$

Empowering Clauses

- Compilation by computing subset of formula's implicates
- A clause c is empowering if,
  - $\varphi \models c$
  - $\varphi \wedge \bigwedge_{j \in 1..k,\ j \neq i} \neg l_j \ \not\vdash_1 \ \bot$; and
  - $\varphi \wedge \bigwedge_{j \in 1..k,\ j \neq i} \neg l_j \ \not\vdash_1 \ l_i$; where l_i is the empowering literal
- Examples:
  - Given $\varphi = (x \vee z) \wedge (y \vee \neg z)$, $(x \vee y)$ is **not** empowering
    - $\varphi \wedge \neg x \vdash_1 y$
    - $\varphi \wedge \neg y \vdash_1 x$
  - Given $\varphi = (x \vee y \vee z) \wedge (x \vee y \vee \neg z)$, $(x \vee y)$ is empowering
    - $\varphi \models (x \vee y)$
    - $\varphi \wedge \neg y \not\vdash_1 \bot$
    - $\varphi \wedge \neg y \not\vdash_1 x$

Outline

Computing Empowering Implicates with QBF I

- Consider configurable clause $c = (l_1 \vee l_2 \vee \ldots \vee l_k)$, i.e. the target clause to be checked for the empowerment condition
  - Configuration of c quantified existentially

Computing Empowering Implicates with QBF I

- Consider configurable clause $c = (l_1 \vee l_2 \vee \ldots \vee l_k)$, i.e. the target clause to be checked for the empowerment condition
  - Configuration of c quantified existentially

- Validate conditions for clause c to be empowering, with empowering literal l_i
  - $\varphi \models c$
  - $\varphi \wedge \bigwedge_{j \in 1..k,\ j \neq i} \neg l_j \quad \not\models_1 \quad \bot$; and
  - $\varphi \wedge \bigwedge_{j \in 1..k,\ j \neq i} \neg l_j \quad \not\models_1 \quad l_i$

Computing Empowering Implicates with QBF I

- Consider configurable clause $c = (l_1 \vee l_2 \vee \ldots \vee l_k)$, i.e. the target clause to be checked for the empowerment condition
  - Configuration of c quantified existentially

- Validate conditions for clause c to be empowering, with empowering literal l_i
  - $\varphi \models c$
  - $\varphi \wedge \bigwedge_{j \in 1..k,\ j \neq i} \neg l_j \quad \not\models_1 \quad \bot$; and
  - $\varphi \wedge \bigwedge_{j \in 1..k,\ j \neq i} \neg l_j \quad \not\models_1 \quad l_i$

- 2QBF formulation, with $\exists\forall$ quantifier alternation

Computing Empowering Implicates with QBF II

- How to create a configurable clause?

Computing Empowering Implicates with QBF II

- How to create a configurable clause?
  - x_i represented as x_i^0, and $\neg x_i$ represented as x_i^1

Computing Empowering Implicates with QBF II

- How to create a configurable clause?
  - x_i represented as x_i^0, and $\neg x_i$ represented as x_i^1
  - A selector variable s_i^p indicates whether variable x_i contributes a literal with polarity p
    - $l_i^p = x_i^p \wedge s_i^p$, $1 \leq i \leq n$ and $p \in \{0, 1\}$, and $(\neg s_i^0 \vee \neg s_i^1)$

Computing Empowering Implicates with QBF II

- How to create a configurable clause?
  - x_i represented as x_i^0, and $\neg x_i$ represented as x_i^1
  - A selector variable s_i^p indicates whether variable x_i contributes a literal with polarity p
    - $l_i^p = x_i^p \wedge s_i^p$, $1 \leq i \leq n$ and $p \in \{0, 1\}$, and $(\neg s_i^0 \vee \neg s_i^1)$
  - Configurable clause c is defined as:

$$(l_1^0 \vee l_1^1 \vee l_2^0 \vee l_2^1 \vee \cdots \vee l_n^0 \vee l_n^1)$$

Computing Empowering Implicates with QBF II

- How to create a configurable clause?
  - x_i represented as x_i^0, and $\neg x_i$ represented as x_i^1
  - A selector variable s_i^p indicates whether variable x_i contributes a literal with polarity p
    - $l_i^p = x_i^p \wedge s_i^p$, $1 \leq i \leq n$ and $p \in \{0, 1\}$, and $(\neg s_i^0 \vee \neg s_i^1)$
  - Configurable clause c is defined as:

$$(l_1^0 \vee l_1^1 \vee l_2^0 \vee l_2^1 \vee \cdots \vee l_n^0 \vee l_n^1)$$

  - Selector variables $S = \{s_1^0, s_1^1, \ldots, s_n^0, s_n^1\}$ are quantified existentially

Computing Empowering Implicates with QBF III

- Define:
  - $\varphi^a = \varphi \cup \{\neg l : l \in c\}$; and
    - φ^a must be inconsistent, if c is an implicate of φ
  - $\varphi^c = \varphi^a \setminus \{\neg l_i\}$, given $l_i \in c$
    - φ^c should not be inconsistent and should not imply l_i

Computing Empowering Implicates with QBF III

- Define:
  - $\varphi^a = \varphi \cup \{\neg l : l \in c\}$; and
    - φ^a must be inconsistent, if c is an implicate of φ
  - $\varphi^c = \varphi^a \setminus \{\neg l_i\}$, given $l_i \in c$
    - φ^c should not be inconsistent and should not imply l_i

- The template of the QBF formulation becomes:

$$\exists_S \exists_{l_i \in c} . (\varphi \models c) \wedge (\varphi^c \not\models_1 \bot) \wedge (\varphi^c \not\models_1 l_i)$$

Computing Empowering Implicates with QBF III

- Define:
  - $\varphi^a = \varphi \cup \{\neg l : l \in c\}$; and
    - φ^a must be inconsistent, if c is an implicate of φ
  - $\varphi^c = \varphi^a \setminus \{\neg l_i\}$, given $l_i \in c$
    - φ^c should not be inconsistent and should not imply l_i

- The template of the QBF formulation becomes:

$$\exists_S \exists_{l_i \in c}. (\varphi \models c) \wedge (\varphi^c \not\models_1 \bot) \wedge (\varphi^c \not\models_1 l_i)$$

- Can be refined to:

$$\exists_S \exists_{l_i \in c}. \mathsf{Unsat}(\varphi^a) \wedge \mathsf{ProperUPConfig}(\varphi^c) \wedge \mathsf{NonUPImplied}(\varphi^c, l_i)$$

Computing Empowering Implicates with QBF IV

- Unsat(φ^a)
  - Represented with $\forall_X . (\neg\varphi^a(X))$

- ProperUPConfig(φ^c) **&** NonUPImplied(φ^c, l_i)
  - Must represent validate unit propagations, and check for these conditions (technical details in the paper)

Outline

Computing Empowering Clauses I

- Can use QBF formulation to compute empowering clauses
- But, some empowering clauses can become non-empowering
  - $\varphi = (a \vee b) \wedge (\neg a \vee x) \wedge (\neg a \vee y) \wedge (\neg b \vee x) \wedge (\neg b \vee y) \wedge (\neg x \vee y)$

Computing Empowering Clauses I

- Can use QBF formulation to compute empowering clauses
- But, some empowering clauses can become non-empowering
  - $\varphi = (a \vee b) \wedge (\neg a \vee x) \wedge (\neg a \vee y) \wedge (\neg b \vee x) \wedge (\neg b \vee y) \wedge (\neg x \vee y)$
  - Compilation sequence: $[\varphi; (y); (x)]$

Computing Empowering Clauses I

- Can use QBF formulation to compute empowering clauses
- But, some empowering clauses can become non-empowering
  - $\varphi = (a \vee b) \wedge (\neg a \vee x) \wedge (\neg a \vee y) \wedge (\neg b \vee x) \wedge (\neg b \vee y) \wedge (\neg x \vee y)$
  - Compilation sequence: $[\varphi; (y); (x)]$
  - When added, (y) is empowering. Why?

Computing Empowering Clauses I

- Can use QBF formulation to compute empowering clauses
- But, some empowering clauses can become non-empowering
  - $\varphi = (a \vee b) \wedge (\neg a \vee x) \wedge (\neg a \vee y) \wedge (\neg b \vee x) \wedge (\neg b \vee y) \wedge (\neg x \vee y)$
  - Compilation sequence: $[\varphi; (y); (x)]$
  - When added, (y) is empowering. Why?
    - $\varphi \models y$
    - $\varphi \not\vdash_1 \bot$
    - $\varphi \not\vdash_1 y$

Computing Empowering Clauses I

- Can use QBF formulation to compute empowering clauses
- But, some empowering clauses can become non-empowering
  - $\varphi = (a \vee b) \wedge (\neg a \vee x) \wedge (\neg a \vee y) \wedge (\neg b \vee x) \wedge (\neg b \vee y) \wedge (\neg x \vee y)$
  - Compilation sequence: $[\varphi; (y); (x)]$
  - When added, (y) is empowering. Why?
    - $\varphi \models y$
    - $\varphi \not\vdash_1 \bot$
    - $\varphi \not\vdash_1 y$
  - When added, (x) is empowering. Why?

Computing Empowering Clauses I

- Can use QBF formulation to compute empowering clauses
- But, some empowering clauses can become non-empowering
  - $\varphi = (a \vee b) \wedge (\neg a \vee x) \wedge (\neg a \vee y) \wedge (\neg b \vee x) \wedge (\neg b \vee y) \wedge (\neg x \vee y)$
  - Compilation sequence: $[\varphi; (y); (x)]$
  - When added, (y) is empowering. Why?
    - $\varphi \models y$
    - $\varphi \not\vdash_1 \bot$
    - $\varphi \not\vdash_1 y$
  - When added, (x) is empowering. Why?
    - $\varphi \wedge (y) \models x$
    - $\varphi \wedge (y) \not\vdash_1 \bot$
    - $\varphi \wedge (y) \not\vdash_1 x$

Computing Empowering Clauses I

- Can use QBF formulation to compute empowering clauses
- But, some empowering clauses can become non-empowering
  - $\varphi = (a \vee b) \wedge (\neg a \vee x) \wedge (\neg a \vee y) \wedge (\neg b \vee x) \wedge (\neg b \vee y) \wedge (\neg x \vee y)$
  - Compilation sequence: $[\varphi; (y); (x)]$
  - When added, (y) is empowering. Why?
    - $\varphi \models y$
    - $\varphi \not\vdash_1 \bot$
    - $\varphi \not\vdash_1 y$
  - When added, (x) is empowering. Why?
    - $\varphi \wedge (y) \models x$
    - $\varphi \wedge (y) \not\vdash_1 \bot$
    - $\varphi \wedge (y) \not\vdash_1 x$
  - After adding (x), (y) becomes non-empowering. Why?

Computing Empowering Clauses I

- Can use QBF formulation to compute empowering clauses
- But, some empowering clauses can become non-empowering
  - $\varphi = (a \vee b) \wedge (\neg a \vee x) \wedge (\neg a \vee y) \wedge (\neg b \vee x) \wedge (\neg b \vee y) \wedge (\neg x \vee y)$
  - Compilation sequence: $[\varphi; (y); (x)]$
  - When added, (y) is empowering. Why?
    - $\varphi \models y$
    - $\varphi \not\vdash_1 \bot$
    - $\varphi \not\vdash_1 y$
  - When added, (x) is empowering. Why?
    - $\varphi \wedge (y) \models x$
    - $\varphi \wedge (y) \not\vdash_1 \bot$
    - $\varphi \wedge (y) \not\vdash_1 x$
  - After adding (x), (y) becomes non-empowering. Why?
    - $\varphi \wedge (x) \vdash_1 y$

Computing Empowering Clauses I

- Can use QBF formulation to compute empowering clauses
- But, some empowering clauses can become non-empowering
  - $\varphi = (a \vee b) \wedge (\neg a \vee x) \wedge (\neg a \vee y) \wedge (\neg b \vee x) \wedge (\neg b \vee y) \wedge (\neg x \vee y)$
  - Compilation sequence: $[\varphi; (y); (x)]$
  - When added, (y) is empowering. Why?
    - $\varphi \models y$
    - $\varphi \not\vdash_1 \bot$
    - $\varphi \not\vdash_1 y$
  - When added, (x) is empowering. Why?
    - $\varphi \wedge (y) \models x$
    - $\varphi \wedge (y) \not\vdash_1 \bot$
    - $\varphi \wedge (y) \not\vdash_1 x$
  - After adding (x), (y) becomes non-empowering. Why?
    - $\varphi \wedge (x) \vdash_1 y$
- It is possible to generate exponentially many empowering clauses that become deprecated (i.e. non-empowering)

Computing Empowering Clauses II

- Claim:
  In a length-increasing compilation sequence, a clause of length k never deprecates a clause of length $j < k$

Computing Empowering Clauses II

- Claim:
  In a length-increasing compilation sequence, a clause of length k never deprecates a clause of length $j < k$

- Approach:
  Compute empowering implicates by increasing size
  - Easy to adapt 2QBF formulation

Computing Empowering Clauses II

- Claim:
  In a length-increasing compilation sequence, a clause of length k never deprecates a clause of length $j < k$
- Approach:
  Compute empowering implicates by increasing size
  - Easy to adapt 2QBF formulation
- Remark:
  There can still be deprecated clauses of size k due to other clauses of size k

Outline

Properties of Compilation with Empowerment I

- Main claims:

Properties of Compilation with Empowerment I

- Main claims:
  - All implicates computed in a length-increasing compilation sequence are prime

Properties of Compilation with Empowerment I

- Main claims:
  - All implicates computed in a length-increasing compilation sequence are prime
  - There are formulas with exponentially more prime implicates than empowering implicates

Properties of Compilation with Empowerment I

- Main claims:
  - All implicates computed in a length-increasing compilation sequence are prime
  - There are formulas with exponentially more prime implicates than empowering implicates
  - There are formulas with exponentially more prime implicates generated with merge-resolution than empowering implicates

Properties of Compilation with Empowerment II

- All implicates computed in a length-increasing compilation sequence are prime
  - Compilation sequence $[\varphi; c_1; \cdots; c_k]$

Properties of Compilation with Empowerment II

- All implicates computed in a length-increasing compilation sequence are prime
  - Compilation sequence $[\varphi; c_1; \cdots; c_k]$
  - Assume c_j not prime

Properties of Compilation with Empowerment II

- All implicates computed in a length-increasing compilation sequence are prime
  - Compilation sequence $[\varphi; c_1; \cdots; c_k]$
  - Assume c_j not prime
  - Then, there exists c_i that subsumes c_j
    - $|c_i| < |c_j|$ and $i < j$
    - I.e. $[\varphi; c_1; \cdots; c_i; \cdots c_j; \cdots c_k]$

Properties of Compilation with Empowerment II

- All implicates computed in a length-increasing compilation sequence are prime
  - Compilation sequence $[\varphi; c_1; \cdots; c_k]$
  - Assume c_j not prime
  - Then, there exists c_i that subsumes c_j
    - $|c_i| < |c_j|$ and $i < j$
    - I.e. $[\varphi; c_1; \cdots; c_i; \cdots c_j; \cdots c_k]$
  - But, c_j cannot be empowering
    - Whenever c_j becomes unit, c_i either becomes unit or becomes inconsistent

Properties of Compilation with Empowerment II

- There are formulas with exponentially more prime implicates than empowering implicates

Properties of Compilation with Empowerment II

- There are formulas with exponentially more prime implicates than empowering implicates
  - Consider EVEN_n formulas

$$
\begin{array}{lll}
 & (\neg x_1 \vee y_1) \wedge (\neg y_1 \vee x_1) & \text{"}y_1 = x_1\text{"} \\
\wedge & \bigwedge_{i \in 2..n} \left(\begin{array}{l} (\neg y_i \vee y_{i-1} \vee x_i) \wedge \\ (\neg y_i \vee \neg y_{i-1} \vee \neg x_i) \wedge \\ (y_i \vee \neg y_{i-1} \vee x_i) \wedge \\ (y_i \vee y_{i-1} \vee \neg x_i) \end{array} \right) & \text{"}y_i = y_{i-1} \oplus x_i\text{"} \\
\wedge & (\neg y_n) & \text{"}\textit{output gate } y_n \textit{ is false}\text{"}
\end{array}
$$

Properties of Compilation with Empowerment II

- There are formulas with exponentially more prime implicates than empowering implicates
  - Consider EVEN_n formulas

$$
\begin{array}{lll}
 & (\neg x_1 \vee y_1) \wedge (\neg y_1 \vee x_1) & \text{"}y_1 = x_1\text{"} \\
\wedge & \bigwedge_{i \in 2..n} \left(\begin{array}{l} (\neg y_i \vee y_{i-1} \vee x_i) \wedge \\ (\neg y_i \vee \neg y_{i-1} \vee \neg x_i) \wedge \\ (y_i \vee \neg y_{i-1} \vee x_i) \wedge \\ (y_i \vee y_{i-1} \vee \neg x_i) \end{array} \right) & \text{"}y_i = y_{i-1} \oplus x_i\text{"} \\
\wedge & (\neg y_n) & \text{"}\textit{output gate } y_n \textit{ is false}\text{"}
\end{array}
$$

  - There are no empowering implicates (see proof details in paper)

Properties of Compilation with Empowerment II

- There are formulas with exponentially more prime implicates than empowering implicates
  - Consider EVEN_n formulas

$$
\begin{array}{lll}
 & (\neg x_1 \vee y_1) \wedge (\neg y_1 \vee x_1) & \text{"}y_1 = x_1\text{"} \\
\wedge & \bigwedge_{i \in 2..n} \left(\begin{array}{l} (\neg y_i \vee y_{i-1} \vee x_i) \wedge \\ (\neg y_i \vee \neg y_{i-1} \vee \neg x_i) \wedge \\ (y_i \vee \neg y_{i-1} \vee x_i) \wedge \\ (y_i \vee y_{i-1} \vee \neg x_i) \end{array} \right) & \text{"}y_i = y_{i-1} \oplus x_i\text{"} \\
\wedge & (\neg y_n) & \text{"}\textit{output gate } y_n \textit{ is false}\text{"}
\end{array}
$$

  - There are no empowering implicates (see proof details in paper)
  - But there are many prime implicates
    - Any clause over the variables $x_1, \cdots, x_n$ with an odd number of negative literals is an implicate
    - These implicates are prime
    - There are 2^{n-1} such clauses

Outline

Conclusions & Research Directions

- Focus on propagation-complete knowledge compilation
- Compilation with empowering implicates
- Exponentially more compact than well-known implicate-based compilation approaches

- Usefulness in practice?
  - Too hard to generate empowring implicates?
  - CNF encoding of CP constraints
- Study other forms of propagation-complete compilation

Thank You