Automated Synthesis: Towards the Holy Grail of AI

Kuldeep S. Meel[1], Supratik Chakraborty[2], S Akshay[2], Priyanka Golia[1,3], Subhajit Roy[3]

[1]National University of Singapore
[2]Indian Institute of Technology Bombay
[3]Indian Institute of Technology Kanpur

IJCAI-2022

The Life of Computer Engineers since middle ages

middle ages:= aka second half of 20th century

Wish I had a **system** that could work like this ...

Wish I had a **system** that could work like this ...

Specification by examples

X_1	X_2	Y
20	3	20
2	9	10
5	30	30
$\vdots$	$\vdots$	$\vdots$

The Life of Computer Engineers since middle ages

middle ages:= aka second half of 20th century

Wish I had a **system** that could work like this ...

Specification by examples

X_1	X_2	Y
20	3	20
2	9	10
5	30	30
⋮	⋮	⋮

X_1, X_2 →  → Y

Specification by logical relation

$$(Y \geq X_1) \wedge (Y \geq X_2) \wedge (Y \geq 10) \wedge$$
$$((Y \leq X_1) \vee (Y \leq X_2) \vee (Y \leq 10))$$

The Life of Computer Engineers since middle ages

middle ages:= aka second half of 20th century

Wish I had a **system** that could work like this ...

Specification by examples

X_1	X_2	Y
20	3	20
2	9	10
5	30	30
⋮	⋮	⋮

Specification by logical relation

$$(Y \geq X_1) \wedge (Y \geq X_2) \wedge (Y \geq 10) \wedge$$
$$((Y \leq X_1) \vee (Y \leq X_2) \vee (Y \leq 10))$$

Specification in natural language

Output Y as max of X_1 and X_2, but if both are less than 10, then output Y as 10

After some effort ...

Specification by logical relation

$$(Y \geq X_1) \wedge (Y \geq X_2) \wedge (Y \geq 10) \wedge$$
$$((Y \leq X_1) \vee (Y \leq X_2) \vee (Y \leq 10))$$

The Life of Computer Engineers since Middle Ages

middle ages:= aka second half of 20th century

After some effort ...

```
input X1, X2;
temp := max(X1, X2);
if (temp < 10) Y := 10;
else Y := temp;
output Y;
```

Specification by logical relation

$$(Y \geq X_1) \wedge (Y \geq X_2) \wedge (Y \geq 10) \wedge$$
$$((Y \leq X_1) \vee (Y \leq X_2) \vee (Y \leq 10))$$

The Life of Computer Engineers since Middle Ages

middle ages:= aka second half of 20th century

After some effort ...

```
input X1, X2;
temp := max(X1, X2);
if (temp < 10) Y := 10;
else Y := temp;
output Y;
```

How do you know this is correct?

Specification by logical relation

$$(Y \geq X_1) \wedge (Y \geq X_2) \wedge (Y \geq 10) \wedge$$
$$((Y \leq X_1) \vee (Y \leq X_2) \vee (Y \leq 10))$$

A Vision for the New Age

Wish I had an **algorithm** that could help me ...

Specification by examples

X_1	X_2	Y
20	3	20
2	9	10
5	30	30
⋮	⋮	⋮

Specification by logical relation

$$(Y \geq X_1) \wedge (Y \geq X_2) \wedge (Y \geq 10) \wedge$$
$$((Y \leq X_1) \vee (Y \leq X_2) \vee (Y \leq 10))$$

Specification in natural language

Output Y as max of X_1 and X_2, but if both are less than 10, then output Y as 10

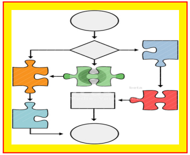
A Vision for the New Age

Wish I had an **algorithm**
that could help me ...

Specification by examples

X_1	X_2	Y
20	3	20
2	9	10
5	30	30
⋮	⋮	⋮

Synthesis Algorithm



Specification by logical relation

$$(Y \geq X_1) \wedge (Y \geq X_2) \wedge (Y \geq 10) \wedge$$
$$((Y \leq X_1) \vee (Y \leq X_2) \vee (Y \leq 10))$$

Specification in natural language

Output Y as max of X_1 and X_2, but if both are less than 10, then output Y as 10

A Vision for the New Age

Wish I had an **algorithm**
that could help me ...

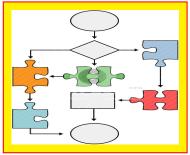
X_1, X_2 →  → Y

Provably correct system

Specification by examples

X_1	X_2	Y
20	3	20
2	9	10
5	30	30
⋮	⋮	⋮

Synthesis Algorithm



Specification by logical relation

$$(Y \geq X_1) \wedge (Y \geq X_2) \wedge (Y \geq 10) \wedge$$
$$((Y \leq X_1) \vee (Y \leq X_2) \vee (Y \leq 10))$$

Specification in natural language

Output Y as max of X_1 and X_2, but if both are less than 10, then output Y as 10

Wish I had an **algorithm** that could help me ...

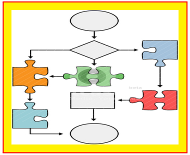
X_1, X_2 →  → Y

Provably correct system **again!**

Specification by examples

X_1	X_2	Y
20	3	30
2	9	12
5	30	30
⋮	⋮	⋮

Synthesis Algorithm



Specification by logical relation

$$(Y \geq X_1 + 10) \wedge (Y \geq X_2) \wedge ((Y \leq X_1 + 10) \vee (Y \leq X_2))$$

Specification in natural language

Output Y as X_2 if it is at least 10 more than X_1, otherwise output $X_1 + 10$

Focus of this talk

Wish I had an algorithm
that could help me ...

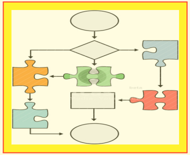
Specification by examples

X_1	X_2	Y
20	3	20
2	9	10
5	30	30
⋮	⋮	⋮

Synthesis Algorithm

X_1, X_2 →  → Y

Provably correct system



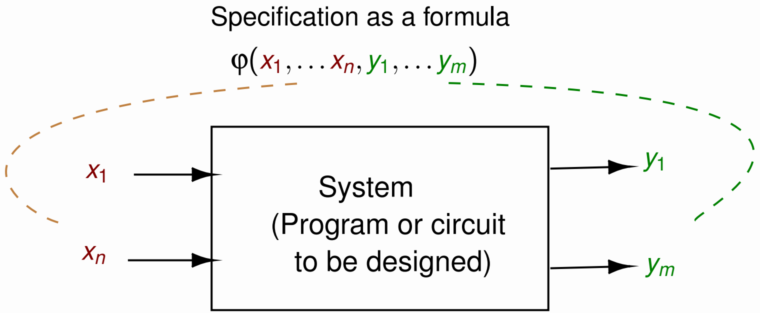
Specification by logical relation

$$(Y \geq X_1) \wedge (Y \geq X_2) \wedge (Y \geq 0) \wedge$$
$$((Y \leq X_1) \vee (Y \leq X_2) \vee (Y \leq 0))$$

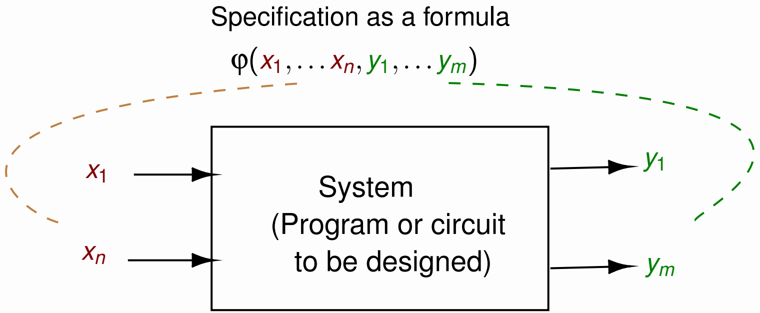
Specification in natural language

Output Y as max of X_1 and X_2, but if both are less than 10, then output Y as 10

Automated Functional Synthesis: A Generic View

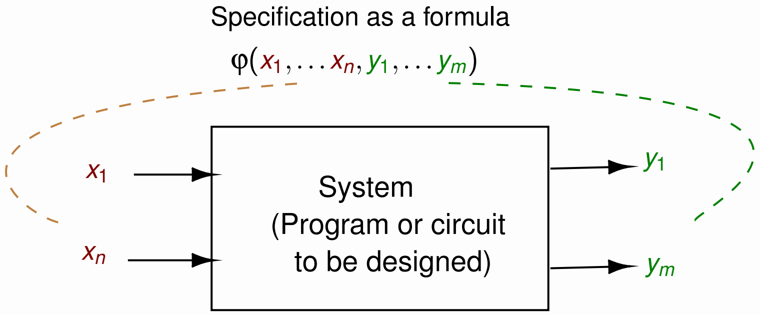


Automated Functional Synthesis: A Generic View



- Goal: Automatically synthesize system s.t. it satisfies $\varphi(x_1,..,x_n,y_1,..,y_m)$ whenever possible
  - x_i *input* variables (vector $\boldsymbol{X}$)
  - y_j *output* variables (vector $\boldsymbol{Y}$)

Automated Functional Synthesis: A Generic View

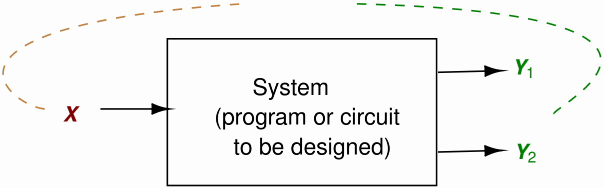


- Goal: Automatically synthesize system s.t. it satisfies $\varphi(x_1, .., x_n, y_1, .., y_m)$ whenever possible
  - x_i *input* variables (vector $\boldsymbol{X}$)
  - y_j *output* variables (vector $\boldsymbol{Y}$)
- Need $\boldsymbol{Y}$ as functions $\boldsymbol{F}$ of $\boldsymbol{X}$ such that $\varphi(\boldsymbol{X}, \boldsymbol{F})$ is satisfied.

Example: Cryptanalysis

Specification as bit-vector formula

$$(X = Y_1 \times_{[n]} Y_2) \wedge \neg(Y_1 = 1_{[n]}) \wedge \neg(Y_2 = 1_{[n]})$$

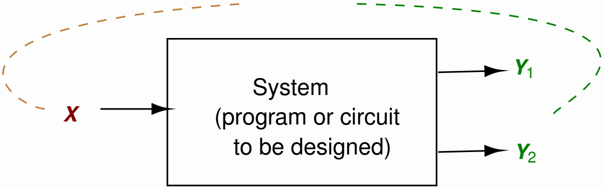


- Synthesize Y_1, Y_2 as functions of X

Example: Cryptanalysis

Specification as bit-vector formula

$$(X = Y_1 \times_{[n]} Y_2) \wedge \neg(Y_1 = 1_{[n]}) \wedge \neg(Y_2 = 1_{[n]})$$

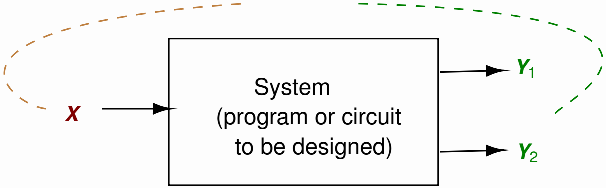


- Synthesize Y_1, Y_2 as functions of X
  - Factorization: Y_1, Y_2 must be non-trivial factors of X

Example: Cryptanalysis

Specification as bit-vector formula

$$(X = Y_1 \times_{[n]} Y_2) \wedge \neg(Y_1 = 1_{[n]}) \wedge \neg(Y_2 = 1_{[n]})$$

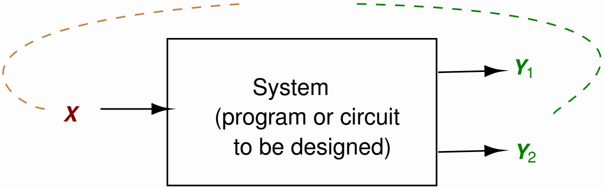


- Synthesize Y_1, Y_2 as functions of X
  - Factorization: Y_1, Y_2 must be non-trivial factors of X
  - Efficient solution would break crypto systems

Example: Cryptanalysis

Specification as bit-vector formula

$$(X = Y_1 \times_{[n]} Y_2) \wedge \neg(Y_1 = 1_{[n]}) \wedge \neg(Y_2 = 1_{[n]})$$

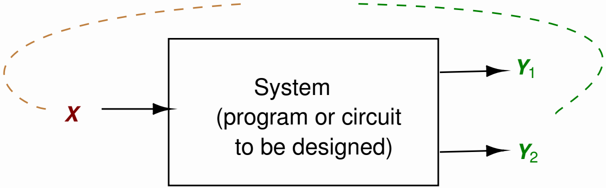


- Synthesize Y_1, Y_2 as functions of X
  - Factorization: Y_1, Y_2 must be non-trivial factors of X
  - Efficient solution would break crypto systems
- Is this spec always satisfiable?

Example: Cryptanalysis

Specification as bit-vector formula

$$(X = Y_1 \times_{[n]} Y_2) \land \neg(Y_1 = 1_{[n]}) \land \neg(Y_2 = 1_{[n]})$$

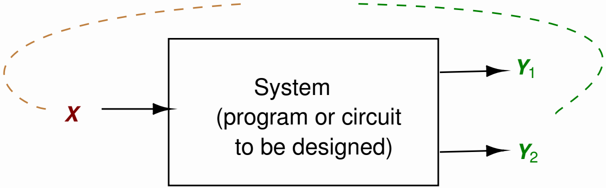


- Synthesize Y_1, Y_2 as functions of X
  - Factorization: Y_1, Y_2 must be non-trivial factors of X
  - Efficient solution would break crypto systems
- Is this spec always satisfiable? (No, X can be prime.)

Example: Cryptanalysis

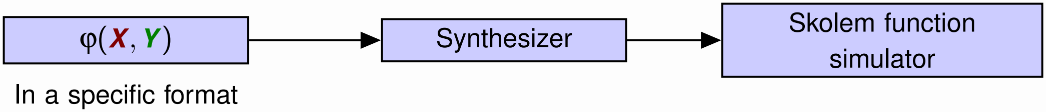
Specification as bit-vector formula

$$(X = Y_1 \times_{[n]} Y_2) \land \neg(Y_1 = 1_{[n]}) \land \neg(Y_2 = 1_{[n]})$$



- Synthesize Y_1, Y_2 as functions of X
  - Factorization: Y_1, Y_2 must be non-trivial factors of X
  - Efficient solution would break crypto systems
- Is this spec always satisfiable? (No, X can be prime.)
  - Synthesis still makes sense even if spec is NOT valid!
  - If X is prime, we don't care what we output
- Goal: Automatically synthesize system s.t. it satisfies $\varphi(x_1, .., x_n, y_1, .., y_m)$ **whenever possible.**

Functional Synthesis: Not Just an Abstract Dream



Boolean Functional Synthesis

- Goal: Automatically synthesize system s.t. it satisfies $\varphi(x_1, .., x_n, y_1, .., y_m)$ **whenever possible**.

Formal definition

Given Boolean relation $\varphi(x_1, .., x_n, y_1, .., y_m)$

- x_1 *input* variables (vector $\boldsymbol{X}$)
- y_j *output* variables (vector $\boldsymbol{Y}$)

- Goal: Automatically synthesize system s.t. it satisfies $\varphi(x_1,..,x_n,y_1,..,y_m)$ **whenever possible**.

Formal definition

Given Boolean relation $\varphi(x_1,..,x_n,y_1,..,y_m)$

- x_1 *input* variables (vector $\boldsymbol{X}$)
- y_j *output* variables (vector $\boldsymbol{Y}$)

Synthesize Boolean functions $F_j(\boldsymbol{X})$ for each y_j s.t.

$$\forall \boldsymbol{X} \left(\exists y_1 \ldots y_m \, \varphi(\boldsymbol{X}, y_1 \ldots y_m) \Leftrightarrow \varphi(\boldsymbol{X}, F_1(\boldsymbol{X}), \ldots F_m(\boldsymbol{X})) \right)$$

Boolean Functional Synthesis

- Goal: Automatically synthesize system s.t. it satisfies $\varphi(x_1, .., x_n, y_1, .., y_m)$ **whenever possible**.

Formal definition

Given Boolean relation $\varphi(x_1, .., x_n, y_1, .., y_m)$

- x_1 *input* variables (vector $\boldsymbol{X}$)
- y_j *output* variables (vector $\boldsymbol{Y}$)

Synthesize Boolean functions $F_j(\boldsymbol{X})$ for each y_j s.t.

$$\forall \boldsymbol{X} \left(\exists y_1 \ldots y_m \, \varphi(\boldsymbol{X}, y_1 \ldots y_m) \Leftrightarrow \varphi(\boldsymbol{X}, F_1(\boldsymbol{X}), \ldots F_m(\boldsymbol{X})) \right)$$

$F_j(\boldsymbol{X})$ is also called a *Skolem function* for y_j in φ.

Example

Let $\boldsymbol{X} = \{x_1, x_2\}$, $\boldsymbol{Y} = \{y_1\}$ and $\varphi(\boldsymbol{X}, \boldsymbol{Y}) = x_1 \vee x_2 \vee y_1$

Possible Skolem function: $F_1(x_1, x_2) := \neg(x_1 \vee x_2)$

Example

Let $\boldsymbol{X} = \{x_1, x_2\}$, $\boldsymbol{Y} = \{y_1\}$ and $\varphi(\boldsymbol{X}, \boldsymbol{Y}) = x_1 \vee x_2 \vee y_1$

Possible Skolem function: $F_1(x_1, x_2) := \neg(x_1 \vee x_2)$

$$\varphi(\boldsymbol{X}, F_1(\boldsymbol{X})) = x_1 \vee x_2 \vee (\neg(x_1 \vee x_2))$$

$\boldsymbol{X}$	$\exists \boldsymbol{Y} \varphi(\boldsymbol{X}, \boldsymbol{Y})$		$\varphi(\boldsymbol{X}, F_1(\boldsymbol{X}))$
$x_1 = 0, x_2 = 0$	$y_1 = 1$	True	True
$x_1 = 0, x_2 = 1$	$y_1 = 1$	True	True
$x_1 = 1, x_2 = 0$	$y_1 = 1$	True	True
$x_1 = 1, x_2 = 1$	$y_1 = 1$	True	True

$$\forall \boldsymbol{X} (\exists \boldsymbol{Y} \varphi(\boldsymbol{X}, \boldsymbol{Y}) \equiv \varphi(\boldsymbol{X}, F_1(\boldsymbol{X})))$$

Example

Let $\boldsymbol{X} = \{x_1, x_2\}$, $\boldsymbol{Y} = \{y_1\}$ and $\varphi(\boldsymbol{X}, \boldsymbol{Y}) = x_1 \vee x_2 \vee y_1$

Possible Skolem function: $F_1(x_1, x_2) := \neg(x_1 \vee x_2)$

$$\varphi(\boldsymbol{X}, F_1(\boldsymbol{X})) = x_1 \vee x_2 \vee (\neg(x_1 \vee x_2))$$

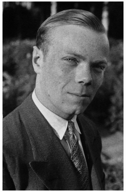
$\boldsymbol{X}$	$\exists \boldsymbol{Y} \varphi(\boldsymbol{X}, \boldsymbol{Y})$		$\varphi(\boldsymbol{X}, F_1(\boldsymbol{X}))$
$x_1 = 0, x_2 = 0$	$y_1 = 1$	True	True
$x_1 = 0, x_2 = 1$	$y_1 = 1$	True	True
$x_1 = 1, x_2 = 0$	$y_1 = 1$	True	True
$x_1 = 1, x_2 = 1$	$y_1 = 1$	True	True

$$\forall \boldsymbol{X} (\exists \boldsymbol{Y} \varphi(\boldsymbol{X}, \boldsymbol{Y}) \equiv \varphi(\boldsymbol{X}, F_1(\boldsymbol{X})))$$

Many possible Skolem functions:

$F_1(x_1, x_2) = \neg x_1$ $\quad F_1(x_1, x_2) = \neg x_2$ $\quad F_1(x_1, x_2) = 1$

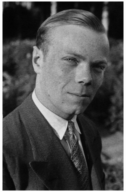
A storied history



Skolem functions play an important role in first order logic

- Getting rid of existential quantifiers
- Seminal work by Thoralf Skolem 1920s and Jacques Herbrand 1930s.
- Skolemization and "Skolem-Normal form"
- Focus on existence of form, NOT computability.

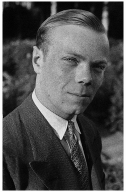
A storied history

Skolem functions play an important role in first order logic

- Getting rid of existential quantifiers
- Seminal work by Thoralf Skolem 1920s and Jacques Herbrand 1930s.
- Skolemization and “Skolem-Normal form”
- Focus on existence of form, NOT computability.

We can trace this history even further back

Skolem functions play an important role in first order logic

- Getting rid of existential quantifiers
- Seminal work by Thoralf Skolem 1920s and Jacques Herbrand 1930s.
- Skolemization and “Skolem-Normal form”
- Focus on existence of form, NOT computability.

We can trace this history even further back

- Existence and construction of Boolean unifiers
- Boole’1847, Lowenheim’1908.


Outline

First part: Applications and Overview

1. Application Domains
2. Theoretical hardness and a high level survey of algorithms

Outline

First part: Applications and Overview

1. Application Domains
2. Theoretical hardness and a high level survey of algorithms

Short break (5 minutes): Stretch yourselves!

Second part: Deep Dive into Recent Advances

3. Two Approaches
   - The Guess-check-and-Repair algorithmic paradigm
     - Counter-example guided and Data-driven approaches

   Coffee break
   - Knowledge representations for efficient synthesis
4. Tool demo
5. Conclusion and the Way Forward

Outline

1. Application Domains
2. Theoretical Hardness and Practical Algorithms
3. Deep Dives
4. Tool Demo
5. Conclusion and the Way Forward

Application Domain 1: Program Synthesis

Given a specification φ, automatically synthesize a program $\mathcal{P}$ such that $\mathcal{P} \models \varphi$.

* Alur et al.,FMCAD'13

Application Domain 1: Program Synthesis

Given a specification φ, automatically synthesize a program $\mathcal{P}$ such that $\mathcal{P} \models \varphi$.

Specifications

- Logical specifications
- Test cases (examples)
- Natural Language
- Demonstrations/Traces
- Programs

* Alur et al.,FMCAD'13

Application Domain 1: Program Synthesis

Given a specification φ, automatically synthesize a program $\mathcal{P}$ such that $\mathcal{P} \models \varphi$.

Specifications

- Logical specifications
- Test cases (examples)
- Natural Language
- Demonstrations/Traces
- Programs

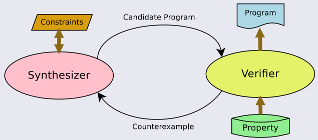
A popular approach: Syntax-Guided Synthesis (SyGuS)*

- a background theory (eg. theory of bit-vectors)
- a semantic correctness specification (in the background theory)
- a language to represent the synthesized program (as a context-free grammar)

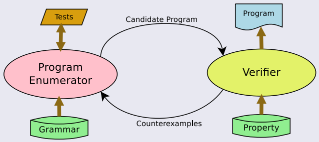
* Alur et al.,FMCAD'13

Application Domain 1: Algorithms for Program Synthesis *†

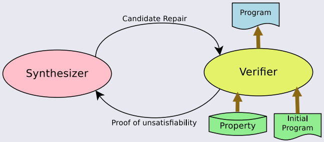
CEGIS (Symbolic)



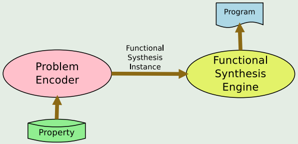
CEGIS (Enumerative)



SyPR: Proof-Guided Repairs



Reduction to Functional Synthesis



---

* CEGIS(Sym): Solar-Lezama, STTT'12. CEGIS(Enum): Alur et al.,

† FMCAD'13; Alur et al., TACAS'17; SyPR: Verma and Roy, ESEC/FSE'17;

Application Domain 1: Link to Boolean Functional Synthesis*

$g(x_1, x_2) \geq x_1$ and
$g(x_1, x_2) \geq x_2$ and
$(g(x_1, x_2) == x_1$ or
$g(x_1, x_2) == x_2)$

- Synthesize program representing function g that satisfies the specification.

* Golia et al., IJCAI'21

Application Domain 1: Link to Boolean Functional Synthesis*

$g(x_1, x_2) \geq x_1$ and
$g(x_1, x_2) \geq x_2$ and
$(g(x_1, x_2) == x_1$ or
$g(x_1, x_2) == x_2)$

$y_1 \geq x_1$ and
$y_1 \geq x_2$ and
$(y_1 == x_1$ or
$y_1 == x_1)$

- Synthesize program representing function g that satisfies the specification.
- Replace every call of functions g by a new variable y_1 in the specification.

$$\forall x_1, x_2 \, \exists y_1 \; \varphi(x_1, x_2, y_1)$$

---

* Golia et al., IJCAI'21

Application Domain 1: Link to Boolean Functional Synthesis*

$g(x_1, x_2) \geq x_1$ and
$g(x_1, x_2) \geq x_2$ and
$(g(x_1, x_2) == x_1$ or
$g(x_1, x_2) == x_2)$

$y_1 \geq x_1$ and
$y_1 \geq x_2$ and
$(y_1 == x_1$ or
$y_1 == x_1)$

- Synthesize program representing function g that satisfies the specification.
- Replace every call of functions g by a new variable y_1 in the specification.
- Works with appropriate caveats, e.g., outputs depend on all inputs.

$$\forall x_1, x_2 \, \exists y_1 \; \varphi(x_1, x_2, y_1)$$

The synthesized skolem function is an implementation of the function $g(x_1, x_2)$.

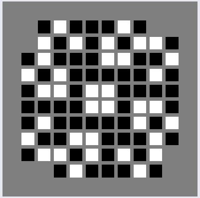
* Golia et al., IJCAI'21

Conway's Game of Life

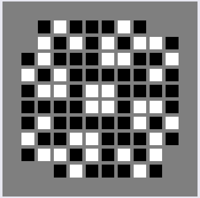
- Infinite 2D grid of cells, each alive or dead in each gen:
    1. (Under-pop) live cell with <2 live neighbors dies;
    2. (Status-quo) live cell with 2 or 3 live neighbors lives;
    3. (Over-pop) live cell >3 live neighbors dies;
    4. (Re-birth) dead cell with 3 live neighbors comes alive

Conway's Game of Life

- Infinite 2D grid of cells, each alive or dead in each gen:
  1. (Under-pop) live cell with < 2 live neighbors dies;
  2. (Status-quo) live cell with 2 or 3 live neighbors lives;
  3. (Over-pop) live cell > 3 live neighbors dies;
  4. (Re-birth) dead cell with 3 live neighbors comes alive

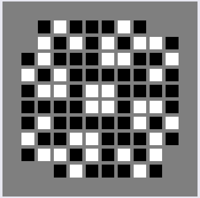


Conway's Game of Life



- Infinite 2D grid of cells, each alive or dead in each gen:
  1. (Under-pop) live cell with <2 live neighbors dies;
  2. (Status-quo) live cell with 2 or 3 live neighbors lives;
  3. (Over-pop) live cell >3 live neighbors dies;
  4. (Re-birth) dead cell with 3 live neighbors comes alive
- Objective: Is there a Garden of Eden (GoE), a configuration with no predecessor?
  - If it does not exist, give a witnessing function that defines the predecessor!

Conway's Game of Life



- Infinite 2D grid of cells, each alive or dead in each gen:
  1. (Under-pop) live cell with < 2 live neighbors dies;
  2. (Status-quo) live cell with 2 or 3 live neighbors lives;
  3. (Over-pop) live cell > 3 live neighbors dies;
  4. (Re-birth) dead cell with 3 live neighbors comes alive
- Objective: Is there a Garden of Eden (GoE), a configuration with no predecessor?
  - If it does not exist, give a witnessing function that defines the predecessor!
  - History from 1971 onwards...

    (https://conwaylife.com/wiki/Garden_of_Eden)

Application Domain 2: Games and planning

Encoded as Skolem function existence and synthesis problem

- Let $\boldsymbol{X}$ be current position, $\boldsymbol{Y}$ be previous position and $T(\boldsymbol{X}, \boldsymbol{Y})$ be transition function
- Then GoE does not exist iff $\forall \boldsymbol{X} \exists \boldsymbol{Y}\ T(\boldsymbol{X}, \boldsymbol{Y})$ is satisfiable!

Application Domain 2: Games and planning

Encoded as Skolem function existence and synthesis problem

- Let $\boldsymbol{X}$ be current position, $\boldsymbol{Y}$ be previous position and $T(\boldsymbol{X}, \boldsymbol{Y})$ be transition function
- Then GoE does not exist iff $\forall \boldsymbol{X} \exists \boldsymbol{Y}\ T(\boldsymbol{X}, \boldsymbol{Y})$ is satisfiable!
- A witness that GoE does not exist is a Skolem function for $\boldsymbol{Y}$.

Application Domain 2: Games and planning

Encoded as Skolem function existence and synthesis problem

- Let $\boldsymbol{X}$ be current position, $\boldsymbol{Y}$ be previous position and $T(\boldsymbol{X}, \boldsymbol{Y})$ be transition function
- Then GoE does not exist iff $\forall \boldsymbol{X} \exists \boldsymbol{Y}\ T(\boldsymbol{X}, \boldsymbol{Y})$ is satisfiable!
- A witness that GoE does not exist is a Skolem function for $\boldsymbol{Y}$.

- $\forall \boldsymbol{X} \exists \boldsymbol{Y}\ T(\boldsymbol{X}, \boldsymbol{Y})$ has two alternating blocks of quantifiers: 2-QBF. In general, can have many!

Application Domain 2: Games and planning

Encoded as Skolem function existence and synthesis problem

- Let $\boldsymbol{X}$ be current position, $\boldsymbol{Y}$ be previous position and $T(\boldsymbol{X}, \boldsymbol{Y})$ be transition function
- Then GoE does not exist iff $\forall \boldsymbol{X} \exists \boldsymbol{Y}\ T(\boldsymbol{X}, \boldsymbol{Y})$ is satisfiable!
- A witness that GoE does not exist is a Skolem function for $\boldsymbol{Y}$.

- $\forall \boldsymbol{X} \exists \boldsymbol{Y}\ T(\boldsymbol{X}, \boldsymbol{Y})$ has two alternating blocks of quantifiers: 2-QBF. In general, can have many!

Quantified Boolean Formula (QBF) or QSAT: Essentially SAT + chunks of quantifiers

$$\forall \boldsymbol{X}_1 \exists \boldsymbol{Y}_1 \forall \boldsymbol{X}_2 \exists \boldsymbol{Y}_2 \ldots \forall \boldsymbol{X}_k \exists \boldsymbol{Y}_k \varphi$$

where φ is a Quantifier-free Boolean Formula, $\boldsymbol{X}_i$, $\boldsymbol{Y}_i$ are sequences of variables.

Application Domain 2: Games and planning

Encoded as Skolem function existence and synthesis problem

- Let $\boldsymbol{X}$ be current position, $\boldsymbol{Y}$ be previous position and $T(\boldsymbol{X}, \boldsymbol{Y})$ be transition function
- Then GoE does not exist iff $\forall \boldsymbol{X} \exists \boldsymbol{Y}\ T(\boldsymbol{X}, \boldsymbol{Y})$ is satisfiable!
- A witness that GoE does not exist is a Skolem function for $\boldsymbol{Y}$.

- $\forall \boldsymbol{X} \exists \boldsymbol{Y}\ T(\boldsymbol{X}, \boldsymbol{Y})$ has two alternating blocks of quantifiers: 2-QBF. In general, can have many!

Quantified Boolean Formula (QBF) or QSAT: Essentially SAT + chunks of quantifiers

$$\forall \boldsymbol{X}_1 \exists \boldsymbol{Y}_1 \forall \boldsymbol{X}_2 \exists \boldsymbol{Y}_2 \ldots \forall \boldsymbol{X}_k \exists \boldsymbol{Y}_k \varphi$$

where φ is a Quantifier-free Boolean Formula, $\boldsymbol{X}_i$, $\boldsymbol{Y}_i$ are sequences of variables.

Any 2-player game can be coded as QBF—Skolem functions are winning strategies of Player 2 ($\exists$-player)!

Other applications

- Quantifier elimination (Of course!)
  - $\exists \boldsymbol{Y} \varphi(\boldsymbol{X}, \boldsymbol{Y}) \equiv \varphi(\boldsymbol{X}, \boldsymbol{F}(\boldsymbol{X}))$ used in fundamental operations like image computation, interpolant generation, computing predicate abstractions etc.

Other applications

- Quantifier elimination (Of course!)
  - $\exists Y \varphi(X, Y) \equiv \varphi(X, F(X))$ used in fundamental operations like image computation, interpolant generation, computing predicate abstractions etc.
- Synthesizing arithmetic functions from specifications of arithmetic relations Fried et al.'16
  - Example: subtract, min, max, floor of avg, sort.

Other applications

- Quantifier elimination (Of course!)
  - $\exists Y \varphi(X, Y) \equiv \varphi(X, F(X))$ used in fundamental operations like image computation, interpolant generation, computing predicate abstractions etc.
- Synthesizing arithmetic functions from specifications of arithmetic relations Fried et al.'16
  - Example: subtract, min, max, floor of avg, sort.
- Disjunctive decomposition of transition relations Trivedi'03

Other applications

- Quantifier elimination (Of course!)
  - $\exists Y \varphi(X, Y) \equiv \varphi(X, F(X))$ used in fundamental operations like image computation, interpolant generation, computing predicate abstractions etc.
- Synthesizing arithmetic functions from specifications of arithmetic relations Fried et al.'16
  - Example: subtract, min, max, floor of avg, sort.
- Disjunctive decomposition of transition relations Trivedi'03
- Circuit repair Gitina et al.'13, Jiang et al.'20, Fujita et al.'20
  - Complete the implementation of a circuit such that it is functionally equivalent to the specification.
- Reactive synthesis
  - Synthesizing winning strategy within the winning region.

Outline

First part: Applications and Overview

1. Application Domains
2. Theoretical hardness and a high level survey of algorithms

Outline

First part: Applications and Overview

1. Application Domains
2. Theoretical hardness and a high level survey of algorithms

Short break (5 minutes): Stretch yourselves!

Second part: Deep Dive into Recent Advances

3. Two Approaches
   - The Guess-check-and-Repair algorithmic paradigm
     - Counter-example guided and Data-driven approaches

   Coffee break
   - Knowledge representations for efficient synthesis
4. Tool demo
5. Conclusion and the Way Forward

Outline

1. Application Domains
2. Theoretical Hardness and Practical Algorithms
3. Deep Dives
4. Tool Demo
5. Conclusion and the Way Forward

Representation: Specification & Skolem functions as Boolean circuits in NNF.

---

* S. Akshay, Supratik Chakraborty, Shubham Goel, Sumith Kulal, Shetal Shah, CAV'18, FMSD'20

How Hard is Boolean Functional Synthesis?

Representation: Specification & Skolem functions as Boolean circuits in NNF.

Time complexity

Boolean functional synthesis is *NP*-hard

* S. Akshay, Supratik Chakraborty, Shubham Goel, Sumith Kulal, Shetal Shah, CAV'18, FMSD'20

How Hard is Boolean Functional Synthesis?

Representation: Specification & Skolem functions as Boolean circuits in NNF.

Time complexity

Boolean functional synthesis is *NP*-hard (not surprising!).

---

* S. Akshay, Supratik Chakraborty, Shubham Goel, Sumith Kulal, Shetal Shah, CAV'18, FMSD'20

How Hard is Boolean Functional Synthesis?

Representation: Specification & Skolem functions as Boolean circuits in NNF.

Time complexity

Boolean functional synthesis is *NP*-hard (not surprising!).

Space complexity *

* S. Akshay, Supratik Chakraborty, Shubham Goel, Sumith Kulal, Shetal Shah, CAV'18, FMSD'20

How Hard is Boolean Functional Synthesis?

Representation: Specification & Skolem functions as Boolean circuits in NNF.

Time complexity

Boolean functional synthesis is *NP*-hard (not surprising!).

Space complexity *

- Unless some well-regarded complexity-theoretic conjectures fail, there exist specifications φ for which Skolem function sizes must be super-polynomial or even exponential in $|\varphi|$.

---

* S. Akshay, Supratik Chakraborty, Shubham Goel, Sumith Kulal, Shetal Shah, CAV'18, FMSD'20

How Hard is Boolean Functional Synthesis?

Representation: Specification & Skolem functions as Boolean circuits in NNF.

Time complexity

Boolean functional synthesis is *NP*-hard (not surprising!).

Space complexity *

- Unless some well-regarded complexity-theoretic conjectures fail, there exist specifications φ for which Skolem function sizes must be super-polynomial or even exponential in $|\varphi|$.

Bottomline: Efficient algorithms for Boolean functional synthesis unlikely

* S. Akshay, Supratik Chakraborty, Shubham Goel, Sumith Kulal, Shetal Shah, CAV'18, FMSD'20

How Hard is Boolean Functional Synthesis?

Representation: Specification & Skolem functions as Boolean circuits in NNF.

Time complexity

Boolean functional synthesis is *NP*-hard (not surprising!).

Space complexity *

- Unless some well-regarded complexity-theoretic conjectures fail, there exist specifications φ for which Skolem function sizes must be super-polynomial or even exponential in $|\varphi|$.

Bottomline: Efficient algorithms for Boolean functional synthesis unlikely

Also note: use of SAT-solvers inevitable or unavoidable!

* S. Akshay, Supratik Chakraborty, Shubham Goel, Sumith Kulal, Shetal Shah, CAV'18, FMSD'20

A Survey of Existing Techniques

Phase I

1. Extract Skolem functions from proof of validity of $\forall \boldsymbol{X} \exists \boldsymbol{Y} \varphi(\boldsymbol{X}, \boldsymbol{Y})$
   Bendetti'05, Jussilla et al.'07, Balabanov et al.'12, Heule et al.'14
   – Efficient if a short proof of validity is found.
2. Using templates
   Solar-Lezama et al.'06, Srivastava et al.'13
   – Effective when small set of candidate Skolem functions known.
3. Self-substitution + function composition
   Jiang'09, Trivedi'03
   – Craig Interpolation-based approach.

Existing Approaches (Cont.)

Phase II

4. Incremental determinization
   Rabe et al.'17,'18
   - Incrementally adds new constraints to the formula to generate a unique Skolem function.
5. Quantifier instantiation techniques in SMT solvers
   Barrett et al.'15, Bierre et al.'17
   - Works even for bit-vector and other theories.
6. Input/output component separation
   Chakraborty et al.'18
   - View specification as made of input and output components.
   - Alternate analysis of each component to generate decision lists.
7. Synthesis from and as ROBDDs
   - Kukula et al.'00, Kuncak et al.'10, Fried et al.'16, Tabajara et al.'17

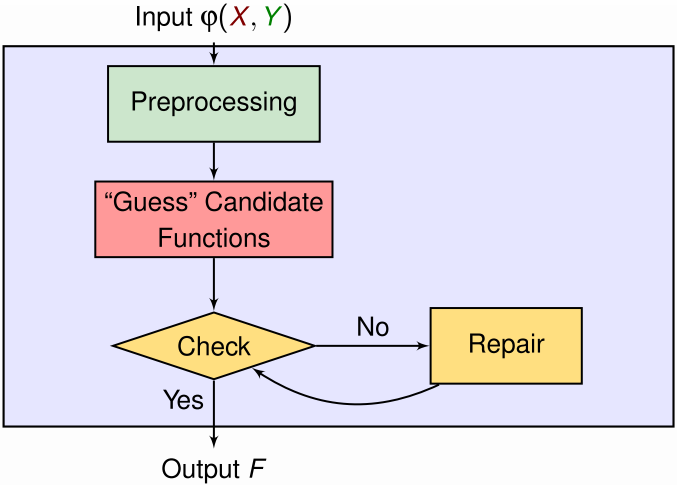
Existing Approaches (Cont.)

Phase III: The Modern Age!

8. Counter-example guided Skolem function generation (Guess + check + repair)
   - Over-approximate initial guess of Skolem functions + refine
     John et al.'15, Akshay et al.'17,'18,'20
   - Machine-learn initial Skolem function + MaxSat-based iterative repair
     Golia et al.'20, '21
9. Knowledge Compilation for Boolean Functional Synthesis (Special normal forms)
   - Synthesis negation normal forms (SynNNF)
     Akshay et al.'19
   - Subset-And-Unsatisfiable Normal Form (SAUNF)
     Shah et al.'21

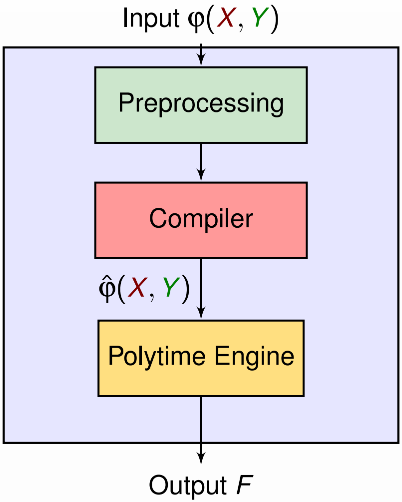
Our focus in the deep-dive: These last approaches!

Counter-example guided Skolem function generation



Machine-learning based
or
Function-approx based

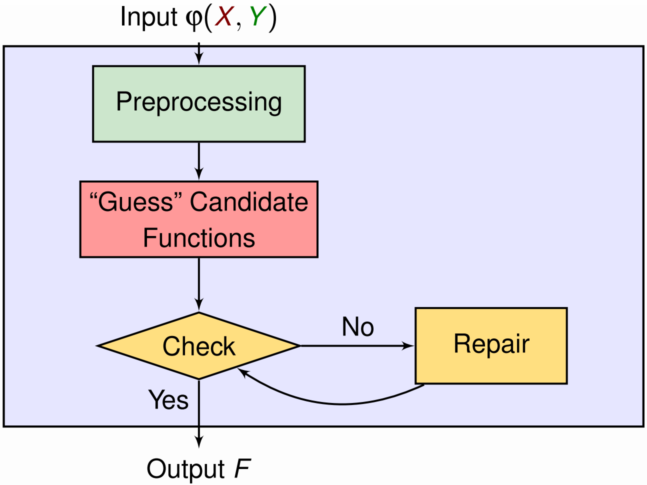
Synthesis via special normal forms



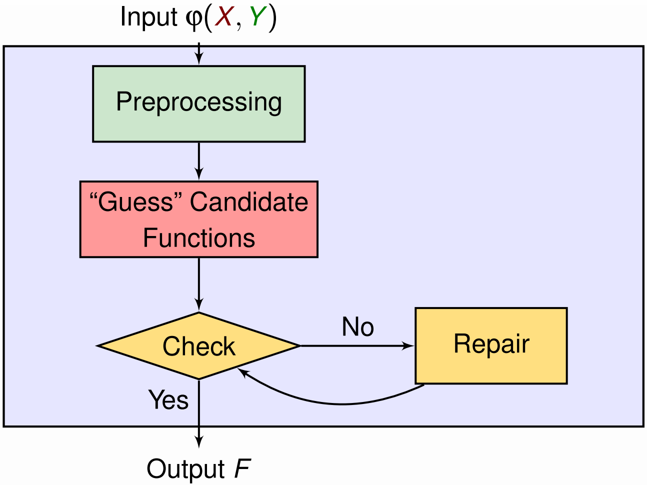
Outline

1. Application Domains
2. Theoretical Hardness and Practical Algorithms
3. Deep Dives
4. Tool Demo
5. Conclusion and the Way Forward

Deep Dive 1: Counter-example guided Skolem function generation

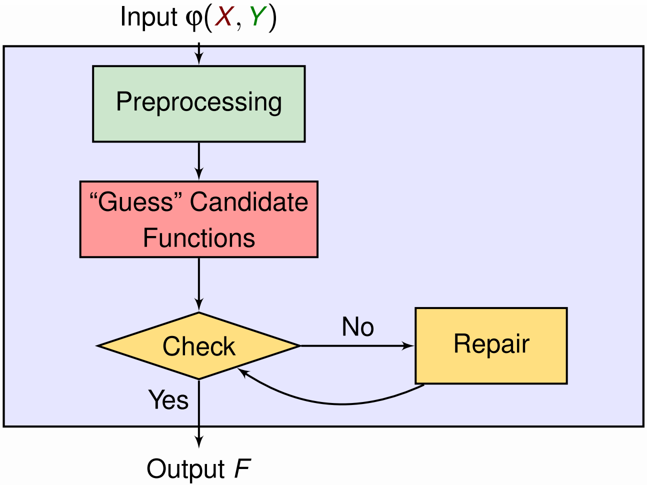


Deep Dive 1: Counter-example guided Skolem function generation



Simple but effective!

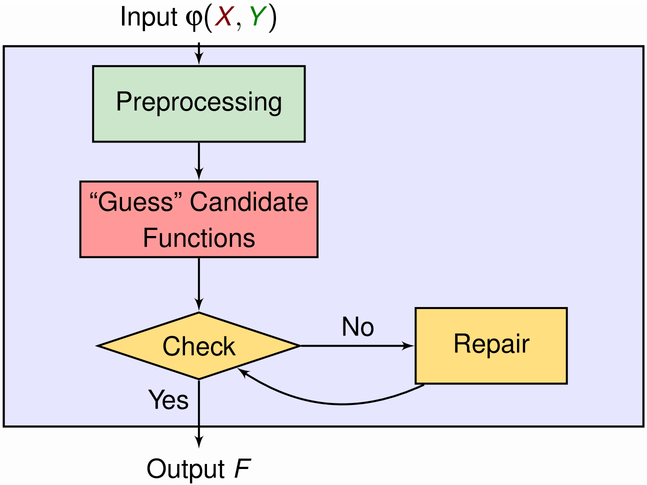
Deep Dive 1: Counter-example guided Skolem function generation



Simple but effective!

Formal methods

Deep Dive 1: Counter-example guided Skolem function generation

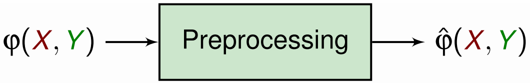


Simple but effective!

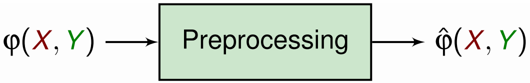
Machine-learning (Manthan)
or
Function-approx (BFSS)

Formal methods

The Preprocessing Module



- Skolem functions of $\hat{\varphi}(X, Y)$
  - are (or can be extended to) Skolem functions for $\varphi(X, Y)$.
  - are easier to synthesise at least for some variables.

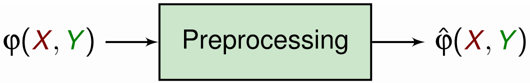


- Skolem functions of $\hat{\varphi}(X, Y)$
  - are (or can be extended to) Skolem functions for $\varphi(X, Y)$.
  - are easier to synthesise at least for some variables.

Pre-process your input

- For unate variables, constant functions suffice. e.g., if $\varphi|_{y=0} \implies \varphi|_{y=1}$ then $F_1 = 1$.

The Preprocessing Module

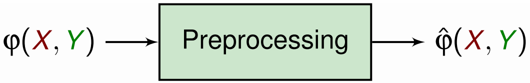


- Skolem functions of $\hat{\varphi}(X, Y)$
  - are (or can be extended to) Skolem functions for $\varphi(X, Y)$.
  - are easier to synthesise at least for some variables.

Pre-process your input

- For unate variables, constant functions suffice. e.g., if $\varphi|_{y=0} \implies \varphi|_{y=1}$ then $F_1 = 1$.
- Uniquely defined variables are easy, e.g., Tseitin variables. y_1 is uniquely defined in

$$\varphi(\boldsymbol{X}, \boldsymbol{Y}) := \ldots \wedge (y_1 \leftrightarrow (x_1 \vee x_2)) \wedge \ldots$$



- Skolem functions of $\hat{\varphi}(X, Y)$
  - are (or can be extended to) Skolem functions for $\varphi(X, Y)$.
  - are easier to synthesise at least for some variables.

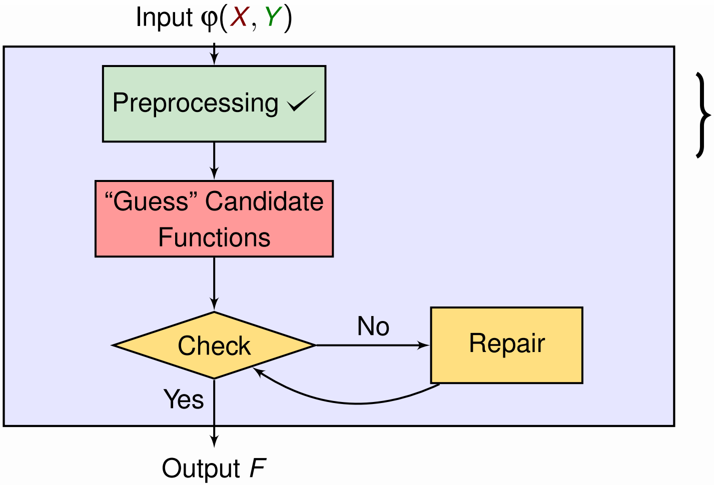
Pre-process your input

- For unate variables, constant functions suffice. e.g., if $\varphi|_{y=0} \implies \varphi|_{y=1}$ then $F_1 = 1$.
- Uniquely defined variables are easy, e.g., Tseitin variables. y_1 is uniquely defined in

$$\varphi(\boldsymbol{X}, \boldsymbol{Y}) := \ldots \wedge (y_1 \leftrightarrow (x_1 \vee x_2)) \wedge \ldots$$

These simple checks are surprisingly effective; handle many variables.

Deep Dive 1: Counter-example guided Skolem function generation



Simple but effective!

The Check Module

How do we check if a given function is a correct Skolem function?

Given functions $F_1, \ldots F_m$, is $\forall \mathbf{X} \big(\exists \mathbf{Y} \varphi(\mathbf{X}, \mathbf{Y}) \Leftrightarrow \varphi(\mathbf{X}, \mathbf{F}(\mathbf{X})) \big)$?

How do we check if a given function is a correct Skolem function?

Given functions $F_1, \ldots F_m$, is $\forall \mathbf{X} \left(\exists \mathbf{Y} \varphi(\mathbf{X}, \mathbf{Y}) \Leftrightarrow \varphi(\mathbf{X}, \mathbf{F}(\mathbf{X}) \right)$?
Can we avoid using a 2-QBF solver and stick to faster SAT-solvers?

The Check Module

How do we check if a given function is a correct Skolem function?

Given functions $F_1, \ldots F_m$, is $\forall \boldsymbol{X} \left(\exists \boldsymbol{Y} \varphi(\boldsymbol{X}, \boldsymbol{Y}) \Leftrightarrow \varphi(\boldsymbol{X}, \boldsymbol{F}(\boldsymbol{X})) \right)$?

Can we avoid using a 2-QBF solver and stick to faster SAT-solvers?

Yes, we can! [John et al.'15]

- Propositional error formula:

$$E(X, Y, Y') := \varphi(X, Y) \wedge \neg\varphi(X, Y') \wedge (Y' \leftrightarrow F(X))$$

The Check Module

How do we check if a given function is a correct Skolem function?

Given functions $F_1, \ldots F_m$, is $\forall \boldsymbol{X} \big(\exists \boldsymbol{Y} \varphi(\boldsymbol{X}, \boldsymbol{Y}) \Leftrightarrow \varphi(\boldsymbol{X}, \boldsymbol{F}(\boldsymbol{X})) \big)$?
Can we avoid using a 2-QBF solver and stick to faster SAT-solvers?

Yes, we can! [John et al.'15]

- Propositional error formula:

$$E(X, Y, Y') := \varphi(X, Y) \wedge \neg\varphi(X, Y') \wedge (Y' \leftrightarrow F(X))$$

- Suppose σ: satisfying assignment of E

How do we check if a given function is a correct Skolem function?

Given functions $F_1, \ldots F_m$, is $\forall \boldsymbol{X} \big(\exists \boldsymbol{Y} \varphi(\boldsymbol{X}, \boldsymbol{Y}) \Leftrightarrow \varphi(\boldsymbol{X}, \boldsymbol{F}(\boldsymbol{X})) \big)$?
Can we avoid using a 2-QBF solver and stick to faster SAT-solvers?

Yes, we can! [John et al.'15]

- Propositional error formula:

$$E(X, Y, Y') := \varphi(X, Y) \wedge \neg\varphi(X, Y') \wedge (Y' \leftrightarrow F(X))$$

- Suppose σ: satisfying assignment of E
  - $\varphi(\sigma[\boldsymbol{X}], \sigma[\boldsymbol{Y}]) = 1, \quad \sigma[Y'] = \boldsymbol{F}(\sigma[\boldsymbol{X}]), \quad \varphi(\sigma[\boldsymbol{X}], \sigma[(Y')]) = 0$

The Check Module

How do we check if a given function is a correct Skolem function?

Given functions $F_1, \ldots F_m$, is $\forall \boldsymbol{X} \big(\exists \boldsymbol{Y} \varphi(\boldsymbol{X}, \boldsymbol{Y}) \Leftrightarrow \varphi(\boldsymbol{X}, \boldsymbol{F}(\boldsymbol{X})) \big)$?
Can we avoid using a 2-QBF solver and stick to faster SAT-solvers?

Yes, we can! [John et al.'15]

- Propositional error formula:

$$E(X, Y, Y') := \varphi(X, Y) \wedge \neg\varphi(X, Y') \wedge (Y' \leftrightarrow F(X))$$

- Suppose σ: satisfying assignment of E
  - $\varphi(\sigma[\boldsymbol{X}], \sigma[\boldsymbol{Y}]) = 1, \quad \sigma[Y'] = \boldsymbol{F}(\sigma[\boldsymbol{X}]), \quad \varphi(\sigma[\boldsymbol{X}], \sigma[(Y')]) = 0$
  - σ is **counterexample** to the claim that $F_1, \ldots F_m$ are all correct Skolem functions.

The Check Module

How do we check if a given function is a correct Skolem function?

Given functions $F_1, \ldots F_m$, is $\forall \mathbf{X}\big(\, \exists \mathbf{Y} \varphi(\mathbf{X}, \mathbf{Y}) \;\Leftrightarrow\; \varphi(\mathbf{X}, \mathbf{F}(\mathbf{X}))\,\big)$?
Can we avoid using a 2-QBF solver and stick to faster SAT-solvers?

Yes, we can! [John et al.'15]

- Propositional error formula:

$$E(X, Y, Y') := \varphi(X, Y) \wedge \neg\varphi(X, Y') \wedge (Y' \leftrightarrow F(X))$$

- Suppose σ: satisfying assignment of E
  - $\varphi(\sigma[\mathbf{X}], \sigma[\mathbf{Y}]) = 1, \quad \sigma[Y'] = \mathbf{F}(\sigma[\mathbf{X}]), \quad \varphi(\sigma[\mathbf{X}], \sigma[(Y')]) = 0$
  - σ is **counterexample** to the claim that $F_1, \ldots F_m$ are all correct Skolem functions.
- E unsatisfiable iff $F_1, \ldots F_m$ are all correct Skolem functions.

The Repair Module

$$E(X, Y, Y') := \varphi(X, Y) \wedge \neg\varphi(X, Y') \wedge (Y' \leftrightarrow F(X))$$

- Let $\sigma := \{x_1 \mapsto 1, x_2 \mapsto 1, y_1 \mapsto 1, y_2 \mapsto 1, y'_1 \mapsto 0, y'_2 \mapsto 0\}$ be a counter-example.

What to Repair?

- Idea: Repair all F_i where $\sigma[y_i] \neq \sigma[y'_i]$.

The Repair Module

$$E(X, Y, Y') := \varphi(X, Y) \wedge \neg\varphi(X, Y') \wedge (Y' \leftrightarrow F(X))$$

- Let $\sigma := \{x_1 \mapsto 1, x_2 \mapsto 1, y_1 \mapsto 1, y_2 \mapsto 1, y'_1 \mapsto 0, y'_2 \mapsto 0\}$ be a counter-example.

What to Repair?

- Idea: Repair all F_i where $\sigma[y_i] \neq \sigma[y'_i]$.
- But $\varphi(X, Y)$ is Boolean Relation, say $\hat{\sigma} = \{x_1 \mapsto 1, x_2 \mapsto 1, y_1 \mapsto 0, y_2 \mapsto 1, y'_1 \mapsto 0, y'_2 \mapsto 0\}$
  - In this case, we don't need to repair F_1.

$$E(X, Y, Y') := \varphi(X, Y) \land \neg\varphi(X, Y') \land (Y' \leftrightarrow F(X))$$

- Let $\sigma := \{x_1 \mapsto 1, x_2 \mapsto 1, y_1 \mapsto 1, y_2 \mapsto 1, y'_1 \mapsto 0, y'_2 \mapsto 0\}$ be a counter-example.

What to Repair?

- Idea: Repair all F_i where $\sigma[y_i] \neq \sigma[y'_i]$.
- But $\varphi(X, Y)$ is Boolean Relation, say $\hat{\sigma} = \{x_1 \mapsto 1, x_2 \mapsto 1, y_1 \mapsto 0, y_2 \mapsto 1, y'_1 \mapsto 0, y'_2 \mapsto 0\}$
  - In this case, we don't need to repair F_1.
- Improvement: MaxSAT-based Identification of *nice counterexamples*
  - Hard Clauses $\varphi(X, Y) \land (X \leftrightarrow \sigma[X])$; Soft Clauses $(Y \leftrightarrow \sigma[Y'])$.

The Repair Module

$$E(X,Y,Y') := \varphi(X,Y) \wedge \neg\varphi(X,Y') \wedge (Y' \leftrightarrow F(X))$$

- Let $\sigma := \{x_1 \mapsto 1, x_2 \mapsto 1, y_1 \mapsto 1, y_2 \mapsto 1, y_1' \mapsto 0, y_2' \mapsto 0\}$ be a counter-example.

What to Repair?

- Idea: Repair all F_i where $\sigma[y_i] \neq \sigma[y_i']$.
- But $\varphi(X,Y)$ is Boolean Relation, say $\hat{\sigma} = \{x_1 \mapsto 1, x_2 \mapsto 1, y_1 \mapsto 0, y_2 \mapsto 1, y_1' \mapsto 0, y_2' \mapsto 0\}$
  - In this case, we don't need to repair F_1.
- Improvement: MaxSAT-based Identification of *nice counterexamples*
  - Hard Clauses $\varphi(X,Y) \wedge (X \leftrightarrow \sigma[X])$; Soft Clauses $(Y \leftrightarrow \sigma[Y'])$.

How to Repair?

- For improved cex $\hat{\sigma}$, we want to repair F_2.

The Repair Module

$$E(X, Y, Y') := \varphi(X, Y) \wedge \neg\varphi(X, Y') \wedge (Y' \leftrightarrow F(X))$$

- Let $\sigma := \{x_1 \mapsto 1, x_2 \mapsto 1, y_1 \mapsto 1, y_2 \mapsto 1, y'_1 \mapsto 0, y'_2 \mapsto 0\}$ be a counter-example.

What to Repair?

- Idea: Repair all F_i where $\sigma[y_i] \neq \sigma[y'_i]$.
- But $\varphi(X, Y)$ is Boolean Relation, say $\hat{\sigma} = \{x_1 \mapsto 1, x_2 \mapsto 1, y_1 \mapsto 0, y_2 \mapsto 1, y'_1 \mapsto 0, y'_2 \mapsto 0\}$
  - In this case, we don't need to repair F_1.
- Improvement: MaxSAT-based Identification of *nice counterexamples*
  - Hard Clauses $\varphi(X, Y) \wedge (X \leftrightarrow \sigma[X])$; Soft Clauses $(Y \leftrightarrow \sigma[Y'])$.

How to Repair?

- For improved cex $\hat{\sigma}$, we want to repair F_2. Idea: From $\hat{\sigma}$: if $x_1 \wedge x_2 \wedge \neg y_1$, then set $y_2 = 1$.

The Repair Module

$$E(X, Y, Y') := \varphi(X, Y) \wedge \neg\varphi(X, Y') \wedge (Y' \leftrightarrow F(X))$$

- Let $\sigma := \{x_1 \mapsto 1, x_2 \mapsto 1, y_1 \mapsto 1, y_2 \mapsto 1, y'_1 \mapsto 0, y'_2 \mapsto 0\}$ be a counter-example.

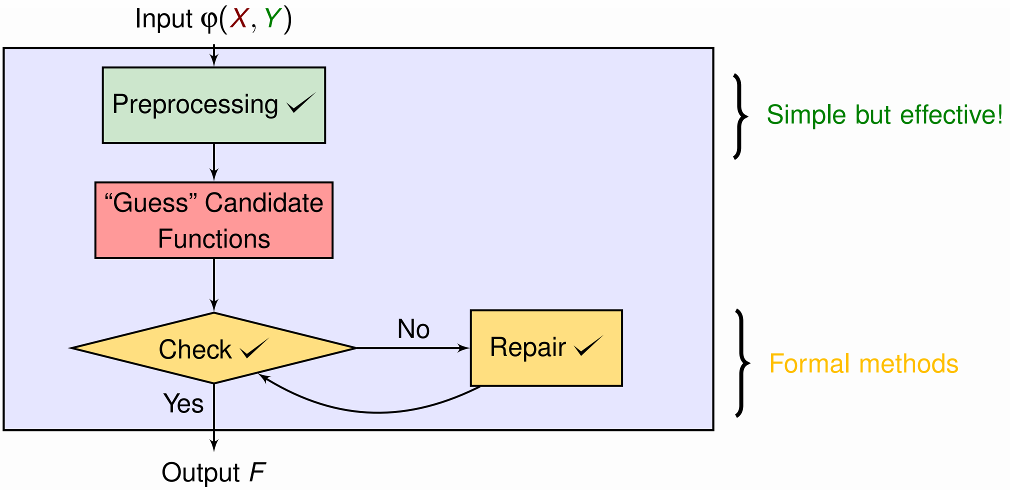
What to Repair?

- Idea: Repair all F_i where $\sigma[y_i] \neq \sigma[y'_i]$.
- But $\varphi(X, Y)$ is Boolean Relation, say $\hat{\sigma} = \{x_1 \mapsto 1, x_2 \mapsto 1, y_1 \mapsto 0, y_2 \mapsto 1, y'_1 \mapsto 0, y'_2 \mapsto 0\}$
  - In this case, we don't need to repair F_1.
- Improvement: MaxSAT-based Identification of *nice counterexamples*
  - Hard Clauses $\varphi(X, Y) \wedge (X \leftrightarrow \sigma[X])$; Soft Clauses $(Y \leftrightarrow \sigma[Y'])$.

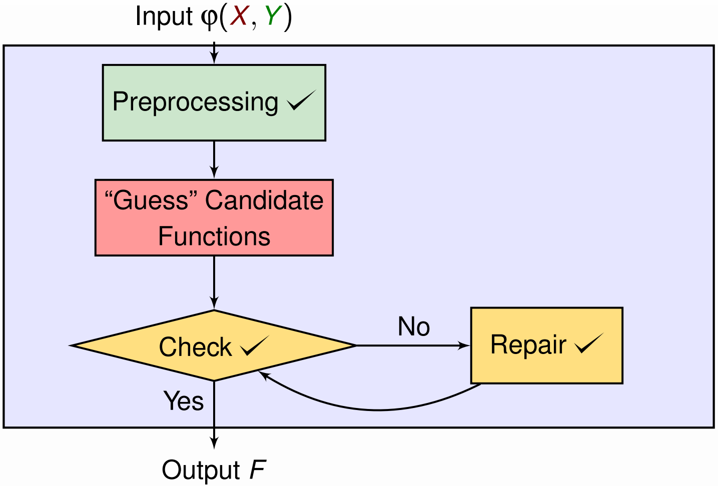
How to Repair?

- For improved cex $\hat{\sigma}$, we want to repair F_2. Idea: From $\hat{\sigma}$: if $x_1 \wedge x_2 \wedge \neg y_1$, then set $y_2 = 1$.
- Improvement: Use UNSAT Core of $\varphi(X, Y) \wedge x_1 \wedge x_2 \wedge \neg y_1 \wedge \neg y_2$.

Deep Dive 1: Counter-example guided Skolem function generation



Deep Dive 1: Counter-example guided Skolem function generation

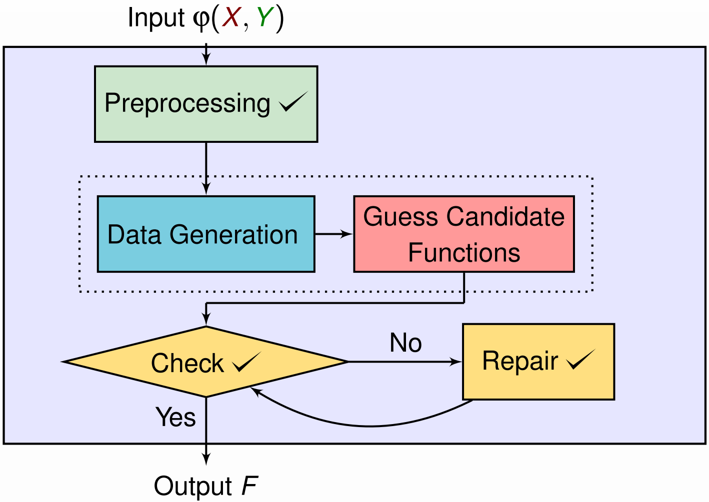


Simple but effective!

Machine-learning based

Formal methods

Machine-learning based guessing of candidate Skolem functions (Manthan)



Simple but effective!

Machine-learning based

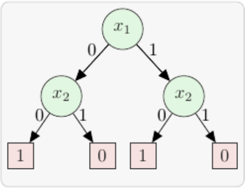
Formal methods

$\varphi(x_1, x_2, y_1, y_2)$ $\longrightarrow$

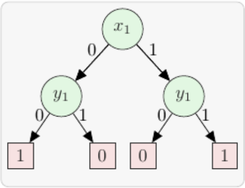
x_1	x_2	y_1	y_2
0	0	1	0
0	1	0	1
1	0	1	1
1	1	0	0

Learn Candidate Functions

x_1	x_2	y_1	y_2
0	0	1	0
0	1	0	1
1	0	1	1
1	1	0	0



$p_1 := (\neg x_1 \wedge \neg x_2)$,
$p_2 := (x_1 \wedge \neg x_2)$
f_1 = if p_1 then 1
elif p_2 then 1
else 0



$p_1 := (\neg x_1 \wedge \neg y_1)$,
$p_2 := (x_1 \wedge y_1)$
f_1 = if p_1 then 1
elif p_2 then 1
else 0

Data Generation

Potential Strategy: Randomly sample satisfying assignment of $\varphi(X, Y)$.

Challenge: Multiple valuations of y_1, y_2 for same valuation of x_1, x_2.

Data Generation

Potential Strategy: Randomly sample satisfying assignment of $\varphi(X, Y)$.

Challenge: Multiple valuations of y_1, y_2 for same valuation of x_1, x_2.

$$\varphi(x_1, x_2, y_1, y_2) : (x_1 \vee x_2 \vee y_1) \wedge (\neg x_1 \vee \neg x_2 \vee \neg y_2)$$

x_1	x_2	y_1	y_2
0	0	1	0/1
0	1	0/1	0/1
1	0	0/1	0/1
1	1	0/1	0

$$\varphi(x_1, x_2, y_1, y_2) : (x_1 \vee x_2 \vee y_1) \wedge (\neg x_1 \vee \neg x_2 \vee \neg y_2)$$

x_1	x_2	y_1	y_2
0	0	1	0/1
0	1	0/1	0/1
1	0	0/1	0/1
1	1	0/1	0

Uniform Sampler →

x_1	x_2	y_1	y_2
0	0	1	1
0	1	0	1
1	0	0	1
1	1	0	0

Data Generation

$$\varphi(x_1, x_2, y_1, y_2) : (x_1 \vee x_2 \vee y_1) \wedge (\neg x_1 \vee \neg x_2 \vee \neg y_2)$$

x_1	x_2	y_1	y_2
0	0	1	0/1
0	1	0/1	0/1
1	0	0/1	0/1
1	1	0/1	0

Uniform Sampler →

x_1	x_2	y_1	y_2
0	0	1	1
0	1	0	1
1	0	0	1
1	1	0	0

- Possible Skolem functions:
  - $F_1(x_1, x_2) = \neg(x_1 \vee x_2)$
  - $F_2(x_1, x_2) = \neg(x_1 \wedge x_2)$

Data Generation

$$\varphi(x_1, x_2, y_1, y_2) : (x_1 \vee x_2 \vee y_1) \wedge (\neg x_1 \vee \neg x_2 \vee \neg y_2)$$

x_1	x_2	y_1	y_2
0	0	1	0/1
0	1	0/1	0/1
1	0	0/1	0/1
1	1	0/1	0

Uniform Sampler →

x_1	x_2	y_1	y_2
0	0	1	1
0	1	0	1
1	0	0	1
1	1	0	0

- Possible Skolem functions:
  - $F_1(x_1, x_2) = \neg(x_1 \vee x_2)$ $F_1(x_1, x_2) = \neg x_1$ $F_1(x_1, x_2) = \neg x_2$ $F_1(x_1, x_2) = 1$
  - $F_2(x_1, x_2) = \neg(x_1 \wedge x_2)$ $F_2(x_1, x_2) = \neg x_1$ $F_2(x_1, x_2) = \neg x_2$ $F_2(x_1, x_2) = 0$

Data Generation

$$\varphi(x_1, x_2, y_1, y_2) : (x_1 \vee x_2 \vee y_1) \wedge (\neg x_1 \vee \neg x_2 \vee \neg y_2)$$

x_1	x_2	y_1	y_2
0	0	1	0/1
0	1	0/1	0/1
1	0	0/1	0/1
1	1	0/1	0

Magical Sampler →

x_1	x_2	y_1	y_2
0	0	1	0
0	1	1	0
1	0	1	0
1	1	1	0

- Possible Skolem functions:
  - $F_1(x_1, x_2) = \neg(x_1 \vee x_2)$ $\quad F_1(x_1, x_2) = \neg x_1$ $\quad F_1(x_1, x_2) = \neg x_2$ $\quad F_1(x_1, x_2) = 1$
  - $F_2(x_1, x_2) = \neg(x_1 \wedge x_2)$ $\quad F_2(x_1, x_2) = \neg x_1$ $\quad F_2(x_1, x_2) = \neg x_2$ $\quad F_2(x_1, x_2) = 0$

Weighted Sampling to Rescue

- $W : X \cup Y \mapsto [0,1]$
- The probability of generation of an assignment is proportional to its weight.

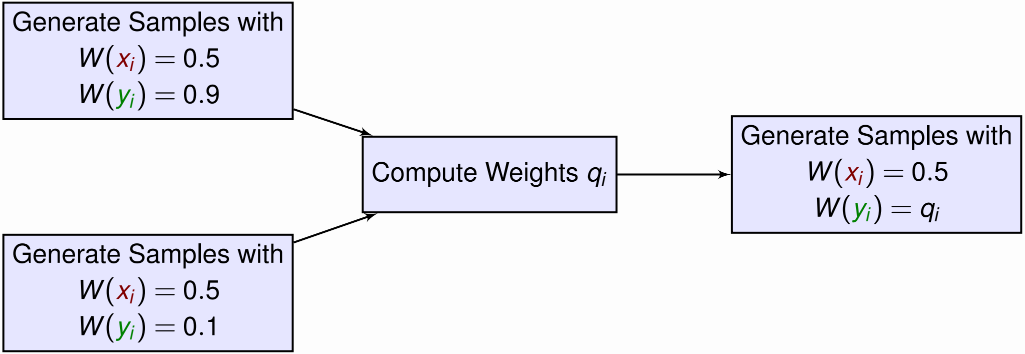
$$W(\sigma) = \prod_{\sigma(z_i)=1} W(z_i) \prod_{\sigma(z_i)=0} (1 - W(z_i))$$

- Example: $W(x_1) = 0.5 \quad W(x_2) = 0.5 \quad W(y_1) = 0.9 \quad W(y_2) = 0.1$
  $\sigma_1 = \{x_1 \mapsto 1, x_2 \mapsto 0, y_1 \mapsto 0, y_2 \mapsto 1\}$

$$W(\sigma_1) = 0.5 \times (1 - 0.5) \times (1 - 0.9) \times 0.1 = 0.0025$$

- Uniform sampling is a special case where all variables are assigned weight of 0.5.

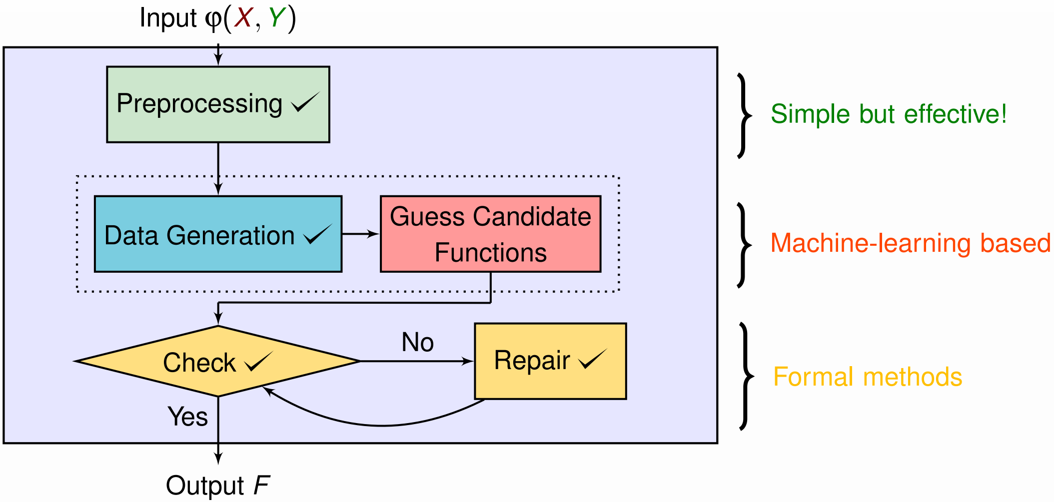
Data Generation



Different Sampling Strategies

- Knowledge representation based techniques
  (Yuan,Shultz, Pixley,Miller,Aziz 1999)
  (Yuan,Aziz, Pixley,Albin, 2004)
  (Kukula and Shiple, 2000)
  (Sharma, Gupta, Meel, Roy, 2018)
  (Gupta, Sharma, Meel, Roy, 2019)
- Hashing based techniques
  (Chakraborty, Meel, and Vardi 2013, 2014,2015)
  (Soos, Meel, and Gocht 2020)
- Mutation based techniques
  (Dutra, Laeufer, Bachrach, Sen, 2018)
- Markov Chain Monte Carlo based techniques
  (Wei and Selman,2005)
  (Kitchen,2010)
- Constraint solver based techniques
  (Ermon, Gomes, Sabharwal, Selman,2012)
- Belief networks based techniques
  (Dechter, Kask, Bin, Emek,2002)
  (Gogate and Dechter,2006)

Machine-learning based guessing of candidate Skolem functions (Manthan)



Learn Candidate Function: Decision Tree Classifier

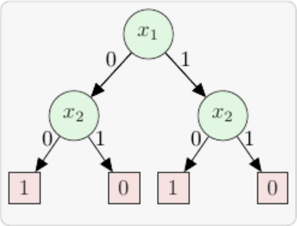
$$\varphi(x_1, x_2, y_1, y_2) : (x_1 \vee x_2 \vee y_1) \wedge (\neg x_1 \vee \neg x_2 \vee \neg y_2)$$

- To learn y_2
  - Feature set: valuation of x_1, x_2, y_1
  - Label: valuation of y_2
  - Learn decision tree to represent y_2 in terms of x_1, x_2, y_1
- To learn y_1
  - Feature set: valuation of x_1, x_2
  - Label: valuation of y_1
  - Learn decision tree to represent y_1 in terms of x_1, x_2

x_1	x_2	y_1	y_2
0	0	1	0
0	1	0	1
1	0	1	1
1	1	0	0

Learning Candidate Functions

x_1	x_2	y_1	y_2
0	0	1	0
0	1	0	1
1	0	1	1
1	1	0	0

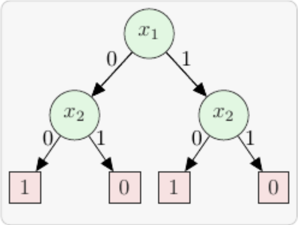


$p_1 := (\neg x_1 \wedge \neg x_2)$,
$p_2 := (x_1 \wedge \neg x_2)$

f_1 = if p_1 then 1
elif p_2 then 1
else 0

Learning Candidate Functions

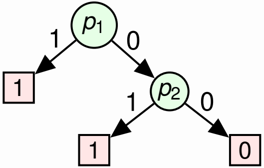
x_1	x_2	y_1	y_2
0	0	1	0
0	1	0	1
1	0	1	1
1	1	0	0



$p_1 := (\neg x_1 \wedge \neg x_2)$,
$p_2 := (x_1 \wedge \neg x_2)$
f_1 = if p_1 then 1
elif p_2 then 1
else 0

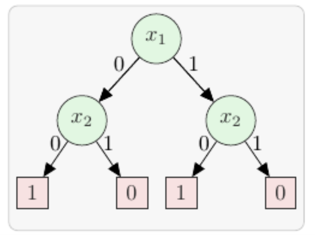
Can reorder p_1, p_2

Learning one level decision list



What Kind of Learning

x_1	x_2	y_1	y_2
0	0	1	0
0	1	0	1
1	0	1	1
1	1	0	0



$p_1 := (\neg x_1 \wedge \neg x_2)$,
$p_2 := (x_1 \wedge \neg x_2)$
f_1 = if p_1 then 1
elif p_2 then 1
else 0

Learning without Error
Every row is a solution of $\varphi(X, Y)$

Learning with Errors
The data is only a subset of solutions.

Revisiting the Repair Module: Candidate Identification

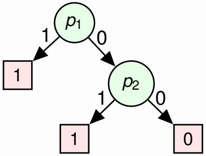
$$E(X, Y, Y') := \varphi(X, Y) \wedge \neg\varphi(X, Y') \wedge (Y' \leftrightarrow F(X))$$

- $\sigma \models E(X, Y, Y')$ be a counterexample to fix.
- Use MaxSAT to find a *nicer* counterexample σ'
- Repair patches: If $\underbrace{x_1 \wedge x_2 \wedge \neg y_1}_{\beta=\{x_1, x_2, \neg y_1\}}$ then $y_2 = 1$

Repair: Adding Level to Decision List

- Candidates are from one level decision list:
  - Learned decision tree: If p_1 then 1, elif p_2 then 1, else 0.
  - p_1, p_2 can be reordered.

Can reorder p_1, p_2



Repair: Adding Level to Decision List

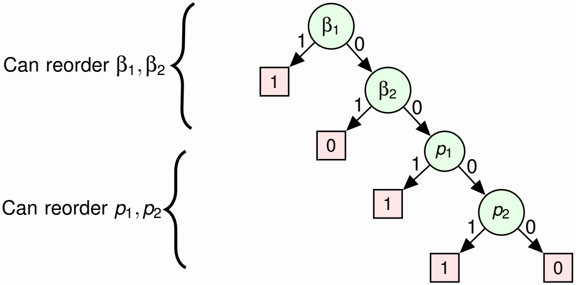
- Candidates are from one level decision list:
  - Learned decision tree: If p_1 then 1, elif p_2 then 1, else 0.
  - p_1, p_2 can be reordered.

- Suppose in repair iterations, we have learned: If β_1 then 1, ... β_2 then 0

  ......

- β_1 and β_2 can be reordered.

- From one-level decision list to two-level decision list.



Manthan

$\varphi(X, Y)$

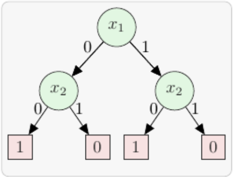
$X = \{x_1, x_2\}$

$Y = \{y_1, y_2\}$

Data Generation

x_1	x_2	y_1	y_2
0	0	1	0
0	1	0	1
1	0	1	0
1	1	0	1

Learn Candidates



Verify Candidates

Check Satisfiability of $E(X, Y, Y')$

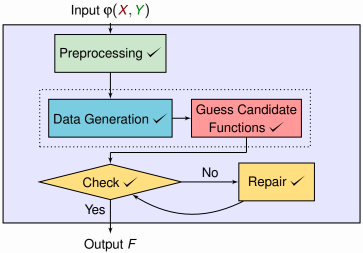
SAT, σ

$G_\sigma(X, Y)$

UNSAT Core-based Repair

UNSAT

Return F



Deep Dive 2

Knowledge Compilation for Boolean Functional Synthesis

- The Guess-check-repair approach was input-agnostic.
- Suffers from worst-case exponential blowup (unavoidable due to hardness results).

- The Guess-check-repair approach was input-agnostic.
- Suffers from worst-case exponential blowup (unavoidable due to hardness results).

This leads us to ask

- Are there special properties of input specification which guarantee provably fast/small solutions?
- Can we develop new algorithms exploiting these properties?

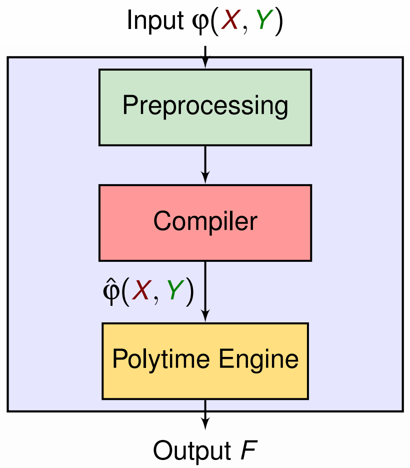
Deep Dive 2: Knowledge Representations and Compilation for Synthesis

- The Guess-check-repair approach was input-agnostic.
- Suffers from worst-case exponential blowup (unavoidable due to hardness results).

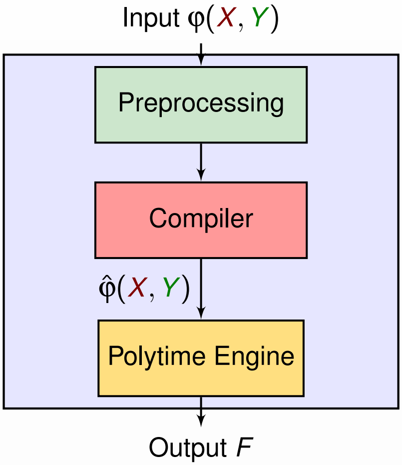
This leads us to ask

- Are there special properties of input specification which guarantee provably fast/small solutions?
- Can we develop new algorithms exploiting these properties?

Leads us to the rich area of Knowledge representations and Knowledge compilation.


Input φ(X, Y)
Preprocessing
Compiler
φ̂(X, Y)
Polytime Engine
Output F

Deep Dive 2: Knowledge Representations and Compilation for Synthesis



The question we will address in this deep dive...

What is $\widehat{\varphi}(\boldsymbol{X}, \boldsymbol{Y})$, i.e., representation of input s.t., Polytime Engine suffices for synthesis?

Let's start with a simple case

What if there is only one output, i.e., $|\boldsymbol{Y}| = 1$.

Let's start with a simple case

What if there is only one output, i.e., $|\boldsymbol{Y}| = 1$.

1-output synthesis is easy: We don't even need to change the Spec!

Spec $\varphi(\boldsymbol{X}, y_1)$:

Let's start with a simple case

What if there is only one output, i.e., $|\boldsymbol{Y}| = 1$.

1-output synthesis is easy: We don't even need to change the Spec!

Spec $\varphi(\boldsymbol{X}, y_1)$: $\varphi(\boldsymbol{X}, 1)$ is a Skolem function for y_1 in $\varphi(\boldsymbol{X}, y_1)$

Let's start with a simple case

What if there is only one output, i.e., $|\boldsymbol{Y}| = 1$.

1-output synthesis is easy: We don't even need to change the Spec!

Spec $\varphi(\boldsymbol{X}, y_1)$: $\varphi(\boldsymbol{X}, 1)$ is a Skolem function for y_1 in $\varphi(\boldsymbol{X}, y_1)$

For any $\boldsymbol{X}$, we have $\exists y_1 \varphi(\boldsymbol{X}, y_1) \Leftrightarrow \varphi(\boldsymbol{X}, 1) \vee \varphi(\boldsymbol{X}, 0) \Leftrightarrow \varphi(\boldsymbol{X}, \varphi(\boldsymbol{X}, 1))$.

Let's start with a simple case

What if there is only one output, i.e., $|\boldsymbol{Y}| = 1$.

1-output synthesis is easy: We don't even need to change the Spec!

Spec $\varphi(\boldsymbol{X}, y_1)$: $\varphi(\boldsymbol{X}, 1)$ is a Skolem function for y_1 in $\varphi(\boldsymbol{X}, y_1)$

For any $\boldsymbol{X}$, we have $\exists y_1 \varphi(\boldsymbol{X}, y_1) \Leftrightarrow \varphi(\boldsymbol{X}, 1) \vee \varphi(\boldsymbol{X}, 0) \Leftrightarrow \varphi(\boldsymbol{X}, \varphi(\boldsymbol{X}, 1))$.

Corollary

- $\neg\varphi(\boldsymbol{X}, 0)$ is also a correct Skolem function.

Let's start with a simple case

What if there is only one output, i.e., $|\boldsymbol{Y}| = 1$.

1-output synthesis is easy: We don't even need to change the Spec!

Spec $\varphi(\boldsymbol{X}, y_1)$: $\varphi(\boldsymbol{X}, 1)$ is a Skolem function for y_1 in $\varphi(\boldsymbol{X}, y_1)$

For any $\boldsymbol{X}$, we have $\exists y_1 \varphi(\boldsymbol{X}, y_1) \Leftrightarrow \varphi(\boldsymbol{X}, 1) \vee \varphi(\boldsymbol{X}, 0) \Leftrightarrow \varphi(\boldsymbol{X}, \varphi(\boldsymbol{X}, 1))$.

Corollary

- $\neg\varphi(\boldsymbol{X}, 0)$ is also a correct Skolem function.
- Any interpolant between these two is also a correct Skolem function. Jiang '09, Trivedi '03.

Multi-output synthesis and Existential quantification

Multi-output synthesis

Spec $\varphi(\boldsymbol{X}, y_1, \ldots y_m)$: Transform to 1-output synthesis

Multi-output synthesis and Existential quantification

Multi-output synthesis

Spec $\varphi(\boldsymbol{X}, y_1, \ldots y_m)$: Transform to 1-output synthesis

- Construct *new spec* $\varphi'(\boldsymbol{X}, y_m) \equiv \exists y_1 \ldots y_{m-1}\ \varphi$
  - Inputs $\boldsymbol{X}$, output y_m

Multi-output synthesis and Existential quantification

Multi-output synthesis

Spec $\varphi(\boldsymbol{X}, y_1, \ldots y_m)$: Transform to 1-output synthesis

- Construct *new spec* $\varphi'(\boldsymbol{X}, y_m) \equiv \exists y_1 \ldots y_{m-1}\ \varphi$
  - Inputs $\boldsymbol{X}$, output y_m
- Synthesize $F_m(\boldsymbol{X})$ for y_m from φ'

Multi-output synthesis and Existential quantification

Multi-output synthesis

Spec $\varphi(\boldsymbol{X}, y_1, \ldots y_m)$: Transform to 1-output synthesis

- Construct *new spec* $\varphi'(\boldsymbol{X}, y_m) \equiv \exists y_1 \ldots y_{m-1}\ \varphi$
  - Inputs $\boldsymbol{X}$, output y_m
- Synthesize $F_m(\boldsymbol{X})$ for y_m from φ'
- Construct *new spec* $\varphi''(\boldsymbol{X}, y_{m-1}, y_m) \equiv \exists y_1 \ldots y_{m-2}\, \varphi$
  - Inputs $\boldsymbol{X}$, y_m; output y_{m-1}

Multi-output synthesis and Existential quantification

Multi-output synthesis

Spec $\varphi(\boldsymbol{X}, y_1, \ldots y_m)$: Transform to 1-output synthesis

- Construct *new spec* $\varphi'(\boldsymbol{X}, y_m) \equiv \exists y_1 \ldots y_{m-1}\, \varphi$
  - Inputs $\boldsymbol{X}$, output y_m
- Synthesize $F_m(\boldsymbol{X})$ for y_m from φ'
- Construct *new spec* $\varphi''(\boldsymbol{X}, y_{m-1}, y_m) \equiv \exists y_1 \ldots y_{m-2}\, \varphi$
  - Inputs $\boldsymbol{X}$, y_m; output y_{m-1}
- Synthesize $F_{m-1}(\boldsymbol{X}, y_m)$ for y_{m-1}; substitute $F_m(\boldsymbol{X})$ for y_m

Multi-output synthesis and Existential quantification

Multi-output synthesis

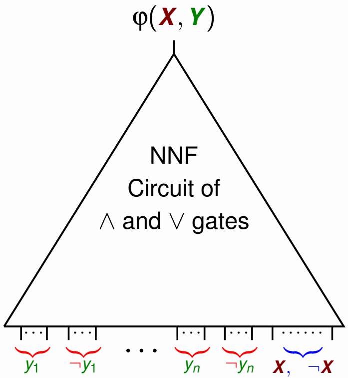
Spec $\varphi(\boldsymbol{X}, y_1, \ldots y_m)$: Transform to 1-output synthesis

- Construct *new spec* $\varphi'(\boldsymbol{X}, y_m) \equiv \exists y_1 \ldots y_{m-1}\ \varphi$
  - Inputs $\boldsymbol{X}$, output y_m
- Synthesize $F_m(\boldsymbol{X})$ for y_m from φ'
- Construct *new spec* $\varphi''(\boldsymbol{X}, y_{m-1}, y_m) \equiv \exists y_1 \ldots y_{m-2}\ \varphi$
  - Inputs $\boldsymbol{X}$, y_m; output y_{m-1}
- Synthesize $F_{m-1}(\boldsymbol{X}, y_m)$ for y_{m-1}; substitute $F_m(\boldsymbol{X})$ for y_m
- Repeat ...

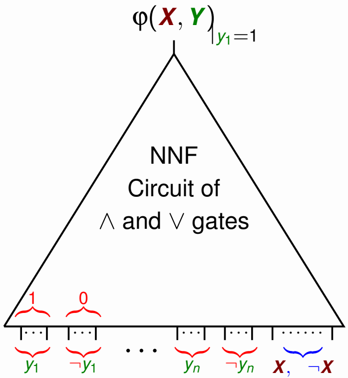
So, to compute Skolem functions, just need to efficiently compute

$\exists y_1 \ldots y_i\ \varphi(\boldsymbol{X}, y_1, \ldots y_m)\ \forall i \in \{1, \ldots m\}$

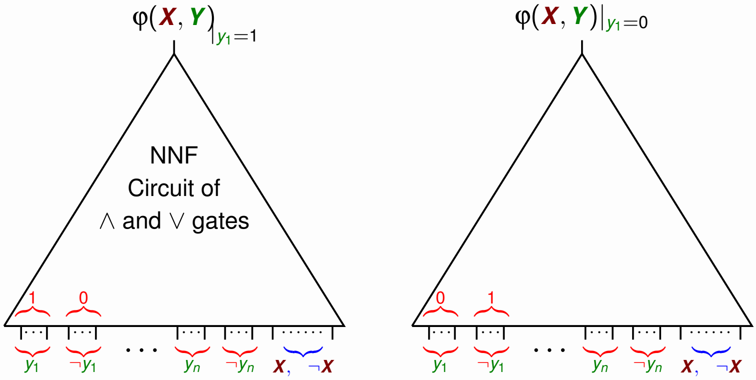
Existential Quantification with NNF circuits



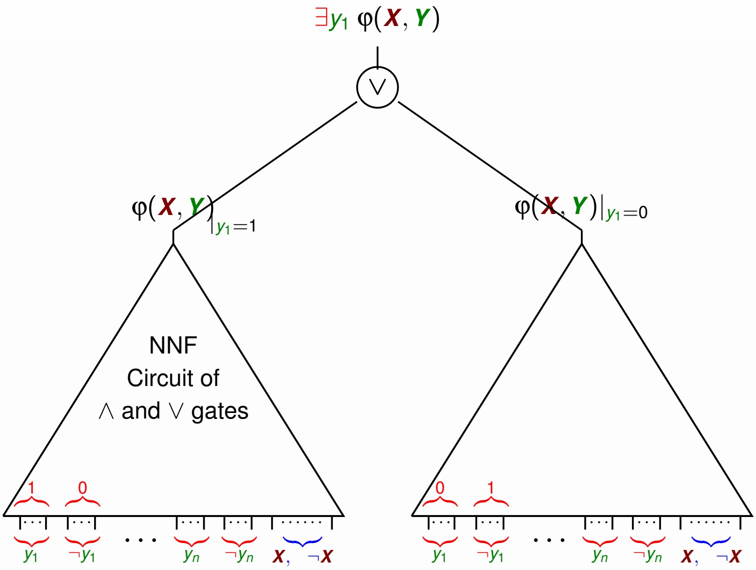
Existential Quantification with NNF circuits



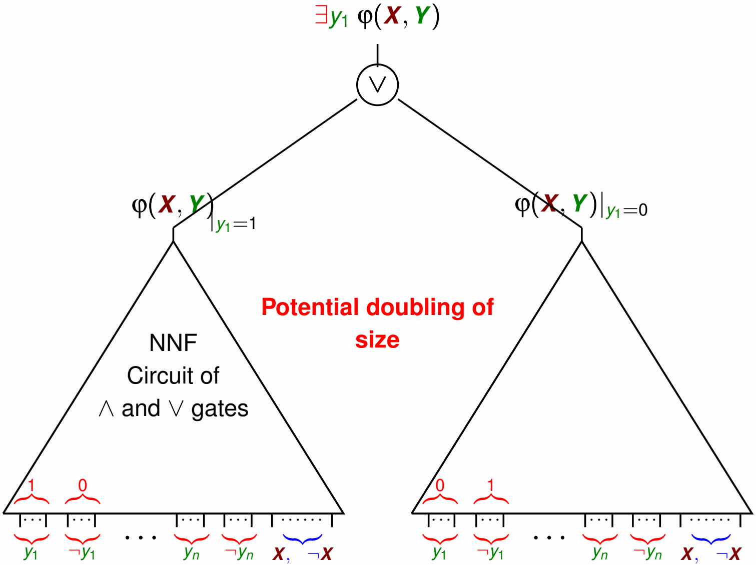
Existential Quantification with NNF circuits



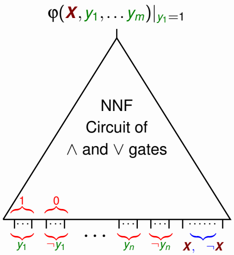
Existential Quantification with NNF circuits

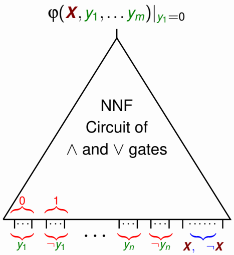


Existential Quantification with NNF circuits

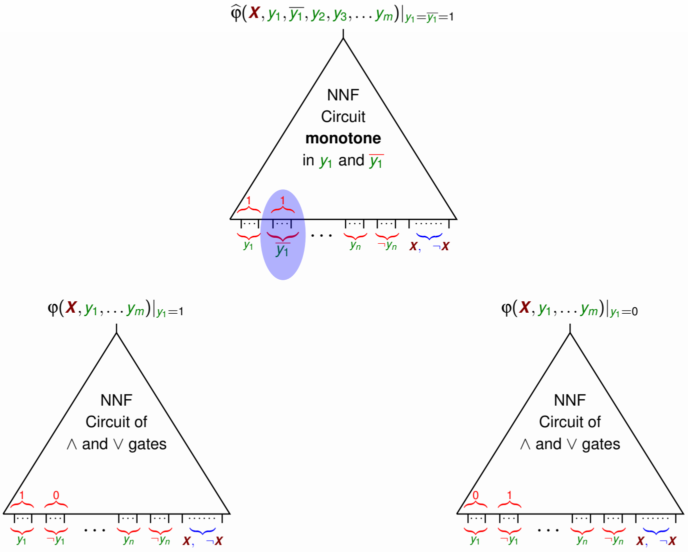


Over-approximating $\exists y_1\ \varphi(X, Y)$ Sans Doubling

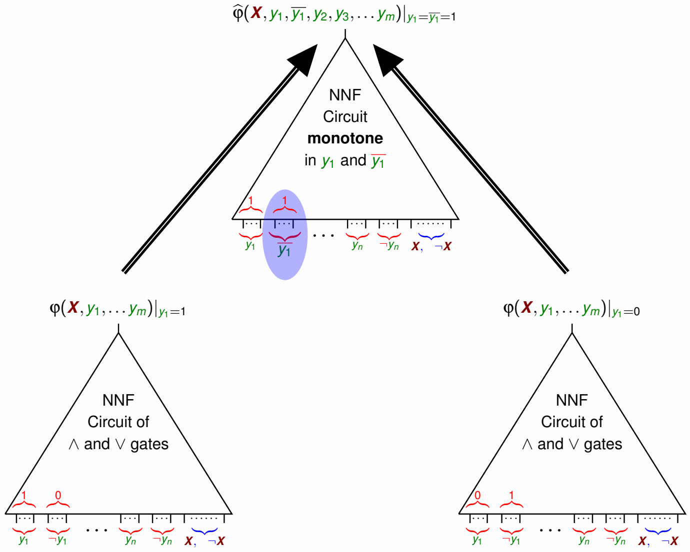




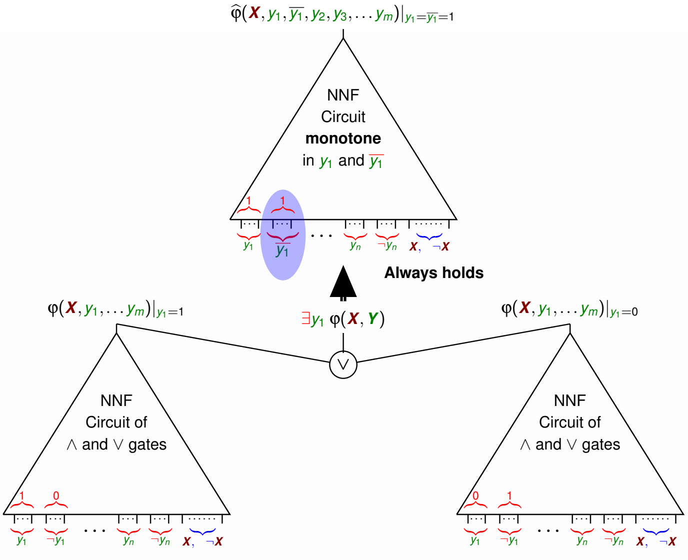
Over-approximating $\exists y_1\, \varphi(X, Y)$ Sans Doubling



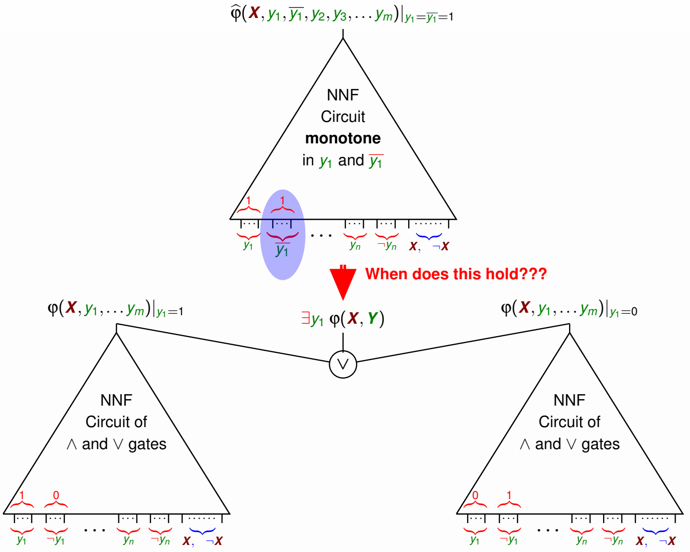
Over-approximating $\exists y_1\ \varphi(X, Y)$ Sans Doubling



Over-approximating $\exists y_1\ \varphi(X, Y)$ Sans Doubling



Can We Represent Quantification **Exactly** sans Blow-up?



The positive form and existential quantification

Take first output: $\exists y_1 \varphi(\boldsymbol{X}, \boldsymbol{Y}) \Rightarrow \widehat{\varphi}\,|_{y_1=1,\overline{y_1}=1}$.

The positive form and existential quantification

Take first output: $\exists y_1 \varphi(\boldsymbol{X}, \boldsymbol{Y}) \Rightarrow \widehat{\varphi}\,|_{y_1=1,\overline{y_1}=1}$. When does the reverse implication hold?

The positive form and existential quantification

Take first output: $\exists y_1 \varphi(\boldsymbol{X}, \boldsymbol{Y}) \Rightarrow \widehat{\varphi}\,|_{y_1=1,\overline{y_1}=1}$. When does the reverse implication hold?
Let's ask the opposite.

The positive form and existential quantification

Take first output: $\exists y_1 \varphi(\boldsymbol{X}, \boldsymbol{Y}) \Rightarrow \widehat{\varphi}\,|_{y_1=1,\overline{y_1}=1}$. When does the reverse implication hold?
Let's ask the opposite.

When do we have $\exists y_1 \varphi(\boldsymbol{X}, \boldsymbol{Y}) \not\Leftarrow \widehat{\varphi}\,|_{y_1=1,\overline{y_1}=1}$?

The positive form and existential quantification

Take first output: $\exists y_1 \varphi(\boldsymbol{X}, \boldsymbol{Y}) \Rightarrow \widehat{\varphi}\,|_{y_1=1,\overline{y_1}=1}$. When does the reverse implication hold?
Let's ask the opposite.

When do we have $\exists y_1 \varphi(\boldsymbol{X}, \boldsymbol{Y}) \not\Leftarrow \widehat{\varphi}\,|_{y_1=1,\overline{y_1}=1}$?

- Exactly when
  - $\widehat{\varphi}_1\,|_{y_1=1,\overline{y_1}=1} = 1$

Take first output: $\exists y_1 \varphi(\boldsymbol{X}, \boldsymbol{Y}) \Rightarrow \widehat{\varphi}\,|_{y_1=1,\overline{y_1}=1}$. When does the reverse implication hold?
Let's ask the opposite.

When do we have $\exists y_1 \varphi(\boldsymbol{X}, \boldsymbol{Y}) \not\Leftarrow \widehat{\varphi}\,|_{y_1=1,\overline{y_1}=1}$?

- Exactly when
  - $\widehat{\varphi}_1\,|_{y_1=1,\overline{y_1}=1} = 1$
  - $\exists y_1 \varphi(\boldsymbol{X}, \boldsymbol{Y}) \Leftrightarrow \varphi\,|_{y_1=1} \vee \varphi\,|_{y_1=0} = 0$
    - $\varphi\,|_{y_1=1} \Leftrightarrow \widehat{\varphi}\,|_{y_1=1,\overline{y_1}=0} = 0$
    - $\varphi\,|_{y_1=0} \Leftrightarrow \widehat{\varphi}\,|_{y_1=0,\overline{y_1}=1} = 0$

Take first output: $\exists y_1 \varphi(\mathbf{X}, \mathbf{Y}) \Rightarrow \widehat{\varphi}\,|_{y_1=1,\overline{y_1}=1}$. When does the reverse implication hold?
Let's ask the opposite.

When do we have $\exists y_1 \varphi(\mathbf{X}, \mathbf{Y}) \not\Leftarrow \widehat{\varphi}\,|_{y_1=1,\overline{y_1}=1}$?

- Exactly when
  - $\widehat{\varphi}_1\,|_{y_1=1,\overline{y_1}=1} \;=\; 1$
  - $\exists y_1 \varphi(\mathbf{X}, \mathbf{Y}) \Leftrightarrow \varphi\,|_{y_1=1} \;\vee\; \varphi\,|_{y_1=0} \;=\; 0$
    - $\varphi\,|_{y_1=1} \;\Leftrightarrow\; \widehat{\varphi}\,|_{y_1=1,\overline{y_1}=0} \;=\; 0$
    - $\varphi\,|_{y_1=0} \;\Leftrightarrow\; \widehat{\varphi}\,|_{y_1=0,\overline{y_1}=1} \;=\; 0$
    - (By monotonicity of $\widehat{\varphi}$ w.r.t y_1 and $\overline{y_1}$) $\widehat{\varphi}\,|_{y_1=0,\overline{y_1}=0} \;=\; 0$

y_1	$\overline{y_1}$	$\widehat{\varphi}$
1	1	1
1	0	0
0	1	0
0	0	0

The positive form and existential quantification

Take first output: $\exists y_1 \varphi(\boldsymbol{X}, \boldsymbol{Y}) \Rightarrow \widehat{\varphi}\,|_{y_1=1,\overline{y_1}=1}$. When does the reverse implication hold? Let's ask the opposite.

When do we have $\exists y_1 \varphi(\boldsymbol{X}, \boldsymbol{Y}) \not\Leftarrow \widehat{\varphi}\,|_{y_1=1,\overline{y_1}=1}$?

- Exactly when
  - $\widehat{\varphi}_1\,|_{y_1=1,\overline{y_1}=1} \;=\; 1$
  - $\exists y_1 \varphi(\boldsymbol{X}, \boldsymbol{Y}) \Leftrightarrow \varphi\,|_{y_1=1} \;\vee\; \varphi\,|_{y_1=0} \;=\; 0$
    - $\varphi\,|_{y_1=1} \;\Leftrightarrow\; \widehat{\varphi}\,|_{y_1=1,\overline{y_1}=0} \;=\; 0$
    - $\varphi\,|_{y_1=0} \;\Leftrightarrow\; \widehat{\varphi}\,|_{y_1=0,\overline{y_1}=1} \;=\; 0$
    - (By monotonicity of $\widehat{\varphi}$ w.r.t y_1 and $\overline{y_1}$) $\widehat{\varphi}\,|_{y_1=0,\overline{y_1}=0} \;=\; 0$
- For some values for other outputs and inputs, $\widehat{\varphi} \equiv y_1 \wedge \overline{y_1}$.

y_1	$\overline{y_1}$	$\widehat{\varphi}$
1	1	1
1	0	0
0	1	0
0	0	0

The positive form and existential quantification

Take first output: $\exists y_1 \varphi(\boldsymbol{X}, \boldsymbol{Y}) \Rightarrow \widehat{\varphi}\,|_{y_1=1,\overline{y_1}=1}$. When does the reverse implication hold? Let's ask the opposite.

When do we have $\exists y_1 \varphi(\boldsymbol{X}, \boldsymbol{Y}) \not\Leftarrow \widehat{\varphi}\,|_{y_1=1,\overline{y_1}=1}$?

- Exactly when
  - $\widehat{\varphi}_1\,|_{y_1=1,\overline{y_1}=1} \;=\; 1$
  - $\exists y_1 \varphi(\boldsymbol{X}, \boldsymbol{Y}) \Leftrightarrow \varphi\,|_{y_1=1} \;\vee\; \varphi\,|_{y_1=0} \;=\; 0$
    - $\varphi\,|_{y_1=1} \;\Leftrightarrow\; \widehat{\varphi}\,|_{y_1=1,\overline{y_1}=0} \;=\; 0$
    - $\varphi\,|_{y_1=0} \;\Leftrightarrow\; \widehat{\varphi}\,|_{y_1=0,\overline{y_1}=1} \;=\; 0$
    - (By monotonicity of $\widehat{\varphi}$ w.r.t y_1 and $\overline{y_1}$) $\widehat{\varphi}\,|_{y_1=0,\overline{y_1}=0} \;=\; 0$
- For some values for other outputs and inputs, $\widehat{\varphi} \equiv y_1 \wedge \overline{y_1}$.

y_1	$\overline{y_1}$	$\widehat{\varphi}$
1	1	1
1	0	0
0	1	0
0	0	0

So, what should we avoid?

- For some values for the other variables, we have $\widehat{\varphi} \Leftrightarrow y_1 \wedge \overline{y_1}$.

The positive form and existential quantification

Take first output: $\exists y_1 \varphi(\boldsymbol{X}, \boldsymbol{Y}) \Rightarrow \widehat{\varphi}\,|_{y_1=1,\overline{y_1}=1}$. When does the reverse implication hold? Let's ask the opposite.

When do we have $\exists y_1 \varphi(\boldsymbol{X}, \boldsymbol{Y}) \not\Leftarrow \widehat{\varphi}\,|_{y_1=1,\overline{y_1}=1}$?

- Exactly when
  - $\widehat{\varphi}_1\,|_{y_1=1,\overline{y_1}=1} = 1$
  - $\exists y_1 \varphi(\boldsymbol{X}, \boldsymbol{Y}) \Leftrightarrow \varphi\,|_{y_1=1} \vee \varphi\,|_{y_1=0} = 0$
    - $\varphi\,|_{y_1=1} \Leftrightarrow \widehat{\varphi}\,|_{y_1=1,\overline{y_1}=0} = 0$
    - $\varphi\,|_{y_1=0} \Leftrightarrow \widehat{\varphi}\,|_{y_1=0,\overline{y_1}=1} = 0$
    - (By monotonicity of $\widehat{\varphi}$ w.r.t y_1 and $\overline{y_1}$) $\widehat{\varphi}\,|_{y_1=0,\overline{y_1}=0} = 0$
- For some values for other outputs and inputs, $\widehat{\varphi} \equiv y_1 \wedge \overline{y_1}$.

y_1	$\overline{y_1}$	$\widehat{\varphi}$
1	1	1
1	0	0
0	1	0
0	0	0

So, what should we avoid?

- For some values for the other variables, we have $\widehat{\varphi} \Leftrightarrow y_1 \wedge \overline{y_1}$.
- If we can avoid it, we get $\exists y_1 \varphi(\boldsymbol{X}, \boldsymbol{Y}) \Leftrightarrow \widehat{\varphi}\,|_{y_1=1,\overline{y_1}=1}$

The positive form and existential quantification

Take first output: $\exists y_1 \varphi(\boldsymbol{X}, \boldsymbol{Y}) \Rightarrow \widehat{\varphi}\,|_{y_1=1,\overline{y_1}=1}$. When does the reverse implication hold?
Let's ask the opposite.

When do we have $\exists y_1 \varphi(\boldsymbol{X}, \boldsymbol{Y}) \not\Leftarrow \widehat{\varphi}\,|_{y_1=1,\overline{y_1}=1}$?

- Exactly when
  - $\widehat{\varphi}_1\,|_{y_1=1,\overline{y_1}=1} \; = 1$
  - $\exists y_1 \varphi(\boldsymbol{X}, \boldsymbol{Y}) \Leftrightarrow \varphi\,|_{y_1=1} \;\vee\; \varphi\,|_{y_1=0} \; = 0$
    - $\varphi\,|_{y_1=1} \;\Leftrightarrow\; \widehat{\varphi}\,|_{y_1=1,\overline{y_1}=0} \; = 0$
    - $\varphi\,|_{y_1=0} \;\Leftrightarrow\; \widehat{\varphi}\,|_{y_1=0,\overline{y_1}=1} \; = 0$
    - (By monotonicity of $\widehat{\varphi}$ w.r.t y_1 and $\overline{y_1}$) $\widehat{\varphi}\,|_{y_1=0,\overline{y_1}=0} \; = 0$
- For some values for other outputs and inputs, $\widehat{\varphi} \equiv y_1 \wedge \overline{y_1}$.

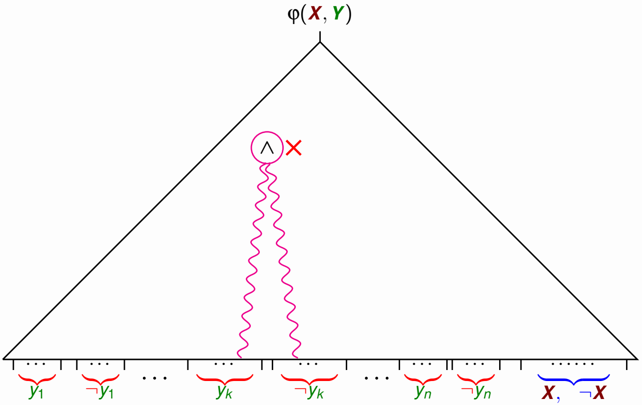
y_1	$\overline{y_1}$	$\widehat{\varphi}$
1	1	1
1	0	0
0	1	0
0	0	0

So, what should we avoid?

- For some values for the other variables, we have $\widehat{\varphi} \Leftrightarrow y_1 \wedge \overline{y_1}$.
- If we can avoid it, we get $\exists y_1 \varphi(\boldsymbol{X}, \boldsymbol{Y}) \Leftrightarrow \widehat{\varphi}\,|_{y_1=1,\overline{y_1}=1}$
- Can generalize this to multiple outputs...

A simple yet special Normal Form

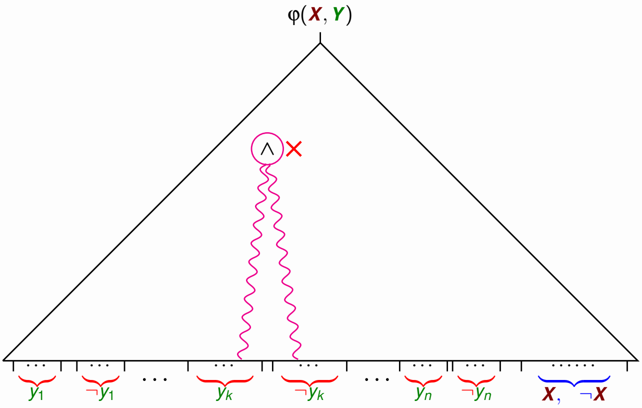
- Weak Decomposable Negation Normal Form (wDNNF)*: Forbidden **structure/syntax**



* S. Akshay, Supratik Chakraborty, Shubham Goel, Sumith Kulal, Shetal Shah, CAV'18.

A simple yet special Normal Form

- Weak Decomposable Negation Normal Form (wDNNF)*: Forbidden **structure/syntax**
- Generalizes DNNF†, well-studied in KR community.

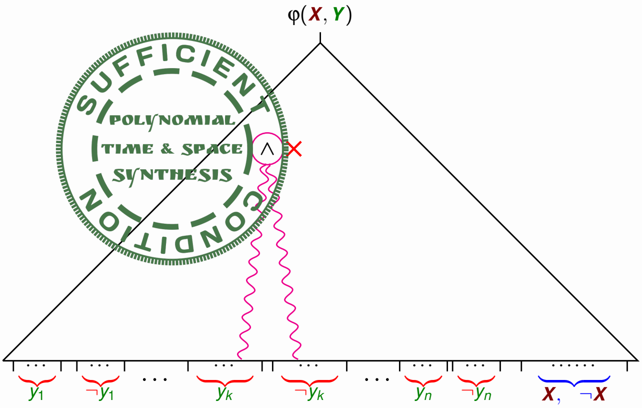


* S. Akshay, Supratik Chakraborty, Shubham Goel, Sumith Kulal, Shetal Shah, CAV'18.

† Adnan Darwiche, J. ACM '01

A simple yet special Normal Form

- Weak Decomposable Negation Normal Form (wDNNF)*: Forbidden **structure/syntax**
- Generalizes DNNF†, well-studied in KR community.



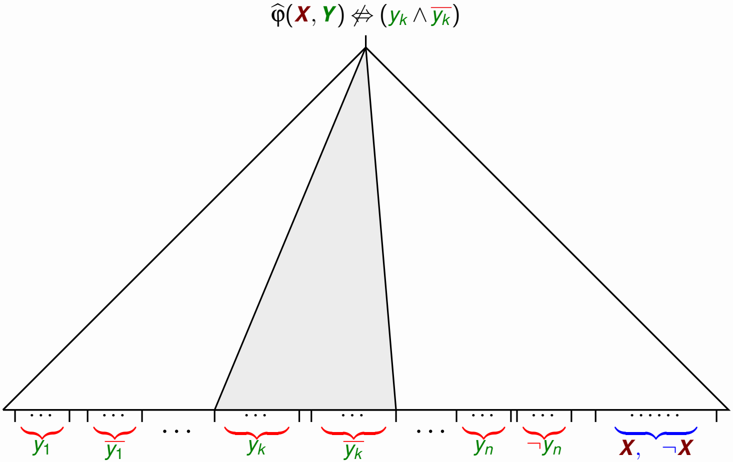
---

* S. Akshay, Supratik Chakraborty, Shubham Goel, Sumith Kulal, Shetal Shah, CAV'18.

† Adnan Darwiche, J. ACM '01

A semantic Normal Form

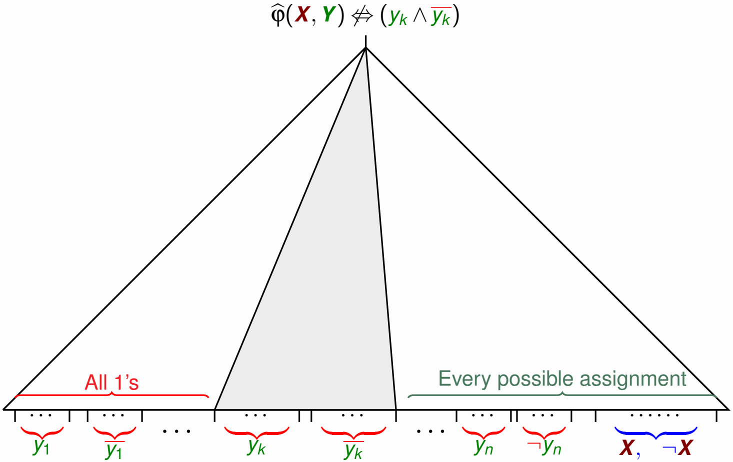
- Synthesis Negation Normal Form (SynNNF)*: Forbidden **semantics**



* S. Akshay, J. Arora, S. Chakraborty, S. Krishna, D. Raghunathan, S. Shah, FMCAD'19.

A semantic Normal Form

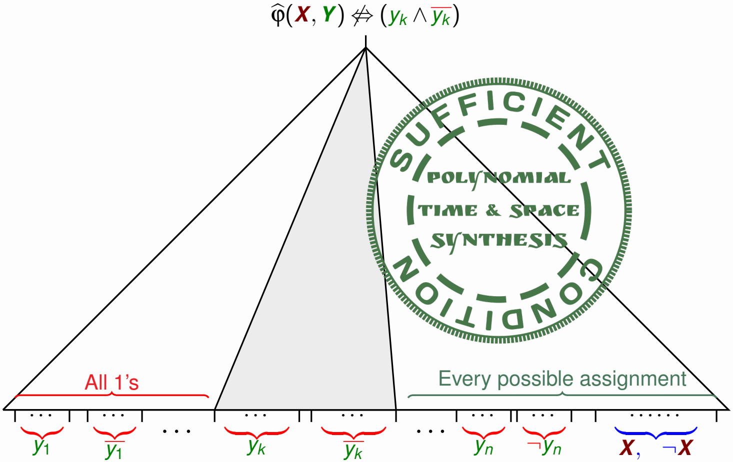
- Synthesis Negation Normal Form (SynNNF)*: Forbidden **semantics**



* S. Akshay, J. Arora, S. Chakraborty, S. Krishna, D. Raghunathan, S. Shah, FMCAD'19.

A semantic Normal Form

- Synthesis Negation Normal Form (SynNNF)*: Forbidden **semantics**



* S. Akshay, J. Arora, S. Chakraborty, S. Krishna, D. Raghunathan, S. Shah, FMCAD'19.

Skolem fn for y_i (in terms of $y_{i+1}, \ldots y_m, \boldsymbol{X}$)

- $\exists y_1, \ldots y_{i-1}\ \varphi(\boldsymbol{X}, y_1, \ldots y_{i-1}, 1, y_{i+1}, \ldots y_m)$

* Adnan Darwiche, J. App. Non Class. Logics'01

SynNNF: A negation normal form for efficient synthesis

Skolem fn for y_i (in terms of $y_{i+1}, \ldots y_m, \boldsymbol{X}$)

- $\exists y_1, \ldots y_{i-1}\ \varphi(\boldsymbol{X}, y_1, \ldots y_{i-1}, 1, y_{i+1}, \ldots y_m)$
- Equivalently, $\widehat{\varphi}\ |_{y_1=1,\overline{y_1}=1,\ldots y_{i-1}=1,\overline{y_{i-1}}=1,y_i=1,\overline{y_i}=0}$, if φ in SynNNF

---

* Adnan Darwiche, J. App. Non Class. Logics'01

SynNNF: A negation normal form for efficient synthesis

Skolem fn for y_i (in terms of $y_{i+1}, \ldots y_m, \boldsymbol{X}$)

- $\exists y_1, \ldots y_{i-1}\ \varphi(\boldsymbol{X}, y_1, \ldots y_{i-1}, 1, y_{i+1}, \ldots y_m)$
- Equivalently, $\widehat{\varphi}\ |_{y_1=1,\overline{y_1}=1,\ldots y_{i-1}=1,\overline{y_{i-1}}=1,y_i=1,\overline{y_i}=0}$, if φ in SynNNF

Poly-time/sized Skolem functions!

---

* Adnan Darwiche, J. App. Non Class. Logics'01

SynNNF: A negation normal form for efficient synthesis

Skolem fn for y_i (in terms of $y_{i+1}, \ldots y_m, \boldsymbol{X}$)

- $\exists y_1, \ldots y_{i-1}\ \varphi(\boldsymbol{X}, y_1, \ldots y_{i-1}, 1, y_{i+1}, \ldots y_m)$
- Equivalently, $\widehat{\varphi}\ |_{y_1=1,\overline{y_1}=1,\ldots y_{i-1}=1,\overline{y_{i-1}}=1,y_i=1,\overline{y_i}=0}$, if φ in SynNNF

Poly-time/sized Skolem functions!

Observations:

- Not purely structural restriction on representation of φ

---

* Adnan Darwiche, J. App. Non Class. Logics'01

SynNNF: A negation normal form for efficient synthesis

Skolem fn for y_i (in terms of $y_{i+1}, \ldots y_m, \boldsymbol{X}$)

- $\exists y_1, \ldots y_{i-1}\ \varphi(\boldsymbol{X}, y_1, \ldots y_{i-1}, 1, y_{i+1}, \ldots y_m)$
- Equivalently, $\widehat{\varphi}\ |_{y_1=1,\overline{y_1}=1,\ldots y_{i-1}=1,\overline{y_{i-1}}=1,y_i=1,\overline{y_i}=0}$, if φ in SynNNF

Poly-time/sized Skolem functions!

Observations:

- Not purely structural restriction on representation of φ
- Reminiscent of Deterministic DNNF (dDNNF)*
  - For every $\vee$ node representing $\varphi_1 \vee \varphi_2$, require $\varphi_1 \wedge \varphi_2 = \bot$.

* Adnan Darwiche, J. App. Non Class. Logics'01

Comparing the Normal Forms

- Every wDNNF, DNNF circuit is also in SynNNF.
- Every FBDD, ROBDD can be compiled in linear time to SynNNF.

Comparing the Normal Forms

- Every wDNNF, DNNF circuit is also in SynNNF.
- Every FBDD, ROBDD can be compiled in linear time to SynNNF.

SynNNF is strictly weaker/more succinct than wDNNF, DNNF, FBDD, ROBDD

Comparing the Normal Forms

- Every wDNNF, DNNF circuit is also in SynNNF.
- Every FBDD, ROBDD can be compiled in linear time to SynNNF.

SynNNF is strictly weaker/more succinct than wDNNF, DNNF, FBDD, ROBDD

Punchline!

SynNNF is exponentially more succinct than DNNF/dDNNF

Comparing the Normal Forms

- Every wDNNF, DNNF circuit is also in SynNNF.
- Every FBDD, ROBDD can be compiled in linear time to SynNNF.

SynNNF is strictly weaker/more succinct than wDNNF, DNNF, FBDD, ROBDD

Punchline!

SynNNF is exponentially more succinct than DNNF/dDNNF, which are themselves exponentially more succinct than ROBDDs/FBDD.

Comparing the Normal Forms

- Every wDNNF, DNNF circuit is also in SynNNF.
- Every FBDD, ROBDD can be compiled in linear time to SynNNF.

SynNNF is strictly weaker/more succinct than wDNNF, DNNF, FBDD, ROBDD

Punchline!

SynNNF is exponentially more succinct than DNNF/dDNNF, which are themselves exponentially more succinct than ROBDDs/FBDD.

What more can we do?

Comparing the Normal Forms

- Every wDNNF, DNNF circuit is also in SynNNF.
- Every FBDD, ROBDD can be compiled in linear time to SynNNF.

SynNNF is strictly weaker/more succinct than wDNNF, DNNF, FBDD, ROBDD

Punchline!

SynNNF is exponentially more succinct than DNNF/dDNNF, which are themselves exponentially more succinct than ROBDDs/FBDD.

What more can we do?

- Does there exists a necessary and sufficient condition for efficient synthesis?

Comparing the Normal Forms

- Every wDNNF, DNNF circuit is also in SynNNF.
- Every FBDD, ROBDD can be compiled in linear time to SynNNF.

SynNNF is strictly weaker/more succinct than wDNNF, DNNF, FBDD, ROBDD

Punchline!

SynNNF is exponentially more succinct than DNNF/dDNNF, which are themselves exponentially more succinct than ROBDDs/FBDD.

What more can we do?

- Does there exists a necessary and sufficient condition for efficient synthesis?
- Subset-And-Unrealizable Normal Form (SAUNF) P. Shah, A. Bansal, S. Akshay, S. Chakraborty, LICS'21.

Compilation to SynNNF and SAUNF

- What about general classes of specs?
  - CNF specs: NNF circuits don't always admit efficient synthesis

Compilation to SynNNF and SAUNF

- What about general classes of specs?
  - CNF specs: NNF circuits don't always admit efficient synthesis

Compiling CNF to SynNNF [Akshay et al. FMCAD'19.]

- Algorithm for compilation: uses ideas from dDNNF-compilation and more
- Prototype implementation C2Syn
- Worst-case exponential-time and space
  - Unavoidable due to hardness results

Compilation to SynNNF and SAUNF

- What about general classes of specs?
  - CNF specs: NNF circuits don't always admit efficient synthesis

Compiling CNF to SynNNF [Akshay et al. FMCAD'19.]

- Algorithm for compilation: uses ideas from dDNNF-compilation and more
- Prototype implementation C2Syn
- Worst-case exponential-time and space
  - Unavoidable due to hardness results

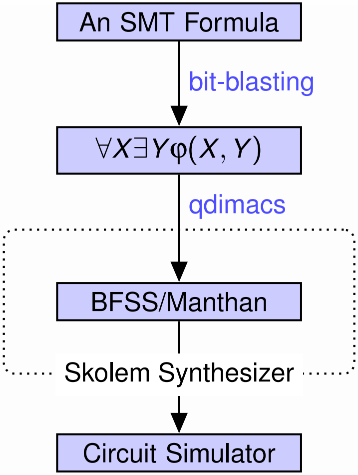
Compiling CNF to SAUNF [Shah et al. LICS'21.]

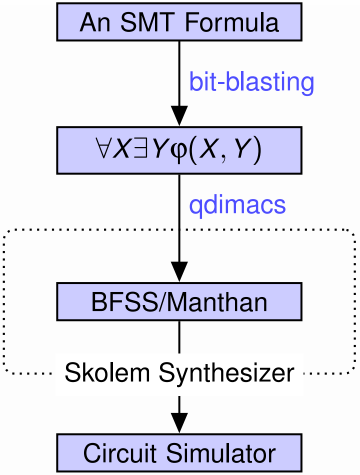
- Algorithm for compilation
- Future work: Implementation and comparisons!

Outline

1. Application Domains
2. Theoretical Hardness and Practical Algorithms
3. Deep Dives
4. Tool Demo
5. Conclusion and the Way Forward

Tool Demo: Pipeline

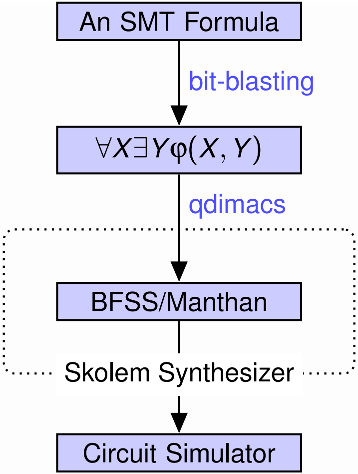





```
(set-logic BV)
;; out function with two 2 bit arguments
(declare-fun out ( (_ BitVec 2) (_ BitVec 2)) (_ BitVec 2))
;; declaring the constant
(declare-const inp1 (_ BitVec 2))
(declare-const inp2 (_ BitVec 2))
;; output of out function should be greater than or equal to first input
(assert (bvuge (out inp1 inp2) inp1))
;; output of out function should be greater than or equal to second input
(assert (bvuge (out inp1 inp2) inp2))
;; output of out function should be either be equal to first input
;; or to the second input
(assert (or (= inp1 (out inp1 inp2)) (= inp2 (out inp1 inp2))))
(check-sat)
```

An SMT formula

Tool Demo: Pipeline

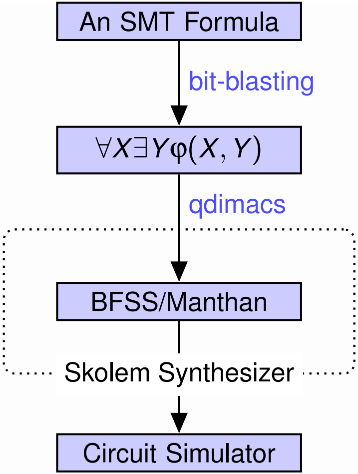



```
(set-logic BV)
;; out function with two 2 bit arguments
(declare-fun out ( (_ BitVec 2) (_ BitVec 2)) (_ BitVec 2))
;; declaring the constant
(declare-const inp1 (_ BitVec 2))
(declare-const inp2 (_ BitVec 2))
;; output of out function should be greater than or equal to first input
(assert (bvuge (out inp1 inp2) inp1))
;; output of out function should be greater than or equal to second input
(assert (bvuge (out inp1 inp2) inp2))
;; output of out function should be either be equal to first input
;; or to the second input
(assert (or (= inp1 (out inp1 inp2)) (= inp2 (out inp1 inp2))))
(check-sat)
```

An SMT formula

```
p cnf 12 32
a 3 5 7 8 0
e 1 2 4 6 9 10 11 12 0
1 -2 0
3 1 0
2 -3 -1 0
4 -5 0
1 -5 0
4 1 0
-2 6 0
7 6 0
2 -7 -6 0
4 -8 0
-8 6 0
4 6 0
-4 -8 -9 0
```

Qdimacs formula

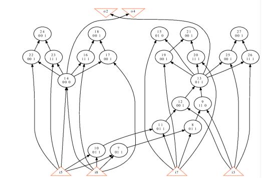



```
(set-logic BV)
;; out function with two 2 bit arguments
(declare-fun out ( (_ BitVec 2) (_ BitVec 2)) (_ BitVec 2))
;; declaring the constant
(declare-const inp1 (_ BitVec 2))
(declare-const inp2 (_ BitVec 2))
;; output of out function should be greater than or equal to first input
(assert (bvuge (out inp1 inp2) inp1))
;; output of out function should be greater than or equal to second input
(assert (bvuge (out inp1 inp2) inp2))
;; output of out function should be either be equal to first input
;; or to the second input
(assert (or (= inp1 (out inp1 inp2)) (= inp2 (out inp1 inp2))))
(check-sat)
```

An SMT formula

```
p cnf 12 32
a 3 5 7 8 0
e 1 2 4 6 9 10 11 12 0
1 -2 0
3 1 0
2 -3 -1 0
4 -5 0
1 -5 0
4 1 0
-2 6 0
7 6 0
2 -7 -6 0
4 -8 0
-8 6 0
4 6 0
-4 -8 -9 0
```

Qdimacs formula



Synthesized Skolem function

Outline

1. Application Domains
2. Theoretical Hardness and Practical Algorithms
3. Deep Dives
4. Tool Demo
5. Conclusion and the Way Forward

Summary

- Functional Synthesis is a fundamental problem with wide variety of applications
  - program synthesis, games and planning, circuit repair
- Long history of work that has sought to push the scalability envelope
- An exciting and diverse set of approaches
  - Guess, check, and repair
  - Knowledge representation
- Promise of scalability: Out of 609 benchmarks

2018	247 solved
2019	280 solved
2020	356 solved
2021	509 solved

Where do we go from here?

1. Benchmarks
2. Notion of Quality
3. Beyond Single Functions
4. Beyond Propositional Logic

Future Directions I: Benchmarks

Promise of scalability: Out of 609 benchmarks

2018 SOTA 247 solved

2019 SOTA 280 solved

2020 SOTA 356 solved

2021 SOTA 509 solved

B. Cook, 2022: **Virtuous cycle in Automated Reasoning**: ...application areas drives more investment in foundational tools, while improvements in the foundational tools drive further applications. Around and around.

Future Directions II: Search for Optimal Functions

- The current formulation allows the solver to find an arbitrary functions
- Opportunity to formalize the notion of quality
- Smaller size?
- Uses gates of particular type?
- Readable?

Future Directions III: Beyond Single Functions

- Enumeration of functions: Knowledge compilation
- Uniform sampling of functions: randomized strategies
- Counting of functions

Future Directions IV: Beyond Propositional Logic

- Past twenty years: Development of solvers with satisfiability modulo theory solvers
  - Capable of handling theories such as string, bitvectors, linear real arithmetic

- Lifting synthesis techniques to SMT
  - Knowledge compilation
  - Machine Learning techniques for SMT learning
  - Repair techniques

Questions?

Promise of scalability: Out of 609 benchmarks

2018 SOTA	247 solved
2019 SOTA	280 solved
2020 SOTA	356 solved
2021 SOTA	509 solved

The Future:

1. Benchmarks
2. Notion of Quality
3. Beyond Single Functions
4. Beyond Propositional Logic