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

AAAI-2022

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

What’s It All About

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

Spec by examples

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

What's It All About

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

Spec by examples

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

Spec by input-output relation

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

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

Spec by examples

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

X_1, X_2 →  → Y

Spec by input-output relation

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

Spec 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

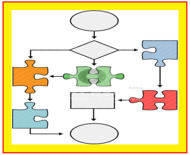
What's It All About

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

Spec by examples

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

Synthesis Algorithm



Spec by input-output relation

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

Spec 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

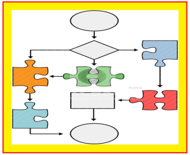
Wish I had an algorithm
that could help me ...

X_1, X_2 →  → Y

Spec by examples

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

Synthesis Algorithm



Spec by input-output relation

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

Spec 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

Focus of this tutorial

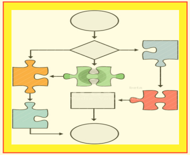
Wish I had an algorithm
that could help me ...

Spec by examples

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

Synthesis Algorithm

X_1, X_2 →  → Y



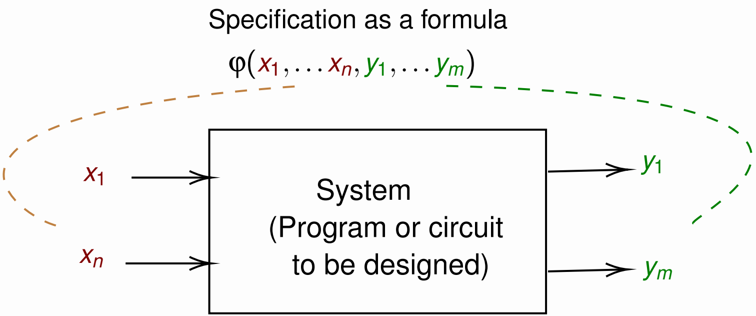
Spec by input-output relation

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

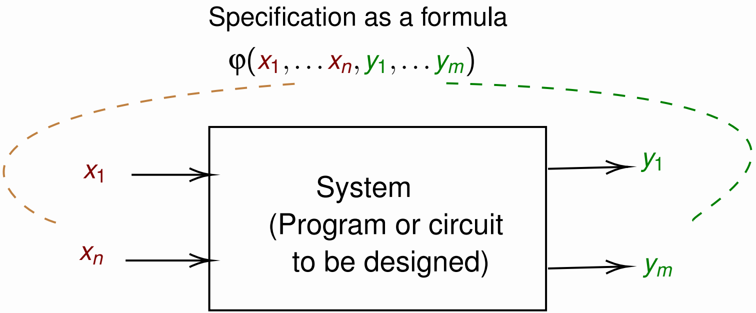
Spec in natural language

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

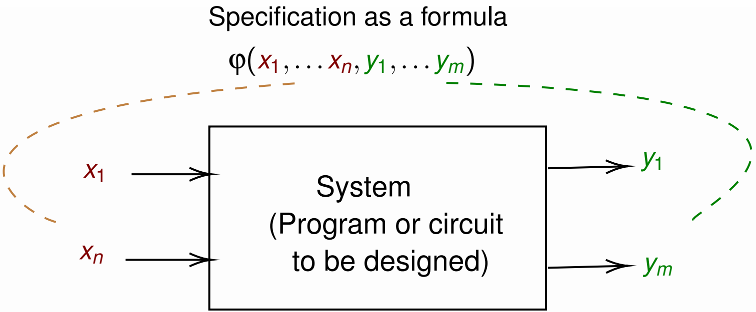
Automated Synthesis: A Generic View



Automated Synthesis: A Generic View

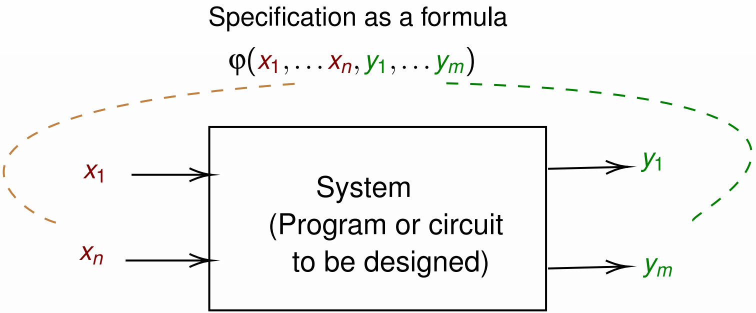


- Goal: Automatically synthesize system s.t. it satisfies $\varphi(x_1, .., x_n, y_1, .., y_m)$
  - x_i *input* variables (vector **X**)
  - y_j *output* variables (vector **Y**)



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

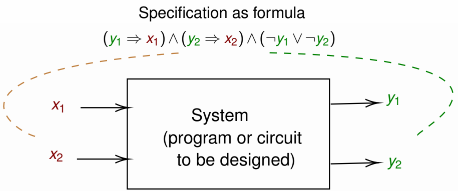
Automated Synthesis: A Generic View



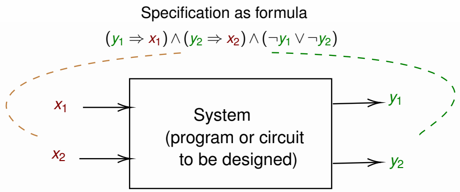
- Goal: Automatically synthesize system s.t. it satisfies $\varphi(x_1,..,x_n,y_1,..,y_m)$ **whenever possible**.
  - x_i *input* variables (vector **X**)
  - y_j *output* variables (vector **Y**)
- Need **Y** as functions **F** of
  - "History" of **X** and **Y**, "State" of system, ... in general

such that $\varphi(\mathbf{X}, \mathbf{F})$ is satisfied.

Automated Synthesis: Concrete View 1 (Memoryless Arbiter)

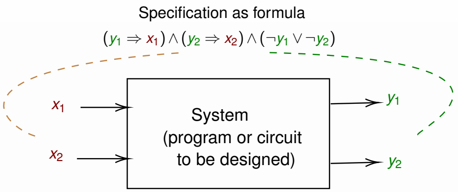


- Specification gives a relation between inputs & outputs



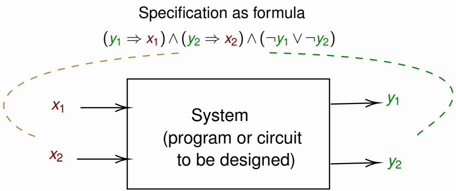
- Specification gives a relation between inputs & outputs
- Doesn't tell us how to obtain y_1, y_2 as functions of x_1, x_2

Automated Synthesis: Concrete View 1 (Memoryless Arbiter)



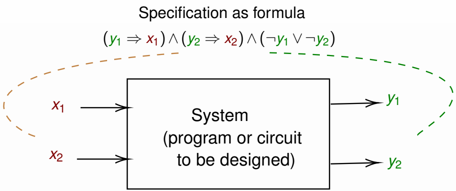
- Specification gives a relation between inputs & outputs
- Doesn't tell us how to obtain y_1, y_2 as functions of x_1, x_2
- Need to synthesize y_1, y_2 as functions of x_1, x_2 s.t. spec is satisfied

Automated Synthesis: Concrete View 1 (Memoryless Arbiter)



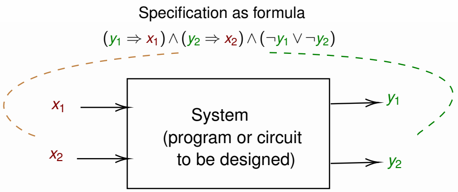
- Specification gives a relation between inputs & outputs
- Doesn't tell us how to obtain y_1, y_2 as functions of x_1, x_2
- Need to synthesize y_1, y_2 as functions of x_1, x_2 s.t. spec is satisfied
- Multiple solutions
  - $y_1 = x_1 \wedge \neg x_2$, $y_2 = x_2$
  - $y_1 = x_1$, $y_2 = x_2 \wedge \neg x_1$

Automated Synthesis: Concrete View 1 (Memoryless Arbiter)



- Specification gives a relation between inputs & outputs
- Doesn't tell us how to obtain y_1, y_2 as functions of x_1, x_2
- Need to synthesize y_1, y_2 as functions of x_1, x_2 s.t. spec is satisfied
- Multiple solutions
  - $y_1 = x_1 \wedge \neg x_2$, $y_2 = x_2$
  - $y_1 = x_1$, $y_2 = x_2 \wedge \neg x_1$
  - Admits "unfair" implementation

Automated Synthesis: Concrete View 1 (Memoryless Arbiter)

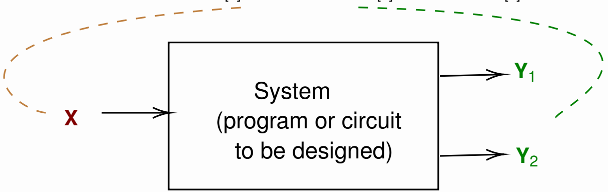


- Specification gives a relation between inputs & outputs
- Doesn't tell us how to obtain y_1, y_2 as functions of x_1, x_2
- Need to synthesize y_1, y_2 as functions of x_1, x_2 s.t. spec is satisfied
- Multiple solutions
  - $y_1 = x_1 \wedge \neg x_2$, $y_2 = x_2$
  - $y_1 = x_1$, $y_2 = x_2 \wedge \neg x_1$
  - Admits "unfair" implementation
- Suffices to give one "good enough" solution

Automated Synthesis: Concrete View 2 (Cryptanalysis)

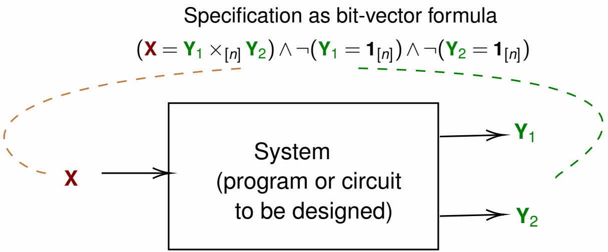
Specification as bit-vector formula

$$(\mathbf{X} = \mathbf{Y}_1 \times_{[n]} \mathbf{Y}_2) \wedge \neg(\mathbf{Y}_1 = \mathbf{1}_{[n]}) \wedge \neg(\mathbf{Y}_2 = \mathbf{1}_{[n]})$$



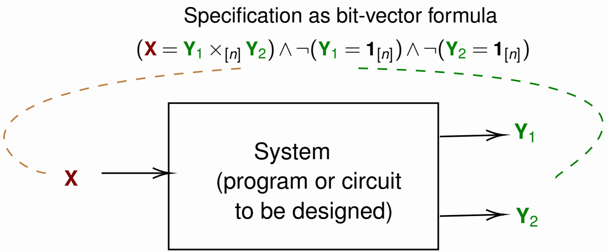
- Synthesize $\mathbf{Y}_1, \mathbf{Y}_2$ as functions of $\mathbf{X}$

Automated Synthesis: Concrete View 2 (Cryptanalysis)



- Synthesize $\mathbf{Y}_1, \mathbf{Y}_2$ as functions of $\mathbf{X}$
  - $\mathbf{Y}_1, \mathbf{Y}_2$ must be non-trivial factors of $\mathbf{X}$

Automated Synthesis: Concrete View 2 (Cryptanalysis)

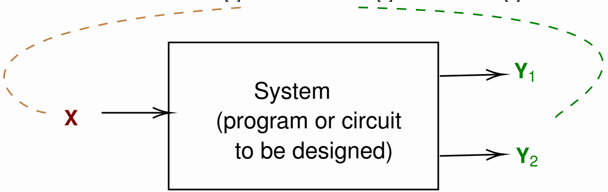


- Synthesize $\mathbf{Y}_1, \mathbf{Y}_2$ as functions of $\mathbf{X}$
  - $\mathbf{Y}_1, \mathbf{Y}_2$ must be non-trivial factors of $\mathbf{X}$
  - Not always satisfiable (if $\mathbf{X}$ is prime)

Automated Synthesis: Concrete View 2 (Cryptanalysis)

Specification as bit-vector formula

$$(\mathbf{X} = \mathbf{Y}_1 \times_{[n]} \mathbf{Y}_2) \wedge \neg(\mathbf{Y}_1 = \mathbf{1}_{[n]}) \wedge \neg(\mathbf{Y}_2 = \mathbf{1}_{[n]})$$



- Synthesize $\mathbf{Y}_1, \mathbf{Y}_2$ as functions of $\mathbf{X}$
  - $\mathbf{Y}_1, \mathbf{Y}_2$ must be non-trivial factors of $\mathbf{X}$
  - Not always satisfiable (if $\mathbf{X}$ is prime)
  - Efficient solution would break crypto systems

Reactive vs Functional Synthesis

Reactive synthesis

- System & environment in continuous temporal interaction
- Specification talks of **infinite sequence of inputs & outputs**
  - Temporal logic, automata over infinite words, ...
- Examples: Operating system, network switch, nuclear plant controller, ...

Reactive vs Functional Synthesis

Reactive synthesis

- System & environment in continuous temporal interaction
- Specification talks of **infinite sequence of inputs & outputs**
  - Temporal logic, automata over infinite words, ...
- Examples: Operating system, network switch, nuclear plant controller, ...
- **Not focus of this tutorial**

Reactive vs Functional Synthesis

Reactive synthesis

- System & environment in continuous temporal interaction
- Specification talks of **infinite sequence of inputs & outputs**
  - Temporal logic, automata over infinite words, ...
- Examples: Operating system, network switch, nuclear plant controller, ...
- **Not focus of this tutorial**

Functional synthesis

- System generates outputs in response to current inputs
- No dependence on past history
- Specification talks of **current input and current output**
  - Propositional/bit-vector/... logics suffice, no temporal operators
- Examples: program synthesis, arithmetic/numerical computation, next-state logic of reactive controllers, ...

Reactive vs Functional Synthesis

Reactive synthesis

- System & environment in continuous temporal interaction
- Specification talks of **infinite sequence of inputs & outputs**
  - Temporal logic, automata over infinite words, ...
- Examples: Operating system, network switch, nuclear plant controller, ...
- **Not focus of this tutorial**

Functional synthesis

- System generates outputs in response to current inputs
- No dependence on past history
- Specification talks of **current input and current output**
  - Propositional/bit-vector/... logics suffice, no temporal operators
- Examples: program synthesis, arithmetic/numerical computation, next-state logic of reactive controllers, ...
- **Focus of this tutorial**

Outline

First half: The basics

1. Formal Problem Statement
2. Application domains
3. Theoretical hardness and practical algorithms

Outline

First half: The basics

1. Formal Problem Statement
2. Application domains
3. Theoretical hardness and practical algorithms

A coffee/tea/dinner/drinks break

Second half: Under the hood

4. Deep Dives
    1. Knowledge compilation
    2. Counter-example guided
    3. Data-driven approaches
5. Tool demos and experimental results
6. Conclusion and the way forward

Outline

Boolean Functional Synthesis

Formal definition

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

- x_1 *input* variables (vector **X**)
- y_j *output* variables (vector **Y**)

Formal definition

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

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

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

$$\forall \mathbf{X}\big(\ \exists y_1 \dots y_m\ \varphi(\mathbf{X}, y_1 \dots y_{y_m})\ \Leftrightarrow\ \varphi(\mathbf{X}, F_1(\mathbf{X}), \dots F_m(\mathbf{X}))\ \big)$$

Boolean Functional Synthesis

Formal definition

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

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

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

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

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

Boolean Functional Synthesis

Formal definition

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

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

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

$$\forall \mathbf{X} (\ \exists y_1 \ldots y_m\ \varphi(\mathbf{X}, y_1 \ldots y_{y_m}) \Leftrightarrow \varphi(\mathbf{X}, F_1(\mathbf{X}), \ldots F_m(\mathbf{X}))\ )$$

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

- What if $\forall \mathbf{X} \exists \mathbf{Y}\ \varphi(\mathbf{X}, \mathbf{Y}) = 0$?

Formal definition

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

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

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

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

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

- What if $\forall \mathbf{X} \exists \mathbf{Y} \, \varphi(\mathbf{X}, \mathbf{Y}) = 0$?
  - Interesting as long as $\exists \mathbf{X} \exists \mathbf{Y} \, \varphi(\mathbf{X}, \mathbf{Y}) = 1$

Boolean Functional Synthesis

Formal definition

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

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

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

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

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

- What if $\forall \mathbf{X} \exists \mathbf{Y} \, \varphi(\mathbf{X}, \mathbf{Y}) = 0$?
  - Interesting as long as $\exists \mathbf{X} \exists \mathbf{Y} \, \varphi(\mathbf{X}, \mathbf{Y}) = 1$
  - $\mathbf{F}(\mathbf{X})$ must give right value of $\mathbf{Y}$ for all $\mathbf{X}$ s.t. $\exists \mathbf{Y} \varphi(\mathbf{X}, \mathbf{Y}) = 1$
    - $\mathbf{F}(\mathbf{X})$ inconsequential for other $\mathbf{X}$

Formal definition

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

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

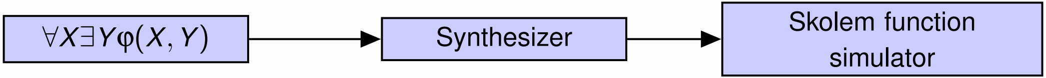
Synthesize Boolean functions $F_j(\mathbf{X})$ for each y_j s.t.

$$\forall \mathbf{X} (\ \exists y_1 \ldots y_m \ \varphi(\mathbf{X}, y_1 \ldots y_{y_m}) \Leftrightarrow \varphi(\mathbf{X}, F_1(\mathbf{X}), \ldots F_m(\mathbf{X}))\ )$$

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

- What if $\forall \mathbf{X} \exists \mathbf{Y}\ \varphi(\mathbf{X}, \mathbf{Y}) = 0$?
  - Interesting as long as $\exists \mathbf{X} \exists \mathbf{Y}\ \varphi(\mathbf{X}, \mathbf{Y}) = 1$
  - $\mathbf{F}(\mathbf{X})$ must give right value of $\mathbf{Y}$ for all $\mathbf{X}$ s.t. $\exists \mathbf{Y} \varphi(\mathbf{X}, \mathbf{Y}) = 1$
    - $\mathbf{F}(\mathbf{X})$ inconsequential for other $\mathbf{X}$
  - Given $\mathbf{X}$, $\mathbf{F}(\mathbf{X})$, easy to check if $\exists \mathbf{Y}\ \varphi(\mathbf{X}, \mathbf{Y}) = \varphi(\mathbf{X}, \mathbf{F}(\mathbf{X})) = 0$

A short tool demo



In a specific format

Outline

Application Domain 1: Program Synthesis

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

Application Domain 1: Program Synthesis

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

Specifications

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

Application Domain 1: Syntax-Guided Synthesis (SyGuS) formalization*

SyGuS was an attempt to formalize the core synthesis problem as a:

- a background theory (eg. QF_UFLIA)
- 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

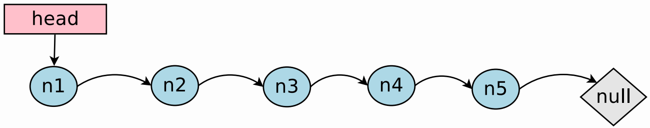
Application Domain 1: Developer Assistance: Heap Manipulations*

Reverse a singly linked list.

*Roy, SAS'13; Garg and Roy, SAS'15; Verma and Roy, ESEC/FSE'17; Verma and Roy, FMSD'22

Application Domain 1: Developer Assistance: Heap Manipulations*

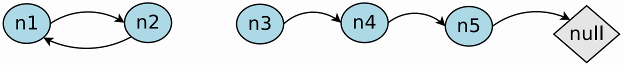
Reverse a singly linked list.



*Roy, SAS'13; Garg and Roy, SAS'15; Verma and Roy, ESEC/FSE'17; Verma and Roy, FMSD'22

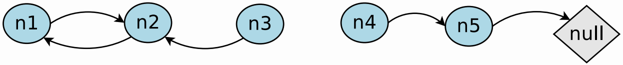
Application Domain 1: Developer Assistance: Heap Manipulations*

Reverse a singly linked list.



*Roy, SAS'13; Garg and Roy, SAS'15; Verma and Roy, ESEC/FSE'17; Verma and Roy, FMSD'22

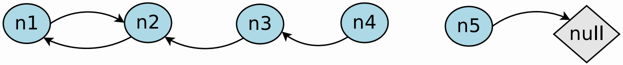
Reverse a singly linked list.



*Roy, SAS'13; Garg and Roy, SAS'15; Verma and Roy, ESEC/FSE'17; Verma and Roy, FMSD'22

Application Domain 1: Developer Assistance: Heap Manipulations*

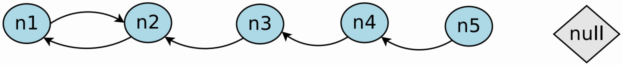
Reverse a singly linked list.



*Roy, SAS'13; Garg and Roy, SAS'15; Verma and Roy, ESEC/FSE'17; Verma and Roy, FMSD'22

Application Domain 1: Developer Assistance: Heap Manipulations*

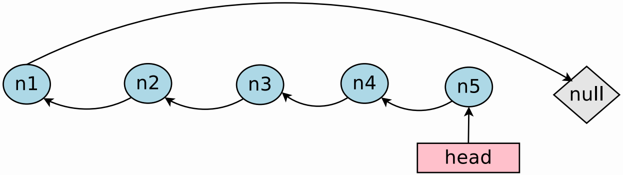
Reverse a singly linked list.



*Roy, SAS'13; Garg and Roy, SAS'15; Verma and Roy, ESEC/FSE'17; Verma and Roy, FMSD'22

Application Domain 1: Developer Assistance: Heap Manipulations*

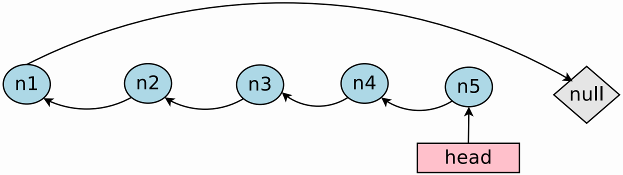
Reverse a singly linked list.



*Roy, SAS'13; Garg and Roy, SAS'15; Verma and Roy, ESEC/FSE'17; Verma and Roy, FMSD'22

Application Domain 1: Developer Assistance: Heap Manipulations*

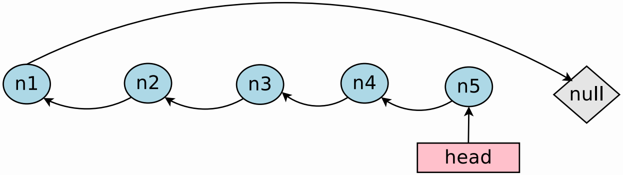
Reverse a singly linked list.



*Roy, SAS'13; Garg and Roy, SAS'15; Verma and Roy, ESEC/FSE'17; Verma and Roy, FMSD'22

Application Domain 1: Developer Assistance: Heap Manipulations*

Reverse a singly linked list.




```
0 tmp1 = x
1 tmp2 = tmp1.next
  while(not (tmp2 == null))
2   tmp0 = tmp2.next
3   tmp2.next = x
4   x = tmp2
5   tmp2 = tmp0
6 tmp1.next = tmp0
```

*Roy, SAS'13; Garg and Roy, SAS'15; Verma and Roy, ESEC/FSE'17; Verma and Roy, FMSD'22

- **Data preparation**: synthesize R scripts for complex data wrangling tasks*

---

*Feng et al., PLDI’17

†Wang et al., PLDI’17; https://scythe.cs.washington.edu/

‡Bavishi et al., OOPSLA’19

§Wang et al., POPL’20

- **Data preparation**: synthesize R scripts for complex data wrangling tasks*
- **Data extraction**: synthesize SQL queries† and Python scripts‡ from examples of input and output tables

*Feng et al., PLDI'17

†Wang et al., PLDI'17; https://scythe.cs.washington.edu/

‡Bavishi et al., OOPSLA'19

§Wang et al., POPL'20

Application Domain 1: Synthesis for Data Scientists

- **Data preparation**: synthesize R scripts for complex data wrangling tasks[*]
- **Data extraction**: synthesize SQL queries[†] and Python scripts[‡] from examples of input and output tables
- **Data visualization**: automatically synthesize visualizations from data with small example(s) as input [§]

[*] Feng et al., PLDI'17

[†] Wang et al., PLDI'17; https://scythe.cs.washington.edu/

[‡] Bavishi et al., OOPSLA'19

[§] Wang et al., POPL'20

Application Domain 1: Synthesis for Data Scientists

- **Data preparation**: synthesize R scripts for complex data wrangling tasks*
- **Data extraction**: synthesize SQL queries† and Python scripts‡ from examples of input and output tables
- **Data visualization**: automatically synthesize visualizations from data with small example(s) as input §
- **ML pipelines**: allows for generating supervised learning pipelines¶

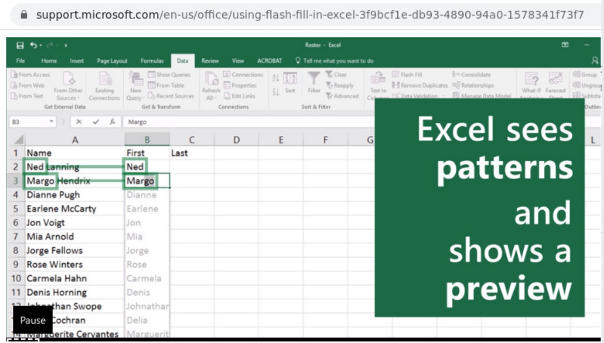
*Feng et al., PLDI'17

†Wang et al., PLDI'17; https://scythe.cs.washington.edu/

‡Bavishi et al., OOPSLA'19

§Wang et al., POPL'20

Flash Fill (Microsoft Excel)*

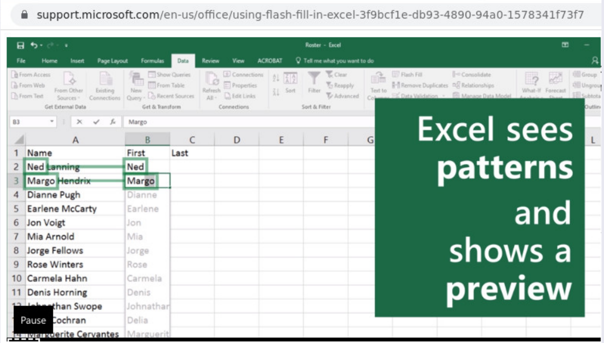


- automatically identifies patterns, and
- synthesizes a program in the background

*Gulwani et al., POPL'11; image and video at https://support.microsoft.com

†Singh and Gulwani, PVLDB'12; Singh and Gulwani, CAV'12; Harris and Gulwani, PLDI'11

Flash Fill (Microsoft Excel)[*]



- automatically identifies patterns, and
- synthesizes a program in the background

Similar line of tools for semantic string, number and table transformations.[†]

[*] Gulwani et al., POPL'11; image and video at https://support.microsoft.com

[†] Singh and Gulwani, PVLDB'12; Singh and Gulwani, CAV'12; Harris and Gulwani, PLDI'11

Application Domain 1: Intelligent Tutoring

- **Problem Generation**: for geometry, natural deduction and arithmetic*

---

*Alvinet al., AAAI'14; Ahmedet al., IJCAI'13; Andersen et al.,CHI'13

†Gulwani et al., PLDI'11

‡Singh et al., PLDI'13; Alur et al., IJCAI'13, Gulwani, GECCO'14

- **Problem Generation**: for geometry, natural deduction and arithmetic*
- **Solution Generation**: geometry constructions†

$MidPoint(Line(p_1, p_2))$

$c_1 = Circle(p_1, len(Line(p_1, p_2)))$
$c_2 = Circle(p_2, len(Line(p_1, p_2)))$
$q_1, q_2 = CircleCircleXn(c_1, c_2)$
$r = LineLineXn(Line(p_1, p_2), Line(q_1, q_2))$
return r
(simplified for presentation)

*Alvinet al., AAAI'14; Ahmedet al., IJCAI'13; Andersen et al.,CHI'13

†Gulwani et al., PLDI'11

‡Singh et al., PLDI'13; Alur et al., IJCAI'13, Gulwani, GECCO'14

Application Domain 1: Intelligent Tutoring

- **Problem Generation**: for geometry, natural deduction and arithmetic*
- **Solution Generation**: geometry constructions†

$MidPoint(Line(p_1, p_2))$

$c_1 = Circle(p_1, len(Line(p_1, p_2)))$
$c_2 = Circle(p_2, len(Line(p_1, p_2)))$
$q_1, q_2 = CircleCircleXn(c_1, c_2)$
$r = LineLineXn(Line(p_1, p_2), Line(q_1, q_2))$
return r
(simplified for presentation)

- **Feedback Generation**: introductory programming and automata‡

---

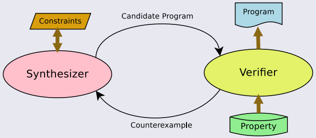
*Alvinet al., AAAI'14; Ahmedet al., IJCAI'13; Andersen et al.,CHI'13

†Gulwani et al., PLDI'11

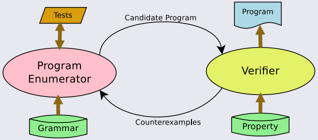
‡Singh et al., PLDI'13; Alur et al., IJCAI'13, Gulwani, GECCO'14

Application Domain 1: Algorithms for Program Synthesis*

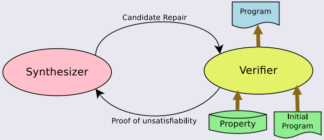
CEGIS (Symbolic)



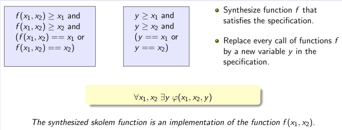
CEGIS (Enumerative)



SyPR: Proof-Guided Repairs



Reduction to QBF



*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; Verma et al., CGO'20; Golia et al., CAV'20; RedQBF: Golia et al., IJCAI'21

Application Domain 1: Program synthesis to Skolem Functional Synthesis*

$$f(x_1, x_2) \geq x_1 \text{ and}$$
$$f(x_1, x_2) \geq x_2 \text{ and}$$
$$(f(x_1, x_2) == x_1 \text{ or}$$
$$f(x_1, x_2) == x_2)$$

- Synthesize function f that satisfies the specification.

*Golia et al., IJCAI'21

Application Domain 1: Program synthesis to Skolem Functional Synthesis*

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

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

- Synthesize function f that satisfies the specification.
- Replace every call of functions f by a new variable y in the specification.

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

*Golia et al., IJCAI'21

Application Domain 1: Program synthesis to Skolem Functional Synthesis*

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

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

- Synthesize function f that satisfies the specification.
- Replace every call of functions f by a new variable y in the specification.

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

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

*Golia et al., IJCAI'21

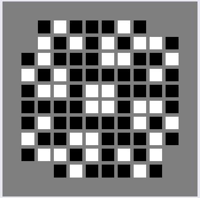
Application Domain 2: Games and planning

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

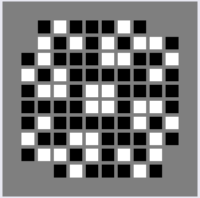
Conway's Game of Life

- 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



Application Domain 2: Games and planning

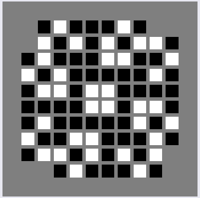
Conway's Game of Life



- 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!

Application Domain 2: Games and planning

Conway's Game of Life

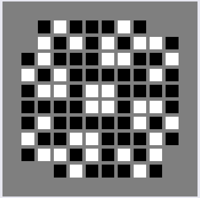


- Infinite 2D grid of cells, each alive or dead in each gen:
  1. (Under-pop) live cell with <2 live neighbors dies;
  2. (Status-quo) live cell with 2 or 3 live neighbors lives;
  3. (Over-pop) live cell >3 live neighbors dies;
  4. (Re-birth) dead cell with 3 live neighbors comes alive
- Objective: 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!
  - A storied history! From 1971 onwards... `https://conwaylife.com/wiki/Garden_of_Eden`

Application Domain 2: Games and planning

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



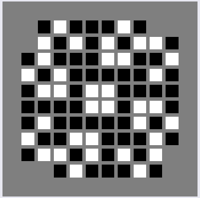
- Objective: Is there a Garden of Eden (GoE), a configuration with no predecessor?
  - If it does not exist, give a witnessing function that defines the predecessor!
  - A storied history! From 1971 onwards... `https://conwaylife.com/wiki/Garden_of_Eden`

Encoded as Skolem function existence and synthesis problem

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

Application Domain 2: Games and planning

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!
  - A storied history! From 1971 onwards... `https://conwaylife.com/wiki/Garden_of_Eden`

Encoded as Skolem function existence and synthesis problem

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

Application Domain 2: Games and planning as QBF

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

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

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

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

Application Domain 2: Games and planning as QBF

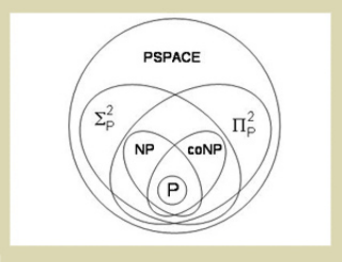
- $\forall \mathbf{X} \exists \mathbf{Y}\ T(\mathbf{X}, \mathbf{Y})$ has two alternating blocks of quantifiers: 2-QBF. In general, can have many!

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

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

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

- A rich theoretical history.
  - Textbook PSPACE-complete problem.



Application Domain 2: Games and planning as QBF

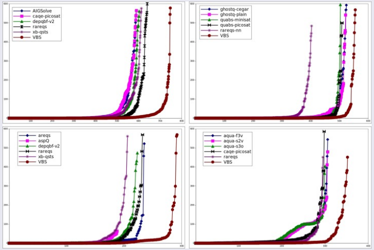
- $\forall \mathbf{X} \exists \mathbf{Y}\ T(\mathbf{X}, \mathbf{Y})$ has two alternating blocks of quantifiers: 2-QBF. In general, can have many!

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

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

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

- A rich theoretical history.
- Huge advances in tools! `https://www.qbflib.org`



Luca Pulina, Martina Seidl: The 2016 and 2017 QBF solvers evaluations (QBFEVAL'16 and QBFEVAL'17). Artif. Intell. 274: 224-248 (2019)

Application Domain 2: Games and planning as QBF

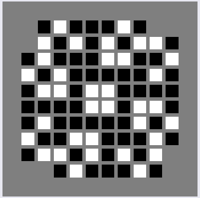
- $\forall \mathbf{X} \exists \mathbf{Y}\ T(\mathbf{X}, \mathbf{Y})$ has two alternating blocks of quantifiers: 2-QBF. In general, can have many!

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

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

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

- A rich theoretical history.
- Huge advances in tools! `https://www.qbflib.org`



Smallest GoE: found by QBFsolver(2011)

Christiaan Hartman, Marijn Heule, Kees Kwekkeboom, Alain Noels: Symmetry in Gardens of Eden. Electron. J. Comb. 20(3): P16 (2013)

Application Domain 2: Games and planning as QBF

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

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

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

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

- A rich theoretical history.
- Huge advances in tools! https://www.qbflib.org
- Any 2-player game can be coded as QBF
  - Skolem functions are winning strategies of Player 2 ($\exists$-player)!

Ankit Shukla, Armin Biere, Luca Pulina, Martina Seidl: A Survey on Applications of Quantified Boolean Formulas. ICTAI 2019: 78-84

Application Domain 2: Games and planning as QBF

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

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

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

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

- A rich theoretical history.
- Huge advances in tools! https://www.qbflib.org
- Any 2-player game can be coded as QBF
  - Skolem functions are winning strategies of Player 2 ($\exists$-player)!

Is it the case that for every first move of P1 there exists a first move of P2, s.t for every second move of P1 there exists a second move of P2... s.t. P2 can win!?

Ankit Shukla, Armin Biere, Luca Pulina, Martina Seidl: A Survey on Applications of Quantified Boolean Formulas. ICTAI 2019: 78-84

Application Domain 2: Games and planning as QBF

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

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

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

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

- A rich theoretical history.
- Huge advances in tools! https://www.qbflib.org
- Any 2-player game can be coded as QBF
  - Skolem functions are winning strategies of Player 2 ($\exists$-player)!
- Many applications of QBF that we dont have time to go into!

Ankit Shukla, Armin Biere, Luca Pulina, Martina Seidl: A Survey on Applications of Quantified Boolean Formulas. ICTAI 2019: 78-84

Application Domain 2: Games and planning with QBF and Skolem fns

Conformant or Conditional Planning in AI

Rintanen, J. 1999. Constructing conditional plans by a theorem-prover. Journal of Artificial Intelligence Research 10:323-352.

- Given set S of states, I, G, formulas over S defining initial and goal states and a set of non-det actions A,
  - does there exist a plan (seq of actions), s.t., for all possible contingencies (initial states and nondet choices), there exist an execution (seq of states) that reaches the goal state.

Application Domain 2: Games and planning with QBF and Skolem fns

Conformant or Conditional Planning in AI

Rintanen, J. 1999. Constructing conditional plans by a theorem-prover. Journal of Artificial Intelligence Research 10:323-352.

- Given set S of states, I, G, formulas over S defining initial and goal states and a set of non-det actions A,
  - does there exist a plan (seq of actions), s.t., for all possible contingencies (initial states and nondet choices), there exist an execution (seq of states) that reaches the goal state.
  - This is a $\exists\forall\exists$ formula, so in 3-QBF.

Application Domain 2: Games and planning with QBF and Skolem fns

Conformant or Conditional Planning in AI

Rintanen, J. 1999. Constructing conditional plans by a theorem-prover. Journal of Artificial Intelligence Research 10:323-352.

- Given set S of states, I, G, formulas over S defining initial and goal states and a set of non-det actions A,
  - does there exist a plan (seq of actions), s.t., for all possible contingencies (initial states and nondet choices), there exist an execution (seq of states) that reaches the goal state.
  - This is a $\exists\forall\exists$ formula, so in 3-QBF.
  - Can also be encoded as $\forall\exists$. Rintanen, J. 2007. Asymptotically Optimal Encodings of Conformant Planning in QBF. AAAI 2007: 1045-1050

Application Domain 2: Games and planning with QBF and Skolem fns

Conformant or Conditional Planning in AI

Rintanen, J. 1999. Constructing conditional plans by a theorem-prover. Journal of Artificial Intelligence Research 10:323-352.

- Given set S of states, I, G, formulas over S defining initial and goal states and a set of non-det actions A,
  - does there exist a plan (seq of actions), s.t., for all possible contingencies (initial states and nondet choices), there exist an execution (seq of states) that reaches the goal state.
  - This is a $\exists\forall\exists$ formula, so in 3-QBF.
  - Can also be encoded as $\forall\exists$. Rintanen, J. 2007. Asymptotically Optimal Encodings of Conformant Planning in QBF. AAAI 2007: 1045-1050

More Planning to QBF approaches:

- Used to reduce size of encoding rather than uncertainty; Arbitrary Quantifier Alternation.

Michael Cashmore, Maria Fox, Enrico Giunchiglia: Partially Grounded Planning as Quantified Boolean Formula. ICAPS 2013

Michael Cashmore, Maria Fox, Enrico Giunchiglia: Planning as Quantified Boolean Formula. ECAI 2012: 217-222.

Application Domain 2: Games and planning with QBF and Skolem fns

Conformant or Conditional Planning in AI

Rintanen, J. 1999. Constructing conditional plans by a theorem-prover. Journal of Artificial Intelligence Research 10:323-352.

- Given set S of states, I, G, formulas over S defining initial and goal states and a set of non-det actions A,
  - does there exist a plan (seq of actions), s.t., for all possible contingencies (initial states and nondet choices), there exist an execution (seq of states) that reaches the goal state.
  - This is a $\exists\forall\exists$ formula, so in 3-QBF.
  - Can also be encoded as $\forall\exists$. Rintanen, J. 2007. Asymptotically Optimal Encodings of Conformant Planning in QBF. AAAI 2007: 1045-1050

More Planning to QBF approaches:

- Used to reduce size of encoding rather than uncertainty; Arbitrary Quantifier Alternation.

Michael Cashmore, Maria Fox, Enrico Giunchiglia: Partially Grounded Planning as Quantified Boolean Formula. ICAPS 2013

Michael Cashmore, Maria Fox, Enrico Giunchiglia: Planning as Quantified Boolean Formula. ECAI 2012: 217-222.

Bottomline

Synthesizing Skolem functions synthesizes the plans in all these cases!

Application Domain 3: Reactive Synthesis

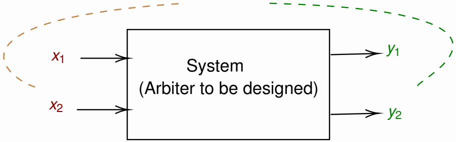
Boolean functional synthesis can help reactive synthesis too!

Application Domain 3: Reactive Synthesis

Boolean functional synthesis can help reactive synthesis too!

Linear Temporal Logic (LTL) spec

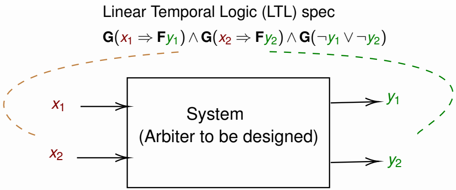
$\mathbf{G}(x_1 \Rightarrow \mathbf{F}y_1) \wedge \mathbf{G}(x_2 \Rightarrow \mathbf{F}y_2) \wedge \mathbf{G}(\neg y_1 \vee \neg y_2)$



- Specification has temporal operators

Application Domain 3: Reactive Synthesis

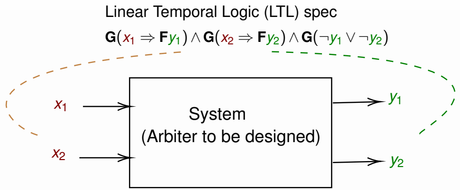
Boolean functional synthesis can help reactive synthesis too!



- Specification has temporal operators
- **G**: at **all** times; **F**: **now** or **in future**

Application Domain 3: Reactive Synthesis

Boolean functional synthesis can help reactive synthesis too!



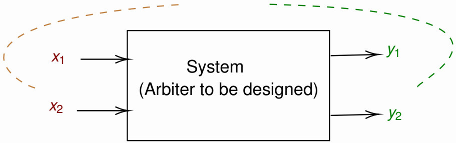
- Specification has temporal operators
- **G**: at **all** times; **F**: **now** or **in future**
- At **all** times
  - If a request comes on x_i, a grant goes on y_i **then or later**.
  - Both grants y_1 and y_2 can't be asserted

Application Domain 3: Reactive Synthesis

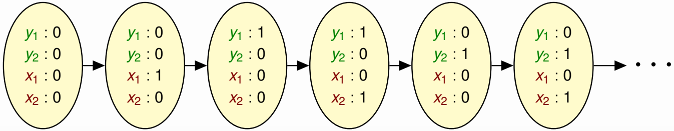
Boolean functional synthesis can help reactive synthesis too!

Linear Temporal Logic (LTL) spec

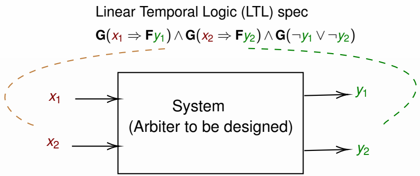
$$\mathbf{G}(x_1 \Rightarrow \mathbf{F}y_1) \wedge \mathbf{G}(x_2 \Rightarrow \mathbf{F}y_2) \wedge \mathbf{G}(\neg y_1 \vee \neg y_2)$$



- Specification has temporal operators
- **G**: at **all** times; **F**: **now** or **in future**
- At **all** times
  - If a request comes on x_i, a grant goes on y_i **then or later**.
  - Both grants y_1 and y_2 can't be asserted
- Relates infinite sequence of **X** and **Y** values



Application Domain 3: Reactive synthesis

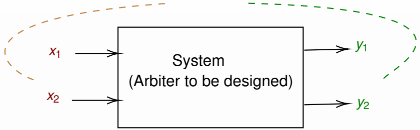


- 2-player game between system and environment
  - System wins if **Y** can be generated to satisfy spec

Application Domain 3: Reactive synthesis

Linear Temporal Logic (LTL) spec

$\mathbf{G}(x_1 \Rightarrow \mathbf{F}y_1) \wedge \mathbf{G}(x_2 \Rightarrow \mathbf{F}y_2) \wedge \mathbf{G}(\neg y_1 \vee \neg y_2)$

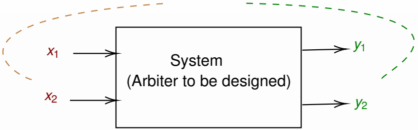


- 2-player game between system and environment
  - System wins if **Y** can be generated to satisfy spec
- Strategy for generating **Y**
  - Winning strategy in two-player game

Application Domain 3: Reactive synthesis

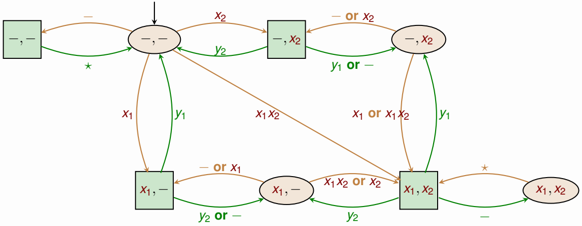
Linear Temporal Logic (LTL) spec

$\mathbf{G}(x_1 \Rightarrow \mathbf{F}y_1) \wedge \mathbf{G}(x_2 \Rightarrow \mathbf{F}y_2) \wedge \mathbf{G}(\neg y_1 \vee \neg y_2)$



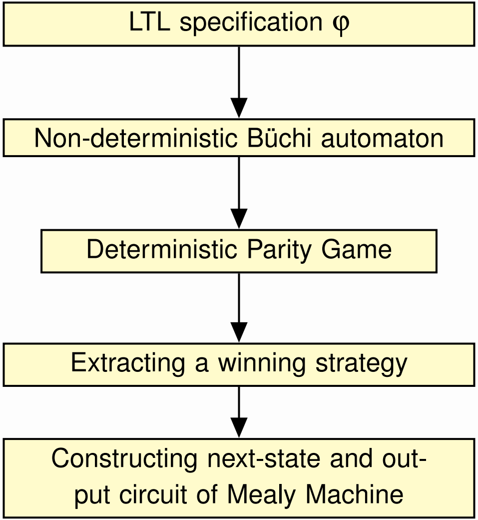
- 2-player game between system and environment
  - System wins if **Y** can be generated to satisfy spec
- Strategy for generating **Y**
  - Winning strategy in two-player game

Game graph:

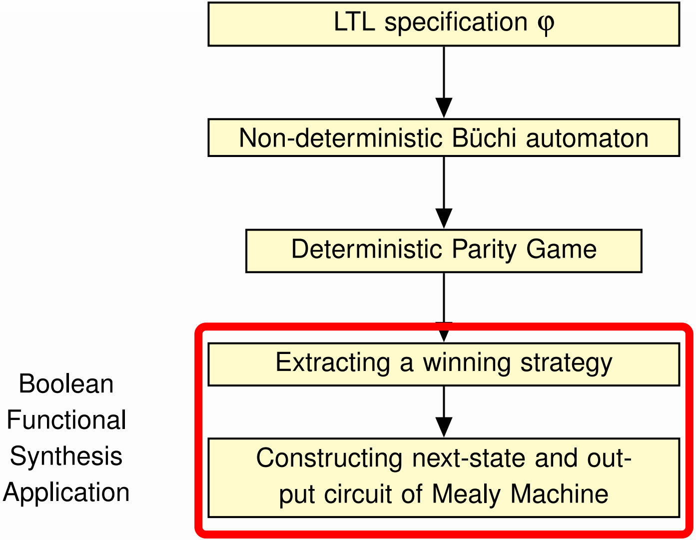


Application Domain 3: Reactive Synthesis

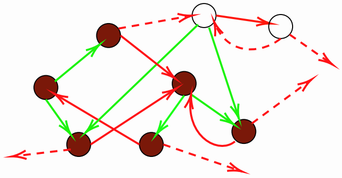
Basic Steps in Synthesis from LTL



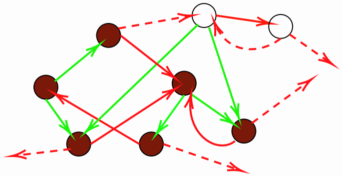
Basic Steps in Synthesis from LTL



Application Domain 3: Winning strategy in reactive synthesis



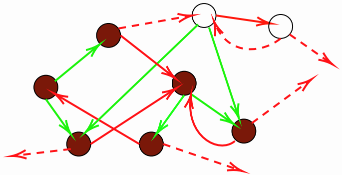
Application Domain 3: Winning strategy in reactive synthesis



Winning Region

- *Winning region* in state transition graph
  - Can always satisfy spec from these states
- Synthesize *winning strategy* to stay within *winning region*

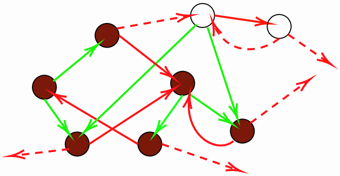
Application Domain 3: Winning strategy in reactive synthesis



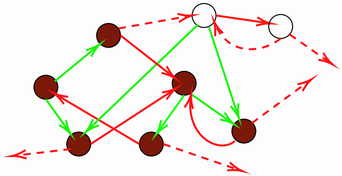
Winning Region

- *Winning region* in state transition graph
  - Can always satisfy spec from these states
- Synthesize *winning strategy* to stay within *winning region*
  - Given a state, if there exists red transition to winning region, choose that
  - ∀ state ∃ Y WinRgn(NxtSt(state, **Y**)) = 1

Application Domain 3: Winning strategy in reactive synthesis



- *Winning region* in state transition graph
  - Can always satisfy spec from these states
- Synthesize *winning strategy* to stay within *winning region*
  - Given a state, if there exists red transition to winning region, choose that
  - ∀ state ∃ Y WinRgn(NxtSt(state, **Y**)) = 1
    - No temporal operators



Winning Region

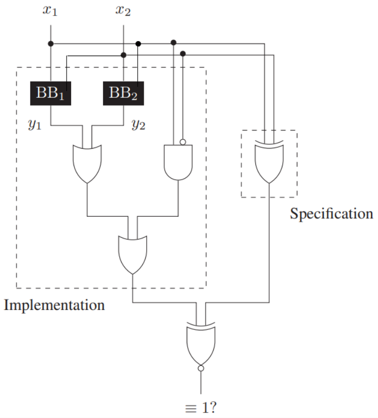
- *Winning region* in state transition graph
  - Can always satisfy spec from these states
- Synthesize *winning strategy* to stay within *winning region*
  - Given a state, if there exists red transition to winning region, choose that
  - ∀ state ∃ Y WinRgn(NxtSt(state, **Y**)) = 1
    - No temporal operators
  - Not always satisfiable

Application Domain 4: Circuit Repair

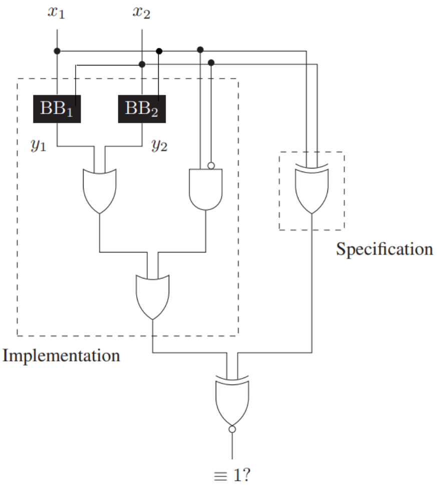
- **Given:** An incomplete implementation and specification.
- **Objective:** Complete the implementation s.t. it is functionally equivalent to specification.

Application Domain 4: Circuit Repair

- **Given:** An incomplete implementation and specification.
- **Objective:** Complete the implementation s.t. it is functionally equivalent to specification.

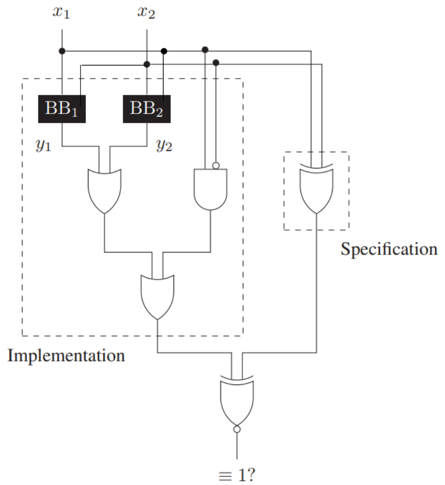


Application Domain 4: Circuit Repair



- Inputs x_1, x_2, Outputs y_1, y_2.
- Synthesise functions(circuits) for y_1, y_2 such that it satisfy the given specification.

Application Domain 4: Circuit Repair



- Inputs x_1, x_2, Outputs y_1, y_2.
- Synthesise functions(circuits) for y_1, y_2 such that it satisfy the given specification.

$$\forall x_1, x_2 \exists y_1 y_2 \neg(((y_1 \vee y_2) \vee (x_1 \wedge \neg x_2)) \oplus (x_1 \oplus x_2))$$

Image is taken(modified) from Equivalence Checking of Partial Designs Using Dependency Quantified Boolean Formulae, Gitina et al '13
Engineering change order for combinational and sequential design rectification, Jiang et. al'20
Synthesis and optimization of multiple portions of circuits for ECO based on set-covering and QBF formulations, Fujita et al'20

Outline

How Hard is Boolean Skolem Function Synthesis?

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

How Hard is Boolean Skolem Function Synthesis?

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

Time complexity

Boolean function synthesis is *NP*-hard

How Hard is Boolean Skolem Function Synthesis?

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

Time complexity

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

How Hard is Boolean Skolem Function Synthesis?

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

Time complexity

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

Space complexity [ACGKS'18]

How Hard is Boolean Skolem Function Synthesis?

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

Time complexity

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

Space complexity [ACGKS'18]

- Unless $\Pi_2^P = \Sigma_2^P$ (i.e., the Polynomial Hierarchy collapses to 2nd level), there exist $\varphi(\mathbf{X}, \mathbf{Y})$ for which Skolem function sizes are super-polynomial in $|\varphi|$.

How Hard is Boolean Skolem Function Synthesis?

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

Time complexity

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

Space complexity [ACGKS'18]

- Unless $\Pi_2^P = \Sigma_2^P$ (i.e., the Polynomial Hierarchy collapses to 2nd level), there exist $\varphi(\mathbf{X}, \mathbf{Y})$ for which Skolem function sizes are super-polynomial in $|\varphi|$.
- Unless non-uniform exponential-time hypothesis fails, there exist $\varphi(\mathbf{X}, \mathbf{Y})$ for which Skolem function sizes are exponential in $|\varphi|$.

How Hard is Boolean Skolem Function Synthesis?

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

Time complexity

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

Space complexity [ACGKS'18]

- Unless $\Pi_2^P = \Sigma_2^P$ (i.e., the Polynomial Hierarchy collapses to 2nd level), there exist $\varphi(\mathbf{X}, \mathbf{Y})$ for which Skolem function sizes are super-polynomial in $|\varphi|$.
- Unless non-uniform exponential-time hypothesis fails, there exist $\varphi(\mathbf{X}, \mathbf{Y})$ for which Skolem function sizes are exponential in $|\varphi|$.

Efficient algorithms for Boolean functional synthesis unlikely

A Survey of Existing Techniques

0. Closely related to most general Boolean unifiers
   Boole'1847, Lowenheim'1908, Macii'98
1. Extract Skolem functions from proof of validity of $\forall \mathbf{X} \exists \mathbf{Y} \varphi(\mathbf{X}, \mathbf{Y})$
   Bendetti'05, Jussilla et al.'07, Balabanov et al.'12, Heule et al.'14
   - Efficient if a short proof of validity is found.
   - Does not work if $\forall \mathbf{X} \exists \mathbf{Y} \varphi(\mathbf{X}, \mathbf{Y})$ is not valid !
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.
   - Does not scale well with an increase in Y variables.

Existing Approaches (Cont.)

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.)

8. Synthesis from special normal forms: The power of Knowledge Compilation!
   - Synthesis negation normal forms (SynNNF)
     Akshay et al.'19
   - The ultimate normal form Shah et al.'21
9. Counter-example guided Skolem function generation
   - Start with over-approximation of Skolem functions + refine
     John et al.'15, Akshay et al.'17,'18,'20
10. Data-driven Skolem function synthesis
    - Machine-learn Skolem function + MaxSat-based iterative repair
      Golia et al.'20, '21

The last two fall into paradigm of Get Skolem function candidate + check + repair

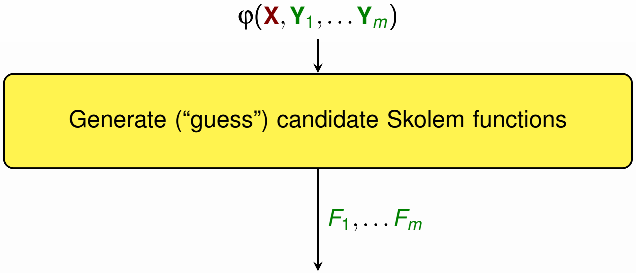
Our focus in the deep-dive: The last three approaches!

Outline

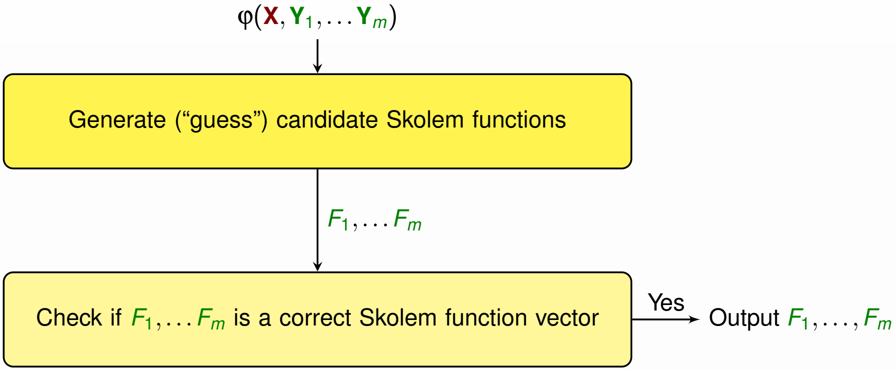
A Guess-Check-Repair Approach

$$\varphi(\mathbf{X}, \mathbf{Y}_1, \ldots \mathbf{Y}_m)$$

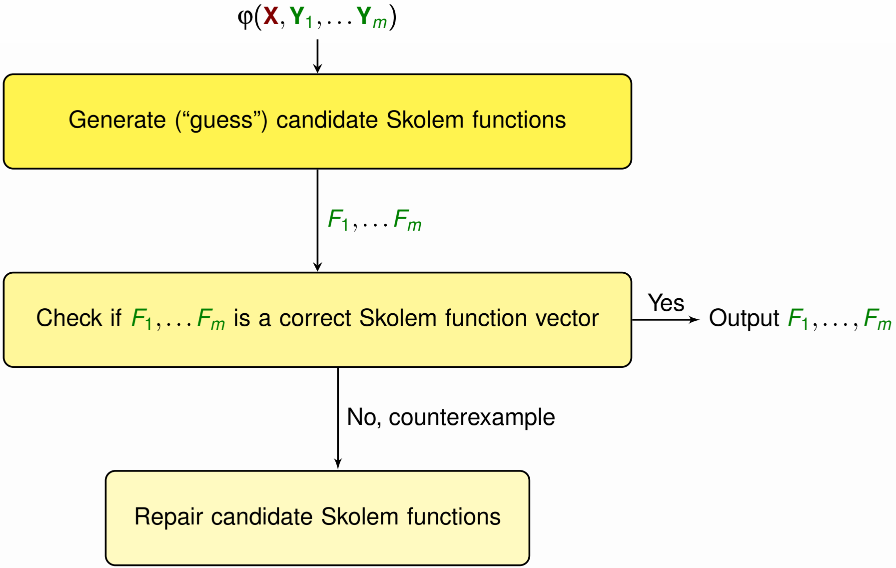
A Guess-Check-Repair Approach



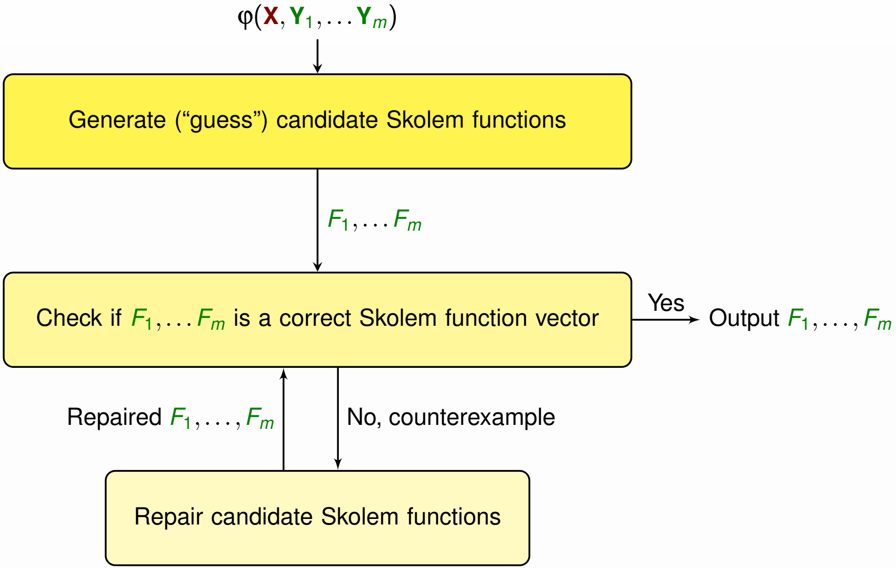
A Guess-Check-Repair Approach



A Guess-Check-Repair Approach



A Guess-Check-Repair Approach

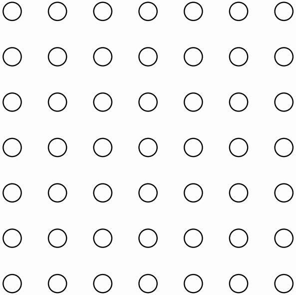


“Guess”-ing candidate Skolem functions ($|\mathbf{Y}| = 1$)

Find $\mathbf{F}(\mathbf{X})$ such that $\exists \mathbf{y}\ \varphi(\mathbf{X},\mathbf{y}) \equiv \varphi(\mathbf{X},\mathbf{F}(\mathbf{X}))$

“Guess”-ing candidate Skolem functions ($|\mathbf{Y}| = 1$)

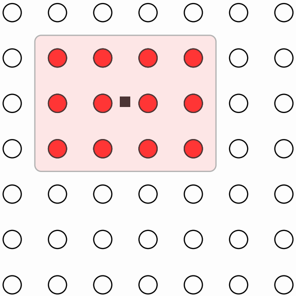
Find $\mathbf{F}(\mathbf{X})$ such that $\exists \mathbf{y}\ \varphi(\mathbf{X}, \mathbf{y}) \equiv \varphi(\mathbf{X}, \mathbf{F}(\mathbf{X}))$



— Set of all valuations of $\mathbf{X}$.

"Guess"-ing candidate Skolem functions ($|\mathbf{Y}| = 1$)

Find $\mathbf{F}(\mathbf{X})$ such that $\exists \mathbf{y}\ \varphi(\mathbf{X}, \mathbf{y}) \equiv \varphi(\mathbf{X}, \mathbf{F}(\mathbf{X}))$

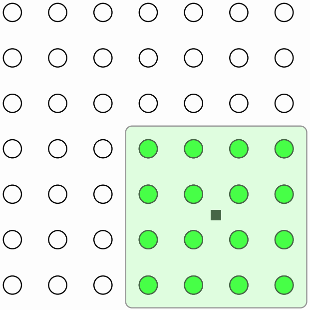


— Can't set $\mathbf{y}$ to 1 to satisfy φ: $\Gamma(\mathbf{X}) \triangleq \neg\varphi(\mathbf{X}, \mathbf{y})[\mathbf{y}1]$

E.g. If $\varphi \equiv (x_1 \vee \mathbf{y}) \wedge (x_1 \vee x_2 \vee \neg\mathbf{y})$, then
$\Gamma(\mathbf{X}) = \neg((x_1 \vee 1) \wedge (x_1 \vee x_2 \vee 0)) = \neg(x_1 \vee x_2) = \neg x_1 \wedge \neg x_2$

“Guess”-ing candidate Skolem functions ($|\mathbf{Y}| = 1$)

Find $\mathbf{F}(\mathbf{X})$ such that $\exists \mathbf{y}\, \varphi(\mathbf{X}, \mathbf{y}) \equiv \varphi(\mathbf{X}, \mathbf{F}(\mathbf{X}))$

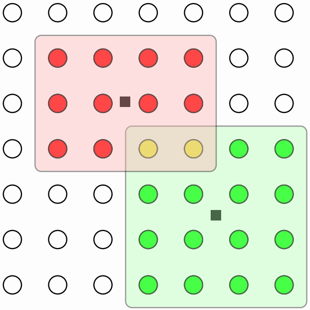


— Can't set $\mathbf{y}$ to 0 to satisfy φ: $\Delta(\mathbf{X}) \triangleq \neg\varphi(\mathbf{X}, \mathbf{y})[\mathbf{y}0]$

E.g. If $\varphi \equiv (x_1 \vee \mathbf{y}) \wedge (x_1 \vee x_2 \vee \neg\mathbf{y})$, then $\Delta(\mathbf{X}) = \neg((x_1 \vee 0) \wedge (x_1 \vee x_2 \vee 1)) = \neg x_1$

“Guess”-ing candidate Skolem functions ($|\mathbf{Y}| = 1$)

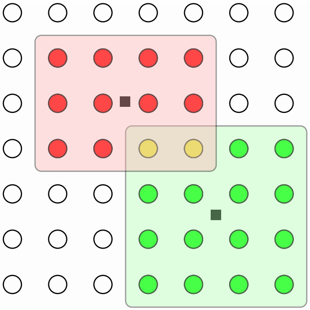
Find $\mathbf{F}(\mathbf{X})$ such that $\exists \mathbf{y}\ \varphi(\mathbf{X},\mathbf{y}) \equiv \varphi(\mathbf{X},\mathbf{F}(\mathbf{X}))$



— Can’t set $\mathbf{y}$ to 1 to satisfy φ: $\Gamma(\mathbf{X}) \triangleq \neg\varphi(\mathbf{X},\mathbf{y})[\mathbf{y}1]$
— Can’t set $\mathbf{y}$ to 0 to satisfy φ: $\Delta(\mathbf{X}) \triangleq \neg\varphi(\mathbf{X},\mathbf{y})[\mathbf{y}0]$

"Guess"-ing candidate Skolem functions ($|\mathbf{Y}| = 1$)

Find $\mathbf{F}(\mathbf{X})$ such that $\exists \mathbf{y}\ \varphi(\mathbf{X}, \mathbf{y}) \equiv \varphi(\mathbf{X}, \mathbf{F}(\mathbf{X}))$



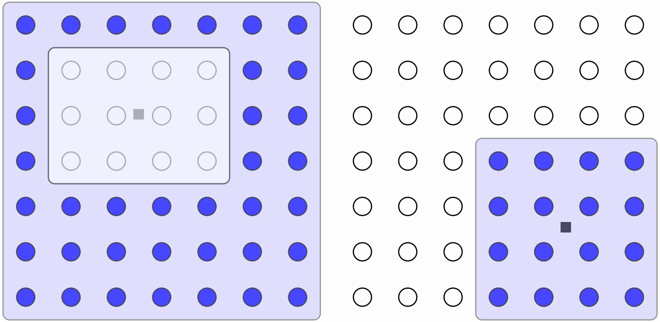
Lemma [Trivedi'03, Jiang'09,Fried et al'16]

Every Skolem function for $\mathbf{y}$ in φ must

- Evaluate to 1 in $(\Delta \setminus \Gamma)$ and to 0 in $(\Gamma \setminus \Delta)$
- Be an **interpolant** of $(\Delta \setminus \Gamma)$ and $(\Gamma \setminus \Delta)$

"Guess"-ing candidate Skolem functions ($|\mathbf{Y}| = 1$)

Find $\mathbf{F}(\mathbf{X})$ such that $\exists \mathbf{y}\ \varphi(\mathbf{X}, \mathbf{y}) \equiv \varphi(\mathbf{X}, \mathbf{F}(\mathbf{X}))$

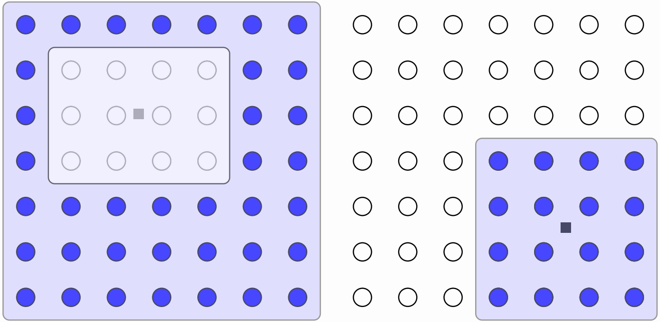


— Specific interpolants of $(\Delta \setminus \Gamma)$ & $(\Gamma \setminus \Delta)$

- $\neg\Gamma \triangleq \varphi(\mathbf{X}, \mathbf{y})[\mathbf{y}1] \equiv \varphi(\mathbf{X}, 1)$
- $\Delta \triangleq \neg\varphi(\mathbf{X}, y)[\mathbf{y}0] \equiv \neg\varphi(\mathbf{X}, 0)$.

“Guess”-ing candidate Skolem functions ($|\mathbf{Y}| = 1$)

Find $\mathbf{F}(\mathbf{X})$ such that $\exists \mathbf{y}\ \varphi(\mathbf{X}, \mathbf{y}) \equiv \varphi(\mathbf{X}, \mathbf{F}(\mathbf{X}))$



— Specific interpolants of $(\Delta \setminus \Gamma)$ & $(\Gamma \setminus \Delta)$

- $\neg\Gamma \triangleq \varphi(\mathbf{X}, \mathbf{y})[\mathbf{y}1] \equiv \varphi(\mathbf{X}, 1)$: Easy solution for 1 output var
- $\Delta \triangleq \neg\varphi(\mathbf{X}, y)[\mathbf{y}0] \equiv \neg\varphi(\mathbf{X}, 0)$.

“Guess”-ing Game: ($|\mathbf{Y}| \geq 2$)

Suppose relational spec is $\varphi(\mathbf{X}, y_1, \boxed{\mathbf{Y}_{2..m}})$

"Guess"-ing Game: ($|\mathbf{Y}| \geq 2$)

Suppose relational spec is $\varphi(\mathbf{X}, y_1, \boxed{\mathbf{Y}_{2..m}})$

- Skolem function for $\boxed{\mathbf{Y}_{2..m}}$ depends on that for y_1 in general

"Guess"-ing Game: ($|\mathbf{Y}| \geq 2$)

Suppose relational spec is $\varphi(\mathbf{X}, y_1, \boxed{\mathbf{Y}_{2..m}})$

- Skolem function for $\boxed{\mathbf{Y}_{2..m}}$ depends on that for y_1 in general
- For what values of $\mathbf{X}$ can we not set y_1 to 1 (or 0)?
  - $\Gamma^{y_1}(\mathbf{X}) = \neg\exists\, \mathbf{Y}_{2..m}\ \varphi(\mathbf{X}, 1, \boxed{\mathbf{Y}_{2..m}})$
  - $\Delta^{y_1}(\mathbf{X}) = \neg\exists\, \mathbf{Y}_{2..m}\ \varphi(\mathbf{X}, 0, \boxed{\mathbf{Y}_{2..m}})$

“Guess”-ing Game: ($|\mathbf{Y}| \geq 2$)

Suppose relational spec is $\varphi(\mathbf{X}, y_1, \boxed{\mathbf{Y}_{2..m}})$

- Skolem function for $\boxed{\mathbf{Y}_{2..m}}$ depends on that for y_1 in general
- For what values of $\mathbf{X}$ can we not set y_1 to 1 (or 0)?
  - $\Gamma^{y_1}(\mathbf{X}) = \neg\exists\, \mathbf{Y}_{2..m}\; \varphi(\mathbf{X}, 1, \boxed{\mathbf{Y}_{2..m}})$
  - $\Delta^{y_1}(\mathbf{X}) = \neg\exists\, \mathbf{Y}_{2..m}\; \varphi(\mathbf{X}, 0, \boxed{\mathbf{Y}_{2..m}})$
- From $\Gamma^{y_1}(\mathbf{X})$ and $\Delta^{y_1}(\mathbf{X})$, find Skolem function $F_1(\mathbf{X})$ for y_1 What if calculating $\exists$ is expensive?

"Guess"-ing Game: ($|\mathbf{Y}| \geq 2$)

Suppose relational spec is $\varphi(\mathbf{X}, y_1, \boxed{\mathbf{Y}_{2..m}})$

- Skolem function for $\boxed{\mathbf{Y}_{2..m}}$ depends on that for y_1 in general
- For what values of $\mathbf{X}$ can we not set y_1 to 1 (or 0)?
  - $\Gamma^{y_1}(\mathbf{X}) = \neg\exists\, \mathbf{Y}_{2..m}\; \varphi(\mathbf{X}, 1, \boxed{\mathbf{Y}_{2..m}})$
  - $\Delta^{y_1}(\mathbf{X}) = \neg\exists\, \mathbf{Y}_{2..m}\; \varphi(\mathbf{X}, 0, \boxed{\mathbf{Y}_{2..m}})$
- From $\Gamma^{y_1}(\mathbf{X})$ and $\Delta^{y_1}(\mathbf{X})$, find Skolem function $F_1(\mathbf{X})$ for y_1
- To find Skolem function for y_2, consider
  - "Simplified" spec $\varphi_1(\mathbf{X}, y_2, \boxed{\mathbf{Y}_{3..m}}) = \varphi(\mathbf{X}, F_1(\mathbf{X}), y_2, \boxed{\mathbf{Y}_{3..m}})$

"Guess"-ing Game: ($|\mathbf{Y}| \geq 2$)

Suppose relational spec is $\varphi(\mathbf{X}, y_1, \boxed{\mathbf{Y}_{2..m}})$

- Skolem function for $\boxed{\mathbf{Y}_{2..m}}$ depends on that for y_1 in general
- For what values of $\mathbf{X}$ can we not set y_1 to 1 (or 0)?
  - $\Gamma^{y_1}(\mathbf{X}) = \neg\exists\, \mathbf{Y}_{2..m}\ \varphi(\mathbf{X}, 1, \boxed{\mathbf{Y}_{2..m}})$
  - $\Delta^{y_1}(\mathbf{X}) = \neg\exists\, \mathbf{Y}_{2..m}\ \varphi(\mathbf{X}, 0, \boxed{\mathbf{Y}_{2..m}})$
- From $\Gamma^{y_1}(\mathbf{X})$ and $\Delta^{y_1}(\mathbf{X})$, find Skolem function $F_1(\mathbf{X})$ for y_1
- To find Skolem function for y_2, consider
  - "Simplified" spec $\varphi_1(\mathbf{X}, y_2, \boxed{\mathbf{Y}_{3..m}}) = \varphi(\mathbf{X}, F_1(\mathbf{X}), y_2, \boxed{\mathbf{Y}_{3..m}})$
  - Repeat above steps ...

"Guess"-ing Game: ($|\mathbf{Y}| \geq 2$)

Suppose relational spec is $\varphi(\mathbf{X}, y_1, \boxed{\mathbf{Y}_{2..m}})$

- Skolem function for $\boxed{\mathbf{Y}_{2..m}}$ depends on that for y_1 in general
- For what values of $\mathbf{X}$ can we not set y_1 to 1 (or 0)?
  - $\Gamma^{y_1}(\mathbf{X}) = \neg\exists\, \mathbf{Y}_{2..m}\ \varphi(\mathbf{X}, 1, \boxed{\mathbf{Y}_{2..m}})$
  - $\Delta^{y_1}(\mathbf{X}) = \neg\exists\, \mathbf{Y}_{2..m}\ \varphi(\mathbf{X}, 0, \boxed{\mathbf{Y}_{2..m}})$
- From $\Gamma^{y_1}(\mathbf{X})$ and $\Delta^{y_1}(\mathbf{X})$, find Skolem function $F_1(\mathbf{X})$ for y_1
- To find Skolem function for y_2, consider
  - "Simplified" spec $\varphi_1(\mathbf{X}, y_2, \boxed{\mathbf{Y}_{3..m}}) = \varphi(\mathbf{X}, F_1(\mathbf{X}), y_2, \boxed{\mathbf{Y}_{3..m}})$
  - Repeat above steps ...

Are we done?

“Guess”-ing Game: ($|\mathbf{Y}| \geq 2$)

Suppose relational spec is $\varphi(\mathbf{X}, y_1, \boxed{\mathbf{Y}_{2..m}})$

- Skolem function for $\boxed{\mathbf{Y}_{2..m}}$ depends on that for y_1 in general
- For what values of $\mathbf{X}$ can we not set y_1 to 1 (or 0)?
  - $\Gamma^{y_1}(\mathbf{X}) = \neg\exists\, \mathbf{Y}_{2..m}\ \varphi(\mathbf{X}, 1, \boxed{\mathbf{Y}_{2..m}})$
  - $\Delta^{y_1}(\mathbf{X}) = \neg\exists\, \mathbf{Y}_{2..m}\ \varphi(\mathbf{X}, 0, \boxed{\mathbf{Y}_{2..m}})$
- From $\Gamma^{y_1}(\mathbf{X})$ and $\Delta^{y_1}(\mathbf{X})$, find Skolem function $F_1(\mathbf{X})$ for y_1

  What if calculating $\exists\, \mathbf{Y}_{2..m}\ \varphi(\mathbf{X}, y_1, \boxed{\mathbf{Y}_{2..m}})$ is expensive?

- To find Skolem function for y_2, consider
  - “Simplified” spec $\varphi_1(\mathbf{X}, y_2, \boxed{\mathbf{Y}_{3..m}}) = \varphi(\mathbf{X}, F_1(\mathbf{X}), y_2, \boxed{\mathbf{Y}_{3..m}})$
  - Repeat above steps ...

Are we done?

“Guess”-ing Game: ($|\mathbf{Y}| \geq 2$)

Suppose relational spec is $\varphi(\mathbf{X}, y_1, \boxed{\mathbf{Y}_{2..m}})$

- Skolem function for $\boxed{\mathbf{Y}_{2..m}}$ depends on that for y_1 in general
- For what values of $\mathbf{X}$ can we not set y_1 to 1 (or 0)?
  - $\Gamma^{y_1}(\mathbf{X}) = \neg\exists\, \mathbf{Y}_{2..m}\ \varphi(\mathbf{X}, 1, \boxed{\mathbf{Y}_{2..m}})$
  - $\Delta^{y_1}(\mathbf{X}) = \neg\exists\, \mathbf{Y}_{2..m}\ \varphi(\mathbf{X}, 0, \boxed{\mathbf{Y}_{2..m}})$
- From $\Gamma^{y_1}(\mathbf{X})$ and $\Delta^{y_1}(\mathbf{X})$, find Skolem function $F_1(\mathbf{X})$ for y_1

  What if calculating $\exists\, \mathbf{Y}_{2..m}\ \varphi(\mathbf{X}, y_1, \boxed{\mathbf{Y}_{2..m}})$ is expensive?

  - Use easily computed approx of $\exists\, \mathbf{Y}_{2..m}\ \varphi(\mathbf{X}, y_1, \boxed{\mathbf{Y}_{2..m}})$?
  - “Guess” $G_1(\mathbf{X})$ as approx of Skolem function $F_1(\mathbf{X})$?
  - Repair “guess” if needed
- To find Skolem function for y_2, consider
  - “Simplified” spec $\varphi_1(\mathbf{X}, y_2, \boxed{\mathbf{Y}_{3..m}}) = \varphi(\mathbf{X}, F_1(\mathbf{X}), y_2, \boxed{\mathbf{Y}_{3..m}})$
  - Repeat above steps ...

Are we done?

General Idea

Linearly order outputs: $y_1 \prec y_2 \prec \cdots \prec y_m$

General Idea

Linearly order outputs: $y_1 \prec y_2 \prec \cdots \prec y_m$
Express

- y_m as $G_m(\mathbf{X}, x_1, \ldots x_{m-1})$ from spec $\varphi(\mathbf{X}, x_1, \ldots x_{m-1}, y_m)$

General Idea

Linearly order outputs: $y_1 \prec y_2 \prec \cdots \prec y_m$
Express

- y_m as $G_m(\mathbf{X}, x_1, \ldots x_{m-1})$ from spec $\varphi(\mathbf{X}, x_1, \ldots x_{m-1}, y_m)$
- y_{m-1} as $G_{m-1}(\mathbf{X}, x_1, \ldots x_{m-2})$ from $\exists y_m\, \varphi(\mathbf{X}, x_1, \ldots x_{m-2}, y_{m-1}, y_m)$
- $\vdots$

General Idea

Linearly order outputs: $y_1 \prec y_2 \prec \cdots \prec y_m$
Express

- y_m as $G_m(\mathbf{X}, x_1, \ldots x_{m-1})$ from spec $\varphi(\mathbf{X}, x_1, \ldots x_{m-1}, y_m)$
- y_{m-1} as $G_{m-1}(\mathbf{X}, x_1, \ldots x_{m-2})$ from $\exists y_m \, \varphi(\mathbf{X}, x_1, \ldots x_{m-2}, y_{m-1}, y_m)$
- $\vdots$
- y_1 as $G_1(\mathbf{X})$ from $\exists y_2 \ldots \exists y_m \, \varphi(\mathbf{X}, y_1, y_2 \ldots y_m)$

General Idea

Linearly order outputs: $y_1 \prec y_2 \prec \cdots \prec y_m$
Express

- y_m as $G_m(\mathbf{X}, x_1, \ldots x_{m-1})$ from spec $\varphi(\mathbf{X}, x_1, \ldots x_{m-1}, y_m)$
- y_{m-1} as $G_{m-1}(\mathbf{X}, x_1, \ldots x_{m-2})$ from $\exists y_m \, \varphi(\mathbf{X}, x_1, \ldots x_{m-2}, y_{m-1}, y_m)$
- $\vdots$
- y_1 as $G_1(\mathbf{X})$ from $\exists y_2 \ldots \exists y_m \, \varphi(\mathbf{X}, y_1, y_2 \ldots y_m)$

Key Steps

- Generate Skolem functions for 1-output spec
- Compute (approximations of) $\exists y_i \ldots y_m \, \varphi(\mathbf{X}, \mathbf{Y})$

General Idea

Linearly order outputs: $y_1 \prec y_2 \prec \cdots \prec y_m$
Express

- y_m as $G_m(\mathbf{X}, x_1, \ldots x_{m-1})$ from spec $\varphi(\mathbf{X}, x_1, \ldots x_{m-1}, y_m)$
- y_{m-1} as $G_{m-1}(\mathbf{X}, x_1, \ldots x_{m-2})$ from $\exists y_m\, \varphi(\mathbf{X}, x_1, \ldots x_{m-2}, y_{m-1}, y_m)$
- $\vdots$
- y_1 as $G_1(\mathbf{X})$ from $\exists y_2 \ldots \exists y_m\, \varphi(\mathbf{X}, y_1, y_2 \ldots y_m)$

Key Steps

- Generate Skolem functions for 1-output spec
- Compute (approximations of) $\exists y_i \ldots y_m\, \varphi(\mathbf{X}, \mathbf{Y})$

If all guesses correct, a $|\mathbf{X}|$-input, $|\mathbf{Y}|$-output circuit computing the desired Skolem function vector $(F_1, \ldots F_m)$ can be constructed with

- #gates $\leq \sum_{i=1}^{m}$ #gates(G_i) $+2m$
- #wires $\leq \sum_{i=1}^{m}$ #wires(G_i) $+\frac{m(m-1)}{2}$

General Idea

Linearly order outputs: $y_1 \prec y_2 \prec \cdots \prec y_m$

Express

- y_m as $G_m(\mathbf{X}, x_1, \ldots x_{m-1})$ from spec $\varphi(\mathbf{X}, x_1, \ldots x_{m-1}, y_m)$
- y_{m-1} as $G_{m-1}(\mathbf{X}, x_1, \ldots x_{m-2})$ from $\exists y_m\, \varphi(\mathbf{X}, x_1, \ldots x_{m-2}, y_{m-1}, y_m)$
- $\vdots$
- y_1 as $G_1(\mathbf{X})$ from $\exists y_2 \ldots \exists y_m\, \varphi(\mathbf{X}, y_1, y_2 \ldots y_m)$

Key Steps

- Generate Skolem functions for 1-output spec
- Compute (approximations of) $\exists y_i \ldots y_m\, \varphi(\mathbf{X}, \mathbf{Y})$

If all guesses correct, a $|\mathbf{X}|$-input, $|\mathbf{Y}|$-output circuit computing the desired Skolem function vector $(F_1, \ldots F_m)$ can be constructed with

- #gates $\leq \sum_{i=1}^{m}$ #gates(G_i) $+2m$
- #wires $\leq \sum_{i=1}^{m}$ #wires(G_i) $+\frac{m(m-1)}{2}$

Sufficient to compute the G_i functions

Dealing with Existential Quantification

- Compute $\exists y_i \ldots \exists y_m \varphi(\mathbf{X}, x_1, \ldots x_{i-2}, y_{i-1}, y_i, \ldots y_m)$

Dealing with Existential Quantification

- Compute $\exists y_i \ldots \exists y_m \varphi(\mathbf{X}, x_1, \ldots x_{i-2}, y_{i-1}, y_i, \ldots y_m)$
  - Hard in general

Dealing with Existential Quantification

- Compute $\exists y_i \ldots \exists y_m \varphi(\mathbf{X}, x_1, \ldots x_{i-2}, y_{i-1}, y_i, \ldots y_m)$
  - Hard in general
  - Can we use some efficiently computable approximations?

Dealing with Existential Quantification

- Compute $\exists y_i \ldots \exists y_m \varphi(\mathbf{X}, x_1, \ldots x_{i-2}, y_{i-1}, y_i, \ldots y_m)$
  - Hard in general
  - Can we use some efficiently computable approximations?

Represent $\varphi(x_1, .., x_n, y_1, .., y_m)$ as NNF DAG

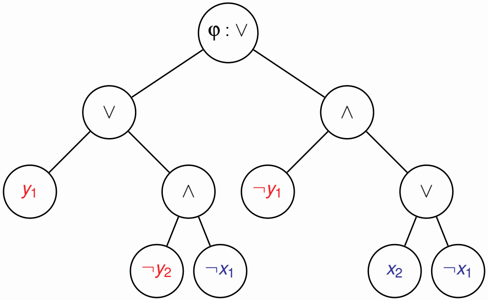
- Boolean circuit, $\wedge$ and $\vee$ internal nodes, $\neg$ at leaves

Dealing with Existential Quantification

- Compute $\exists y_i \ldots \exists y_m \varphi(\mathbf{X}, x_1, \ldots x_{i-2}, y_{i-1}, y_i, \ldots y_m)$
  - Hard in general
  - Can we use some efficiently computable approximations?

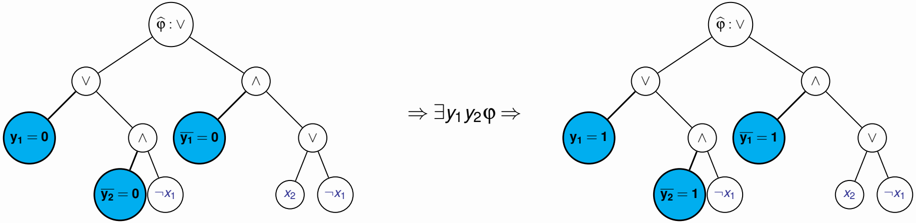
Represent $\varphi(x_1, .., x_n, y_1, .., y_m)$ as NNF DAG

- Boolean circuit, $\wedge$ and $\vee$ internal nodes, $\neg$ at leaves



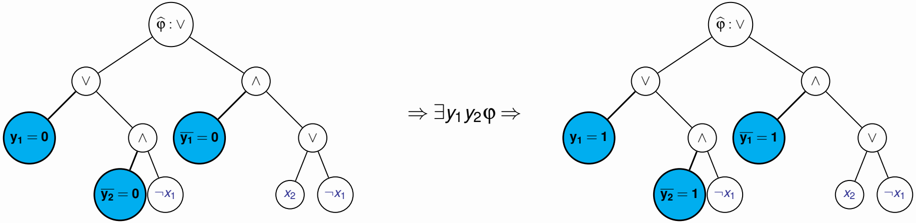
Illustrating Approximations

Replace $\neg y_i$ at leaves with fresh variables $\overline{y_i}$ and call the "new" formula $\widehat{\varphi}$.



Illustrating Approximations

Replace $\neg y_i$ at leaves with fresh variables $\overline{y_i}$ and call the “new” formula $\widehat{\varphi}$.



- $\widehat{\varphi}(x_1 \ldots x_n, \overbrace{0..0}^{i}, y_{i+1} \ldots y_m, \overbrace{0..0}^{i}, \neg y_{i+1} \ldots \neg y_m) \Rightarrow \exists y_1 \ldots y_i\ \varphi(\ldots)$
- $\widehat{\varphi}(x_1 \ldots x_n, \overbrace{1..1}^{i}, y_{i+1} \ldots y_m, \overbrace{1..1}^{i}, \neg y_{i+1} \ldots \neg y_m) \Leftarrow \exists y_1 \ldots y_i\ \varphi(\ldots)$

Checking correctness of "guess"-ed Skolem functions

Given candidate Skolem functions $F_1, \ldots F_m$,

$$\text{Is } \forall \mathbf{X} \left(\exists \mathbf{Y} \varphi(\mathbf{X}, \mathbf{Y}) \Leftrightarrow \varphi(\mathbf{X}, \mathbf{F}(\mathbf{X}) \right) \text{ ?}$$

Checking correctness of “guess”-ed Skolem functions

Given candidate Skolem functions $F_1, \ldots F_m$,

$$\text{Is } \forall \mathbf{X} \big(\exists \mathbf{Y} \varphi(\mathbf{X}, \mathbf{Y}) \Leftrightarrow \varphi(\mathbf{X}, \mathbf{F}(\mathbf{X})) \big) \text{ ?}$$

Can we avoid using a QBF solver?

Checking correctness of "guess"-ed Skolem functions

Given candidate Skolem functions $F_1, \ldots F_m$,

$$\text{Is } \forall \mathbf{X}\left(\exists \mathbf{Y} \varphi(\mathbf{X}, \mathbf{Y}) \;\Leftrightarrow\; \varphi(\mathbf{X}, \mathbf{F}(\mathbf{X}) \right) \text{ ?}$$

Can we avoid using a QBF solver?

Yes, we can! [ACGKS'15]

- Propositional error formula $\varepsilon(\mathbf{X}, \mathbf{Y}, \mathbf{Y}')$:

$$\left(\varphi(\mathbf{X}, \mathbf{Y}') \wedge \bigwedge_{j=1}^{m} (\mathbf{Y}_j \Leftrightarrow F_j) \wedge \neg\varphi(\mathbf{X}, \mathbf{Y})\right)$$

Checking correctness of “guess”-ed Skolem functions

Given candidate Skolem functions $F_1, \ldots F_m$,

$$\text{Is } \forall \mathbf{X} \left(\exists \mathbf{Y} \varphi(\mathbf{X}, \mathbf{Y}) \;\Leftrightarrow\; \varphi(\mathbf{X}, \mathbf{F}(\mathbf{X})) \right) \text{ ?}$$

Can we avoid using a QBF solver?

Yes, we can! [ACGKS’15]

- Propositional error formula $\varepsilon(\mathbf{X}, \mathbf{Y}, \mathbf{Y}')$:

$$\left(\varphi(\mathbf{X}, \mathbf{Y}') \wedge \bigwedge_{j=1}^{m} (\mathbf{Y}_j \Leftrightarrow F_j) \wedge \neg\varphi(\mathbf{X}, \mathbf{Y})\right)$$

- ε unsatisfiable iff $F_1, \ldots F_m$ is correct Skolem function vector

Checking correctness of "guess"-ed Skolem functions

Given candidate Skolem functions $F_1, \ldots F_m$,

$$\text{Is } \forall \mathbf{X} \left(\exists \mathbf{Y} \varphi(\mathbf{X}, \mathbf{Y}) \Leftrightarrow \varphi(\mathbf{X}, \mathbf{F}(\mathbf{X})\right) ?$$

Can we avoid using a QBF solver?

Yes, we can! [ACGKS'15]

- Propositional error formula $\varepsilon(\mathbf{X}, \mathbf{Y}, \mathbf{Y}')$:

$$\left(\varphi(\mathbf{X}, \mathbf{Y}') \wedge \bigwedge_{j=1}^{m} (\mathbf{Y}_j \Leftrightarrow F_j) \wedge \neg\varphi(\mathbf{X}, \mathbf{Y})\right)$$

- ε unsatisfiable iff $F_1, \ldots F_m$ is correct Skolem function vector
- Suppose σ: satisfying assignment of ε

Checking correctness of "guess"-ed Skolem functions

Given candidate Skolem functions $F_1, \ldots F_m$,

$$\text{Is } \forall \mathbf{X}\left(\exists \mathbf{Y} \varphi(\mathbf{X}, \mathbf{Y}) \Leftrightarrow \varphi(\mathbf{X}, \mathbf{F}(\mathbf{X})\right) ?$$

Can we avoid using a QBF solver?

Yes, we can! [ACGKS'15]

- Propositional error formula $\varepsilon(\mathbf{X}, \mathbf{Y}, \mathbf{Y}')$:

$$\left(\varphi(\mathbf{X}, \mathbf{Y}') \wedge \bigwedge_{j=1}^{m} (\mathbf{Y}_j \Leftrightarrow F_j) \wedge \neg\varphi(\mathbf{X}, \mathbf{Y})\right)$$

- ε unsatisfiable iff $F_1, \ldots F_m$ is correct Skolem function vector
- Suppose σ: satisfying assignment of ε
  - $\varphi(\sigma[\mathbf{X}], \sigma[\mathbf{Y}'])] = 1, \quad \sigma[\mathbf{Y}] = \mathbf{F}(\sigma[\mathbf{X}]), \quad \varphi(\sigma[\mathbf{X}], \sigma[\mathbf{Y}]) = 0$

Checking correctness of "guess"-ed Skolem functions

Given candidate Skolem functions $F_1, \ldots F_m$,

$$\text{Is } \forall \mathbf{X} \left(\exists \mathbf{Y} \varphi(\mathbf{X}, \mathbf{Y}) \Leftrightarrow \varphi(\mathbf{X}, \mathbf{F}(\mathbf{X})) \right) ?$$

Can we avoid using a QBF solver?

Yes, we can! [ACGKS'15]

- Propositional error formula $\varepsilon(\mathbf{X}, \mathbf{Y}, \mathbf{Y}')$:

$$\left(\varphi(\mathbf{X}, \mathbf{Y}') \wedge \bigwedge_{j=1}^{m} (\mathbf{Y}_j \Leftrightarrow F_j) \wedge \neg\varphi(\mathbf{X}, \mathbf{Y})\right)$$

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

Counterexample Generalization

Recall: Skolem functions guessed from approximations of

$\exists y_{i+1} \ldots \exists y_m \, \varphi(\mathbf{X}, x_1, \ldots x_{i-1}, y_i, y_{i+1}, \ldots y_m)$

Counterexample Generalization

Recall: Skolem functions guessed from approximations of
$\exists y_{i+1} \ldots \exists y_m \, \varphi(\mathbf{X}, x_1, \ldots x_{i-1}, y_i, y_{i+1}, \ldots y_m)$

- Let $\exists y_{i+1} \ldots \exists y_m \, \varphi(\mathbf{X}, \mathbf{Y}) \Rightarrow \Theta_i(\mathbf{X}, x_1, \ldots x_{i-1}, y_i)$ Over-approx

Counterexample Generalization

Recall: Skolem functions guessed from approximations of $\exists y_{i+1} \ldots \exists y_m \varphi(\mathbf{X}, x_1, \ldots x_{i-1}, y_i, y_{i+1}, \ldots y_m)$

- Let $\exists y_{i+1} \ldots \exists y_m \varphi(\mathbf{X}, \mathbf{Y}) \Rightarrow \Theta_i(\mathbf{X}, x_1, \ldots x_{i-1}, y_i)$
- Let $\delta_i = \neg\Theta_i|_{y_i=0}$; $\gamma_i = \neg\Theta_i|_{y_i=1}$

Under-approximations

Counterexample Generalization

Recall: Skolem functions guessed from approximations of $\exists y_{i+1} \ldots \exists y_m \varphi(\mathbf{X}, x_1, \ldots x_{i-1}, y_i, y_{i+1}, \ldots y_m)$

- Let $\exists y_{i+1} \ldots \exists y_m \varphi(\mathbf{X}, \mathbf{Y}) \Rightarrow \Theta_i(\mathbf{X}, x_1, \ldots x_{i-1}, y_i)$
- Let $\delta_i = \neg\Theta_i|_{y_i=0}; \quad \gamma_i = \neg\Theta_i|_{y_i=1}$
- Initial guess $G_i(\mathbf{X}, x_1, \ldots x_{i-1}) \in \{\delta_i, \neg\gamma_i\}$

Counterexample Generalization

Recall: Skolem functions guessed from approximations of $\exists y_{i+1} \ldots \exists y_m \, \varphi(\mathbf{X}, x_1, \ldots x_{i-1}, y_i, y_{i+1}, \ldots y_m)$

- Let $\exists y_{i+1} \ldots \exists y_m \, \varphi(\mathbf{X}, \mathbf{Y}) \Rightarrow \Theta_i(\mathbf{X}, x_1, \ldots x_{i-1}, y_i)$
- Let $\delta_i = \neg\Theta_i|_{y_i=0}$; $\quad \gamma_i = \neg\Theta_i|_{y_i=1}$
- Initial guess $G_i(\mathbf{X}, x_1, \ldots x_{i-1}) \in \{\delta_i, \neg\gamma_i\}$ **... 1-sided error**
  - $G_i = \delta_i$ **cannot err** if it evaluates to 1
  - $G_i = \neg\gamma_i$ **cannot err** if it evaluates to 0

Counterexample Generalization

Recall: Skolem functions guessed from approximations of

$\exists y_{i+1} \ldots \exists y_m \varphi(\mathbf{X}, x_1, \ldots x_{i-1}, y_i, y_{i+1}, \ldots y_m)$

- Let $\exists y_{i+1} \ldots \exists y_m \varphi(\mathbf{X}, \mathbf{Y}) \Rightarrow \Theta_i(\mathbf{X}, x_1, \ldots x_{i-1}, y_i)$
- Let $\delta_i = \neg\Theta_i|_{y_i=0}$; $\quad \gamma_i = \neg\Theta_i|_{y_i=1}$
- Initial guess $G_i(\mathbf{X}, x_1, \ldots x_{i-1}) \in \{\delta_i, \neg\gamma_i\}$ **... 1-sided error**
  - $G_i = \delta_i$ **cannot err** if it evaluates to 1
  - $G_i = \neg\gamma_i$ **cannot err** if it evaluates to 0

Generalized counterexample

Given $\sigma \models \varepsilon(\mathbf{X}, \mathbf{Y}, \mathbf{Y}')$ and δ_i, γ_i for $1 \leq i \leq m$

Counterexample Generalization

Recall: Skolem functions guessed from approximations of $\exists y_{i+1} \ldots \exists y_m \, \varphi(\mathbf{X}, x_1, \ldots x_{i-1}, y_i, y_{i+1}, \ldots y_m)$

- Let $\exists y_{i+1} \ldots \exists y_m \, \varphi(\mathbf{X}, \mathbf{Y}) \Rightarrow \Theta_i(\mathbf{X}, x_1, \ldots x_{i-1}, y_i)$
- Let $\delta_i = \neg \Theta_i|_{y_i=0}$; $\quad \gamma_i = \neg \Theta_i|_{y_i=1}$
- Initial guess $G_i(\mathbf{X}, x_1, \ldots x_{i-1}) \in \{\delta_i, \neg\gamma_i\}$ **... 1-sided error**
  - $G_i = \delta_i$ **cannot err** if it evaluates to 1
  - $G_i = \neg\gamma_i$ **cannot err** if it evaluates to 0

Generalized counterexample

Given $\sigma \models \varepsilon(\mathbf{X}, \mathbf{Y}, \mathbf{Y}')$ and δ_i, γ_i for $1 \leq i \leq m$

Find function $\mu(\mathbf{X}, x_1, \ldots x_{j-1})$ for some $j \in \{1, \ldots m\}$ s.t.

Counterexample Generalization

Recall: Skolem functions guessed from approximations of

$\exists y_{i+1} \ldots \exists y_m \varphi(\mathbf{X}, x_1, \ldots x_{i-1}, y_i, y_{i+1}, \ldots y_m)$

- Let $\exists y_{i+1} \ldots \exists y_m \varphi(\mathbf{X}, \mathbf{Y}) \Rightarrow \Theta_i(\mathbf{X}, x_1, \ldots x_{i-1}, y_i)$
- Let $\delta_i = \neg\Theta_i|_{y_i=0}$; $\gamma_i = \neg\Theta_i|_{y_i=1}$
- Initial guess $G_i(\mathbf{X}, x_1, \ldots x_{i-1}) \in \{\delta_i, \neg\gamma_i\}$ **... 1-sided error**
  - $G_i = \delta_i$ **cannot err** if it evaluates to 1
  - $G_i = \neg\gamma_i$ **cannot err** if it evaluates to 0

Generalized counterexample

Given $\sigma \models \varepsilon(\mathbf{X}, \mathbf{Y}, \mathbf{Y}')$ and δ_i, γ_i for $1 \leq i \leq m$

Find function $\mu(\mathbf{X}, x_1, \ldots x_{j-1})$ for some $j \in \{1, \ldots m\}$ s.t.

- $\sigma \models \mu$ **... μ generalizes σ**

Counterexample Generalization

Recall: Skolem functions guessed from approximations of

$\exists y_{i+1} \dots \exists y_m \, \varphi(\mathbf{X}, x_1, \dots x_{i-1}, y_i, y_{i+1}, \dots y_m)$

- Let $\exists y_{i+1} \dots \exists y_m \, \varphi(\mathbf{X}, \mathbf{Y}) \Rightarrow \Theta_i(\mathbf{X}, x_1, \dots x_{i-1}, y_i)$
- Let $\delta_i = \neg \Theta_i|_{y_i=0}; \quad \gamma_i = \neg \Theta_i|_{y_i=1}$
- Initial guess $G_i(\mathbf{X}, x_1, \dots x_{i-1}) \in \{\delta_i, \neg\gamma_i\}$ **... 1-sided error**
  - $G_i = \delta_i$ **cannot err** if it evaluates to 1
  - $G_i = \neg\gamma_i$ **cannot err** if it evaluates to 0

Generalized counterexample

Given $\sigma \models \varepsilon(\mathbf{X}, \mathbf{Y}, \mathbf{Y'})$ and δ_i, γ_i for $1 \leq i \leq m$
Find function $\mu(\mathbf{X}, x_1, \dots x_{j-1})$ for some $j \in \{1, \dots m\}$ s.t.

- $\sigma \models \mu$ **...** μ **generalizes** σ
- $\mu \Rightarrow \gamma_j \wedge \delta_j$

Counterexample Generalization

Recall: Skolem functions guessed from approximations of

$\exists y_{i+1} \ldots \exists y_m \varphi(\mathbf{X}, x_1, \ldots x_{i-1}, y_i, y_{i+1}, \ldots y_m)$

- Let $\exists y_{i+1} \ldots \exists y_m \varphi(\mathbf{X}, \mathbf{Y}) \Rightarrow \Theta_i(\mathbf{X}, x_1, \ldots x_{i-1}, y_i)$
- Let $\delta_i = \neg\Theta_i|_{y_i=0}; \quad \gamma_i = \neg\Theta_i|_{y_i=1}$
- Initial guess $G_i(\mathbf{X}, x_1, \ldots x_{i-1}) \in \{\delta_i, \neg\gamma_i\}$ **... 1-sided error**
  - $G_i = \delta_i$ **cannot err** if it evaluates to 1
  - $G_i = \neg\gamma_i$ **cannot err** if it evaluates to 0

Generalized counterexample

Given $\sigma \models \varepsilon(\mathbf{X}, \mathbf{Y}, \mathbf{Y}')$ and δ_i, γ_i for $1 \leq i \leq m$

Find function $\mu(\mathbf{X}, x_1, \ldots x_{j-1})$ for some $j \in \{1, \ldots m\}$ s.t.

- $\sigma \models \mu$ **... μ generalizes σ**
- $\mu \Rightarrow \gamma_j \wedge \delta_j$
  - $\Rightarrow \forall y_j \ldots \forall y_m \neg\varphi(\mathbf{X}, x_1, \ldots x_{j-1}, y_j, y_{j+1}, \ldots y_m)$

Counterexample Generalization

Recall: Skolem functions guessed from approximations of

$\exists y_{i+1} \ldots \exists y_m \varphi(\mathbf{X}, x_1, \ldots x_{i-1}, y_i, y_{i+1}, \ldots y_m)$

- Let $\exists y_{i+1} \ldots \exists y_m \varphi(\mathbf{X}, \mathbf{Y}) \Rightarrow \Theta_i(\mathbf{X}, x_1, \ldots x_{i-1}, y_i)$
- Let $\delta_i = \neg\Theta_i|_{y_i=0}$; $\quad \gamma_i = \neg\Theta_i|_{y_i=1}$
- Initial guess $G_i(\mathbf{X}, x_1, \ldots x_{i-1}) \in \{\delta_i, \neg\gamma_i\}$ **... 1-sided error**
  - $G_i = \delta_i$ **cannot err** if it evaluates to 1
  - $G_i = \neg\gamma_i$ **cannot err** if it evaluates to 0

Generalized counterexample

Given $\sigma \models \varepsilon(\mathbf{X}, \mathbf{Y}, \mathbf{Y}')$ and δ_i, γ_i for $1 \leq i \leq m$

Find function $\mu(\mathbf{X}, x_1, \ldots x_{j-1})$ for some $j \in \{1, \ldots m\}$ s.t.

- $\sigma \models \mu$ **... μ generalizes σ**
- $\mu \Rightarrow \gamma_j \wedge \delta_j$
  - $\Rightarrow \forall y_j \ldots \forall y_m \neg\varphi(\mathbf{X}, x_1, \ldots x_{j-1}, y_j, y_{j+1}, \ldots y_m)$
  - If $\pi \models \mu$, no extension of π satisfies φ **... counterexample**

Counterexample Generalization

Recall: Skolem functions guessed from approximations of

$\exists y_{i+1} \ldots \exists y_m \, \varphi(\mathbf{X}, x_1, \ldots x_{i-1}, y_i, y_{i+1}, \ldots y_m)$

- Let $\exists y_{i+1} \ldots \exists y_m \, \varphi(\mathbf{X}, \mathbf{Y}) \Rightarrow \Theta_i(\mathbf{X}, x_1, \ldots x_{i-1}, y_i)$
- Let $\delta_i = \neg\Theta_i|_{y_i=0}$; $\quad \gamma_i = \neg\Theta_i|_{y_i=1}$
- Initial guess $G_i(\mathbf{X}, x_1, \ldots x_{i-1}) \in \{\delta_i, \neg\gamma_i\}$ **... 1-sided error**
  - $G_i = \delta_i$ **cannot err** if it evaluates to 1
  - $G_i = \neg\gamma_i$ **cannot err** if it evaluates to 0

Generalized counterexample

Given $\sigma \models \varepsilon(\mathbf{X}, \mathbf{Y}, \mathbf{Y}')$ and δ_i, γ_i for $1 \leq i \leq m$

Find function $\mu(\mathbf{X}, x_1, \ldots x_{j-1})$ for some $j \in \{1, \ldots m\}$ s.t.

- $\sigma \models \mu$ **... μ generalizes σ**
- $\mu \Rightarrow \gamma_j \wedge \delta_j$
  - $\Rightarrow \forall y_j \ldots \forall y_m \neg\varphi(\mathbf{X}, x_1, \ldots x_{j-1}, y_j, y_{j+1}, \ldots y_m)$
  - If $\pi \models \mu$, no extension of π satisfies φ **... counterexample**

Must ensure that $(\mathbf{X}, G_1, \ldots G_{j-1})$ never evaluates to π

Repairing "guess"-ed candidate Skolem functions

- Every model of $\mu(\mathbf{X}, x_1, \ldots x_{j-1})$ gives a problematic combination of $G_1, \ldots G_{j-1}$ values

Repairing “guess”-ed candidate Skolem functions

- Every model of $\mu(\mathbf{X}, x_1, \ldots x_{j-1})$ gives a problematic combination of $G_1, \ldots G_{j-1}$ values
- Flip G_{j-1} whenever μ holds

Repairing "guess"-ed candidate Skolem functions

- Every model of $\mu(\mathbf{X}, x_1, \ldots x_{j-1})$ gives a problematic combination of $G_1, \ldots G_{j-1}$ values
- Flip G_{j-1} whenever μ holds
  - Recall $G_{j-1} \in \{\neg\gamma_{j-1}, \delta_{j-1}\}$

Repairing "guess"-ed candidate Skolem functions

- Every model of $\mu(\mathbf{X}, x_1, \ldots x_{j-1})$ gives a problematic combination of $G_1, \ldots G_{j-1}$ values
- Flip G_{j-1} whenever μ holds
  - Recall $G_{j-1} \in \{\neg\gamma_{j-1}, \delta_{j-1}\}$
  - **Only** source of error: under-approximation of $\neg\exists y_j, \ldots \exists y_m \varphi(\mathbf{X}, x_1, \ldots x_{j-2}, y_{j-1}, y_j, \ldots y_m)$

Repairing "guess"-ed candidate Skolem functions

- Every model of $\mu(\mathbf{X}, x_1, \ldots x_{j-1})$ gives a problematic combination of $G_1, \ldots G_{j-1}$ values
- Flip G_{j-1} whenever μ holds
  - Recall $G_{j-1} \in \{\neg\gamma_{j-1}, \delta_{j-1}\}$
  - **Only** source of error: under-approximation of $\neg\exists y_j, \ldots \exists y_m\, \varphi(\mathbf{X}, x_1, \ldots x_{j-2}, y_{j-1}, y_j, \ldots y_m)$
  - Repair: **Expand under-approximation**
    - If G_{j-1} is $\neg\gamma_{j-1}$, $\quad \gamma_{j-1} \leftarrow \gamma_{j-1} \vee \mu|_{\sigma[y_{j-1}]}$
    - If G_{j-1} is δ_{j-1}, $\quad \delta_{j-1} \leftarrow \delta_{j-1} \vee \mu|_{\sigma[y_{j-1}]}$

Repairing "guess"-ed candidate Skolem functions

- Every model of $\mu(\mathbf{X}, x_1, \ldots x_{j-1})$ gives a problematic combination of $G_1, \ldots G_{j-1}$ values
- Flip G_{j-1} whenever μ holds
  - Recall $G_{j-1} \in \{\neg\gamma_{j-1}, \delta_{j-1}\}$
  - **Only** source of error: under-approximation of $\neg\exists y_j, \ldots \exists y_m\, \varphi(\mathbf{X}, x_1, \ldots x_{j-2}, y_{j-1}, y_j, \ldots y_m)$
  - Repair: **Expand under-approximation**
    - If G_{j-1} is $\neg\gamma_{j-1}$, $\quad \gamma_{j-1} \leftarrow \gamma_{j-1} \vee \mu|_{\sigma[y_{j-1}]}$
    - If G_{j-1} is δ_{j-1}, $\quad \delta_{j-1} \leftarrow \delta_{j-1} \vee \mu|_{\sigma[y_{j-1}]}$

Counter-example guided repair by expanding δ_i's and γ_i's.

Repairing “guess”-ed candidate Skolem functions

- Every model of $\mu(\mathbf{X}, x_1, \ldots x_{j-1})$ gives a problematic combination of $G_1, \ldots G_{j-1}$ values
- Flip G_{j-1} whenever μ holds
  - Recall $G_{j-1} \in \{\neg\gamma_{j-1}, \delta_{j-1}\}$
  - **Only** source of error: under-approximation of $\neg\exists y_j, \ldots \exists y_m\, \varphi(\mathbf{X}, x_1, \ldots x_{j-2}, y_{j-1}, y_j, \ldots y_m)$
  - Repair: **Expand under-approximation**
    - If G_{j-1} is $\neg\gamma_{j-1}$, $\quad \gamma_{j-1} \leftarrow \gamma_{j-1} \vee \mu|_{\sigma[y_{j-1}]}$
    - If G_{j-1} is δ_{j-1}, $\quad \delta_{j-1} \leftarrow \delta_{j-1} \vee \mu|_{\sigma[y_{j-1}]}$

Counter-example guided repair by expanding δ_i’s and γ_i’s.

Expansion-based repair

Repairing "guess"-ed candidate Skolem functions

- Every model of $\mu(\mathbf{X}, x_1, \ldots x_{j-1})$ gives a problematic combination of $G_1, \ldots G_{j-1}$ values
- Flip G_{j-1} whenever μ holds
  - Recall $G_{j-1} \in \{\neg\gamma_{j-1}, \delta_{j-1}\}$
  - **Only** source of error: under-approximation of $\neg\exists y_j, \ldots \exists y_m \, \varphi(\mathbf{X}, x_1, \ldots x_{j-2}, y_{j-1}, y_j, \ldots y_m)$
  - Repair: **Expand under-approximation**
    - If G_{j-1} is $\neg\gamma_{j-1}$, $\quad \gamma_{j-1} \leftarrow \gamma_{j-1} \vee \mu|_{\sigma[y_{j-1}]}$
    - If G_{j-1} is δ_{j-1}, $\quad \delta_{j-1} \leftarrow \delta_{j-1} \vee \mu|_{\sigma[y_{j-1}]}$

Counter-example guided repair by expanding δ_i's and γ_i's.

Expansion-based repair

Simple argument for termination – expansions can't go on forever

Deep Dive 2: Knowledge compilation for Boolean Functional Synthesis

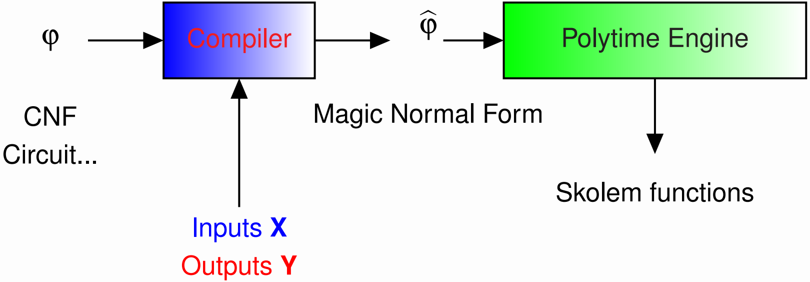
Knowledge Compilation for Synthesis

Our Definition

... a family of approaches for addressing the intractability of synthesis problems. A propositional model is compiled in an off-line phase in order to support some queries in polytime.

Our Definition

... a family of approaches for addressing the intractability of synthesis problems. A propositional model is compiled in an off-line phase in order to support some queries in polytime.



Recall from the previous Deep-Dive

For solving the Skolem function synthesis problem, it suffices to

1. Generate Skolem functions for only 1-output specs
2. For multiple output case, if we can compute $\exists y_i \ldots y_m \, \varphi(\mathbf{X}, \mathbf{Y})$, then it reduces to multiple instances of the single output problem!

Recall from the previous Deep-Dive

For solving the Skolem function synthesis problem, it suffices to

1. Generate Skolem functions for only 1-output specs
   - this is easy: $\varphi(\mathbf{X}, 1)$ and $\neg\varphi(\mathbf{X}, 0)$ are Skolem functions.
2. For multiple output case, if we can compute $\exists y_i \ldots y_m \varphi(\mathbf{X}, \mathbf{Y})$, then it reduces to multiple instances of the single output problem!

Recall from the previous Deep-Dive

For solving the Skolem function synthesis problem, it suffices to

1. Generate Skolem functions for only 1-output specs
   - this is easy: $\varphi(\mathbf{X}, 1)$ and $\neg\varphi(\mathbf{X}, 0)$ are Skolem functions.
2. For multiple output case, if we can compute $\exists y_i \ldots y_m \varphi(\mathbf{X}, \mathbf{Y})$, then it reduces to multiple instances of the single output problem!

Recall from the previous Deep-Dive

For solving the Skolem function synthesis problem, it suffices to

1. Generate Skolem functions for only 1-output specs
   - this is easy: $\varphi(\mathbf{X}, 1)$ and $\neg\varphi(\mathbf{X}, 0)$ are Skolem functions.
2. For multiple output case, if we can compute $\exists y_i \ldots y_m \varphi(\mathbf{X}, \mathbf{Y})$, then it reduces to multiple instances of the single output problem!

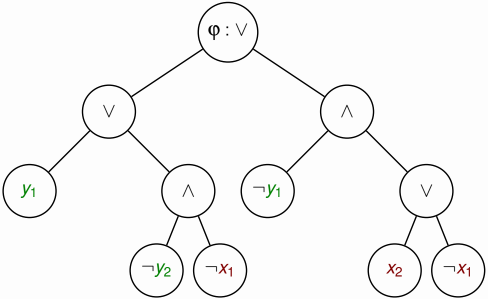
Does there exist a form of the specification where this HARD question is EASY?

Towards a normal form for efficient synthesis

- Represent $\varphi(x_1, .., x_n, y_1, .., y_m)$ NNF DAG
  - Boolean circuit, $\wedge$ and $\vee$ at internal nodes, $\neg$ only at leaves

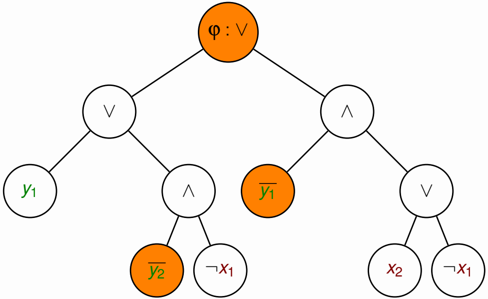
Towards a normal form for efficient synthesis

- Represent $\varphi(x_1, .., x_n, y_1, .., y_m)$ NNF DAG
  - Boolean circuit, $\wedge$ and $\vee$ at internal nodes, $\neg$ only at leaves



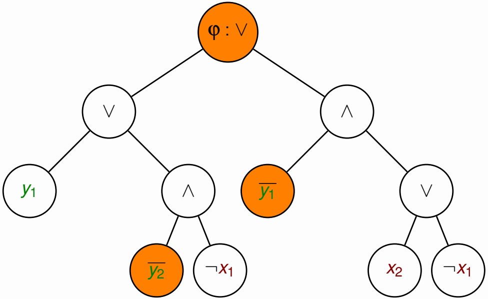
Towards a normal form for efficient synthesis

- Represent $\varphi(x_1, .., x_n, y_1, .., y_m)$ NNF DAG
  - Boolean circuit, $\wedge$ and $\vee$ at internal nodes, $\neg$ only at leaves



Towards a normal form for efficient synthesis

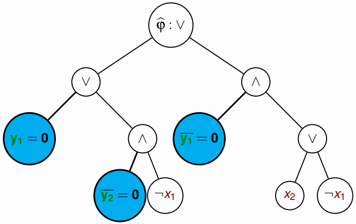
- Represent $\varphi(x_1, .., x_n, y_1, .., y_m)$ NNF DAG
  - Boolean circuit, $\wedge$ and $\vee$ at internal nodes, $\neg$ only at leaves



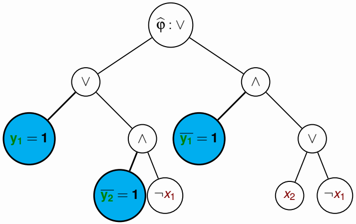
Positive form of specification: $\widehat{\varphi}(\{x_1, \ldots x_n\}, \{y_1, \ldots, y_m, \overline{y_1}, \ldots \overline{y_m}\})$

- Monotone w.r.t all y_i and $\overline{y_i}$

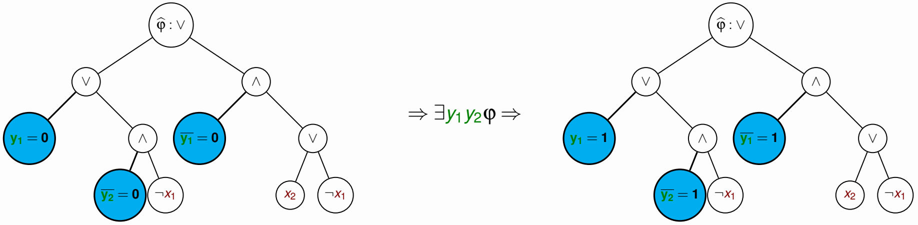
Simple properties of the positive form $\widehat{\varphi}$



$\Rightarrow \exists y_1 y_2 \varphi \Rightarrow$



Simple properties of the positive form $\widehat{\varphi}$



- $\widehat{\varphi}(x_1 \ldots x_n, \overbrace{0..0}^{i}, y_{i+1} \ldots y_m, \overbrace{0..0}^{i}, \neg y_{i+1} \ldots \neg y_m) \Rightarrow \exists y_1 \ldots y_i \; \varphi(\ldots)$
- $\widehat{\varphi}(x_1 \ldots x_n, \overbrace{1..1}^{i}, y_{i+1} \ldots y_m, \overbrace{1..1}^{i}, \neg y_{i+1} \ldots \neg y_m) \Leftarrow \exists y_1 \ldots y_i \; \varphi(\ldots)$

The positive form and existential quantification

Let us take the first output: $\exists y_1 \varphi(\mathbf{X}, \mathbf{Y}) \;\Rightarrow\; \widehat{\varphi}\,|_{y_1=1,\overline{y_1}=1}$

The positive form and existential quantification

Let us take the 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?

The positive form and existential quantification

Let us take the 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.

The positive form and existential quantification

Let us take the 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 $\exists y_1 \varphi(\mathbf{X}, \mathbf{Y}) \not\Leftarrow \widehat{\varphi}\,|_{y_1=1,\overline{y_1}=1}$?

The positive form and existential quantification

Let us take the 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
  - $\widehat{\varphi}_1\,|_{y_1=1,\overline{y_1}=1} \;=\; 1$
  - $\exists y_1 \varphi(\mathbf{X}, \mathbf{Y}) \Leftrightarrow \varphi\,|_{y_1=1} \;\vee\; \varphi\,|_{y_1=0} \;=\; 0$

The positive form and existential quantification

Let us take the 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
  - $\widehat{\varphi}_1\,|_{y_1=1,\overline{y_1}=1} = 1$
  - $\exists y_1 \varphi(\mathbf{X},\mathbf{Y}) \Leftrightarrow \varphi\,|_{y_1=1} \vee \varphi\,|_{y_1=0} = 0$
    - $\varphi\,|_{y_1=1} \Leftrightarrow \widehat{\varphi}\,|_{y_1=1,\overline{y_1}=0} = 0$
    - $\varphi\,|_{y_1=0} \Leftrightarrow \widehat{\varphi}\,|_{y_1=0,\overline{y_1}=1} = 0$

The positive form and existential quantification

Let us take the 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 $\overline{y_1}$) $\widehat{\varphi}\,|_{y_1=0,\overline{y_1}=0} = 0$

The positive form and existential quantification

Let us take the first output: $\exists y_1 \varphi(\mathbf{X}, \mathbf{Y}) \Rightarrow \widehat{\varphi}\mid_{y_1=1,\overline{y_1}=1}$ When does the reverse implication hold?

- Let's ask the opposite. When do we have $\exists y_1 \varphi(\mathbf{X}, \mathbf{Y}) \not\Leftarrow \widehat{\varphi}\mid_{y_1=1,\overline{y_1}=1}$?
- Exactly when
  - $\widehat{\varphi}_1 \mid_{y_1=1,\overline{y_1}=1} = 1$
  - $\exists y_1 \varphi(\mathbf{X}, \mathbf{Y}) \Leftrightarrow \varphi\mid_{y_1=1} \vee \varphi\mid_{y_1=0} = 0$
    - $\varphi\mid_{y_1=1} \Leftrightarrow \widehat{\varphi}\mid_{y_1=1,\overline{y_1}=0} = 0$
    - $\varphi\mid_{y_1=0} \Leftrightarrow \widehat{\varphi}\mid_{y_1=0,\overline{y_1}=1} = 0$
    - (By monotonicity of $\widehat{\varphi}$ w.r.t y_1 and $\overline{y_1}$) $\widehat{\varphi}\mid_{y_1=0,\overline{y_1}=0} = 0$
- In other words, when $\widehat{\varphi}$ "behaves like" $y_1 \wedge \overline{y_1}$.

The positive form and existential quantification

Let us take the 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 $\overline{y_1}$) $\widehat{\varphi}\,|_{y_1=0,\overline{y_1}=0} \;=\; 0$
- In other words, when $\widehat{\varphi}$ "behaves like" $y_1 \wedge \overline{y_1}$.

So, what should we avoid?

- There are some values for the other variables s.t., $\widehat{\varphi} \Leftrightarrow y_1 \wedge \overline{y_1}$.

The positive form and existential quantification

Let us take the first output: $\exists y_1 \varphi(\mathbf{X}, \mathbf{Y}) \Rightarrow \widehat{\varphi} \mid_{y_1=1, \overline{y_1}=1}$ When does the reverse implication hold?

- Let's ask the opposite. When do we have $\exists y_1 \varphi(\mathbf{X}, \mathbf{Y}) \not\Leftarrow \widehat{\varphi} \mid_{y_1=1, \overline{y_1}=1}$?
- Exactly when
  - $\widehat{\varphi}_1 \mid_{y_1=1, \overline{y_1}=1} = 1$
  - $\exists y_1 \varphi(\mathbf{X}, \mathbf{Y}) \Leftrightarrow \varphi \mid_{y_1=1} \vee \varphi \mid_{y_1=0} = 0$
    - $\varphi \mid_{y_1=1} \Leftrightarrow \widehat{\varphi} \mid_{y_1=1, \overline{y_1}=0} = 0$
    - $\varphi \mid_{y_1=0} \Leftrightarrow \widehat{\varphi} \mid_{y_1=0, \overline{y_1}=1} = 0$
    - (By monotonicity of $\widehat{\varphi}$ w.r.t y_1 and $\overline{y_1}$) $\widehat{\varphi} \mid_{y_1=0, \overline{y_1}=0} = 0$
- In other words, when $\widehat{\varphi}$ "behaves like" $y_1 \wedge \overline{y_1}$.

So, what should we avoid?

- There are some values for the other variables s.t., $\widehat{\varphi} \Leftrightarrow y_1 \wedge \overline{y_1}$.
- If we can avoid it, we get $\exists y_1 \varphi(\mathbf{X}, \mathbf{Y}) \Leftrightarrow \widehat{\varphi} \mid_{y_1=1, \overline{y_1}=1}$

The positive form and existential quantification

Let us take the 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 $\overline{y_1}$) $\widehat{\varphi}\,|_{y_1=0,\overline{y_1}=0} = 0$
- In other words, when $\widehat{\varphi}$ "behaves like" $y_1 \wedge \overline{y_1}$.

So, what should we avoid?

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

The core simple idea: Don't be an AND gate!

We can now generalize this to more outputs

The core simple idea: Don't be an AND gate!

We can now generalize this to more outputs

If we can avoid

- $\widehat{\varphi} \Leftrightarrow y_1 \wedge \overline{y_1}$ AND $\widehat{\varphi}\,|_{y_1=1,\overline{y_1}=1} \Leftrightarrow y_2 \wedge \overline{y_2}$.

We can now generalize this to more outputs

If we can avoid

- $\widehat{\varphi} \Leftrightarrow y_1 \wedge \overline{y_1}$ AND $\widehat{\varphi}\,|_{y_1=1,\overline{y_1}=1} \Leftrightarrow y_2 \wedge \overline{y_2}$.

Then we get

- $\exists y_1, y_2 \varphi(\mathbf{X}, \mathbf{Y}) \;\Leftrightarrow\; \widehat{\varphi}\,|_{y_1=1,\overline{y_1}=1,y_2=1,\overline{y_2}=1}$

and so on...

The core simple idea: Don't be an AND gate!

We can now generalize this to more outputs

If we can avoid

- $\widehat{\varphi} \Leftrightarrow y_1 \wedge \overline{y_1}$ AND $\widehat{\varphi}\,|_{y_1=1,\overline{y_1}=1} \Leftrightarrow y_2 \wedge \overline{y_2}$.

Then we get

- $\exists y_1, y_2 \varphi(\mathbf{X}, \mathbf{Y}) \;\Leftrightarrow\; \widehat{\varphi}\,|_{y_1=1,\overline{y_1}=1,y_2=1,\overline{y_2}=1}$

and so on...

The question

- We want to ensure the positive form does not "behave" as $y_i \wedge \overline{y_i}$ for any i.

The core simple idea: Don't be an AND gate!

We can now generalize this to more outputs

If we can avoid

- $\widehat{\varphi} \Leftrightarrow y_1 \wedge \overline{y_1}$ AND $\widehat{\varphi}\,|_{y_1=1,\overline{y_1}=1} \Leftrightarrow y_2 \wedge \overline{y_2}$.

Then we get

- $\exists y_1, y_2 \varphi(\mathbf{X}, \mathbf{Y}) \Leftrightarrow \widehat{\varphi}\,|_{y_1=1,\overline{y_1}=1,y_2=1,\overline{y_2}=1}$

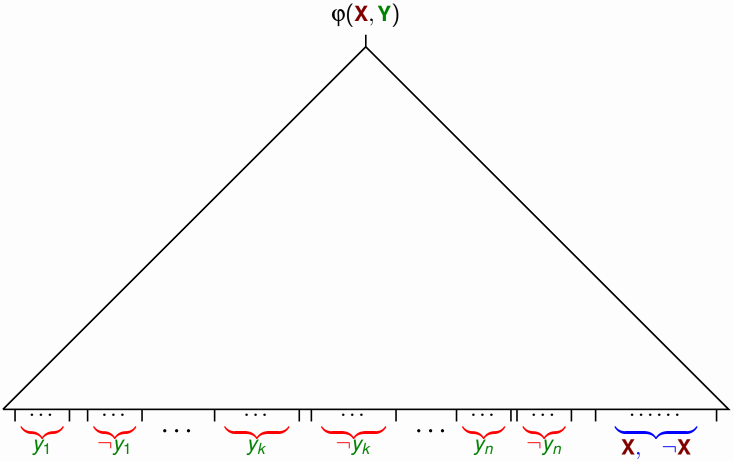
and so on...

The question

- We want to ensure the positive form does not "behave" as $y_i \wedge \overline{y_i}$ for any i.
- What representation of the specification φ ensures this?

A simple yet special Normal Form

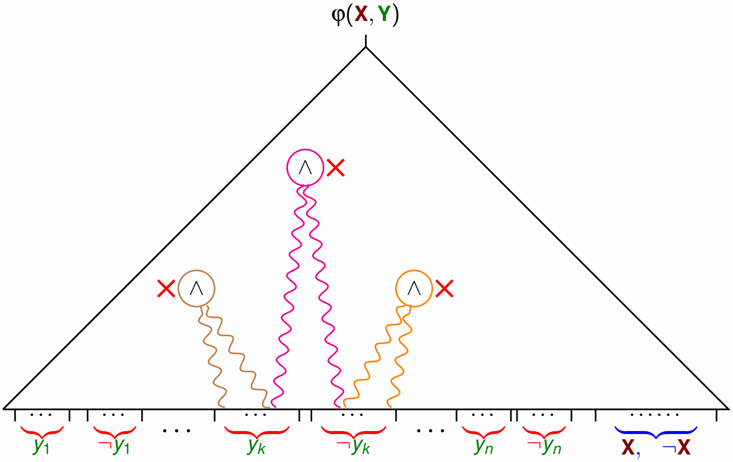
Decomposable Negation Normal Form (DNNF): Forbidden **structure**



Adnan Darwiche: Decomposable negation normal form. J. ACM 48(4): 608-647 (2001)

A simple yet special Normal Form

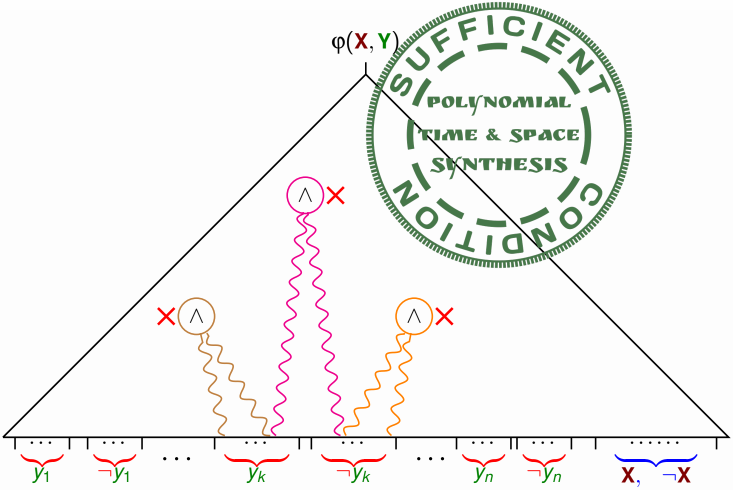
Decomposable Negation Normal Form (DNNF): Forbidden **structure**



Adnan Darwiche: Decomposable negation normal form. J. ACM 48(4): 608-647 (2001)

A simple yet special Normal Form

Decomposable Negation Normal Form (DNNF): Forbidden **structure**

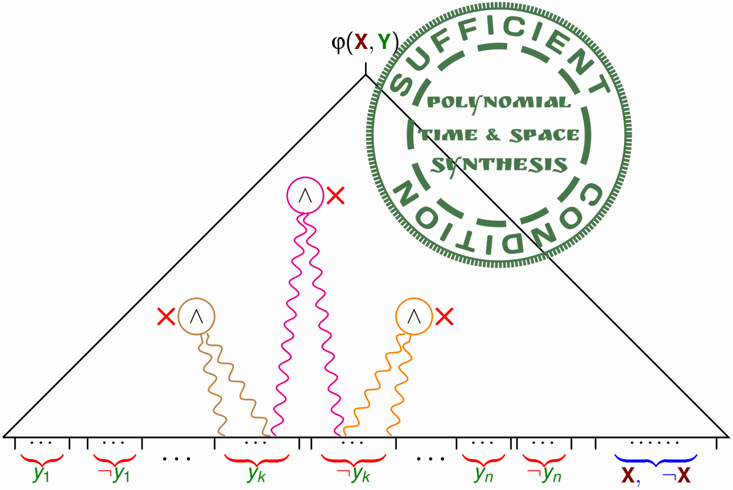


Adnan Darwiche: Decomposable negation normal form. J. ACM 48(4): 608-647 (2001)

A simple yet special Normal Form

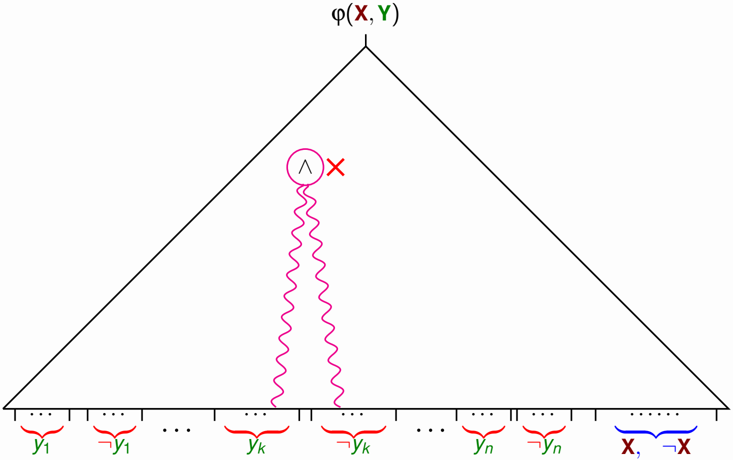
Decomposable Negation Normal Form (DNNF): Forbidden **structure**

DNNF has many other nice properties. Well-studied in the KR community!



Surely, we can do better!

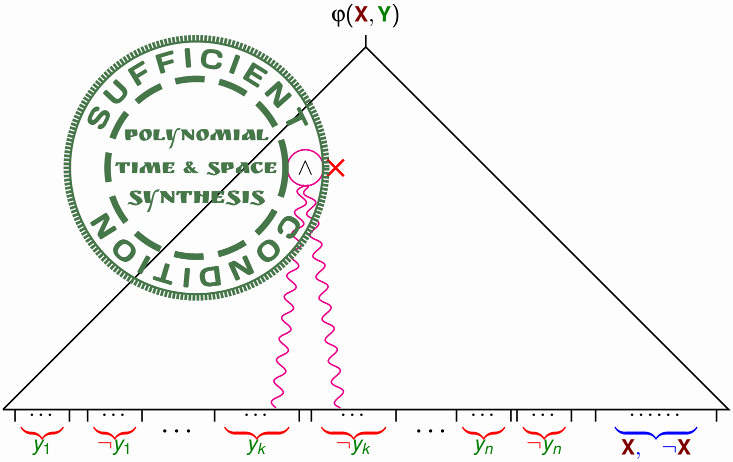
Weak DNNF (wDNNF): Forbidden **structure**



S. Akshay, Supratik Chakraborty, Shubham Goel, Sumith Kulal, Shetal Shah: What's Hard About Boolean Functional Synthesis? CAV (1) 2018: 251-269

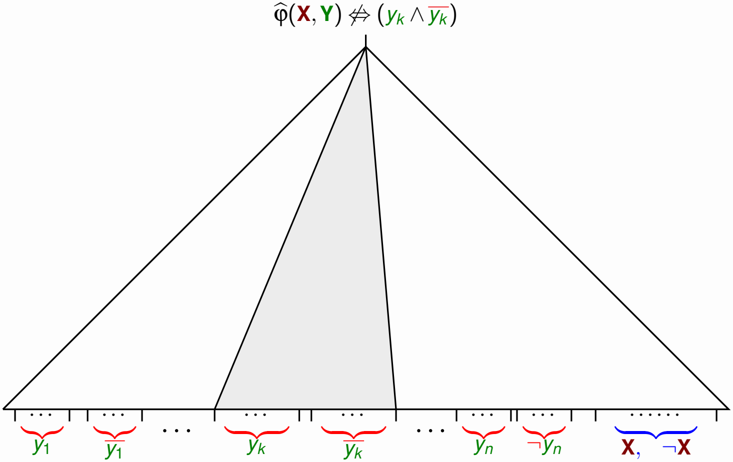
Surely, we can do better!

Weak DNNF (wDNNF): Forbidden **structure**



Exploit the property of the reduct!

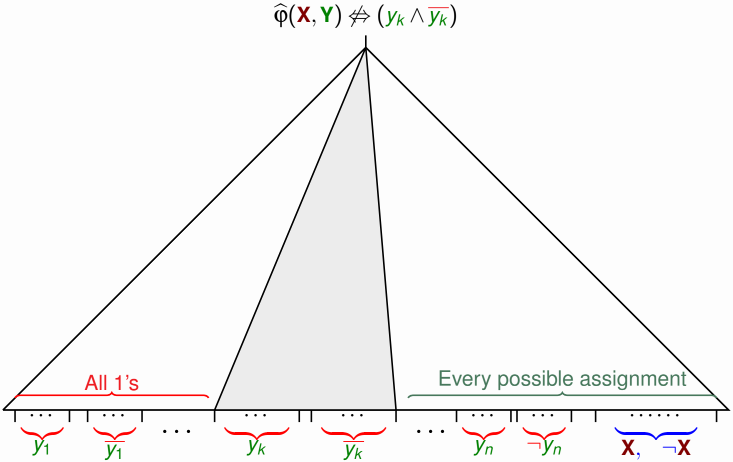
Synthesis Negation Normal Form (SynNNF): Forbidden **semantics**



S. Akshay, J. Arora, S. Chakraborty, S. Krishna, D. Raghunathan, S. Shah: Knowledge Compilation for Boolean Functional Synthesis. FMCAD 2019: 161-169

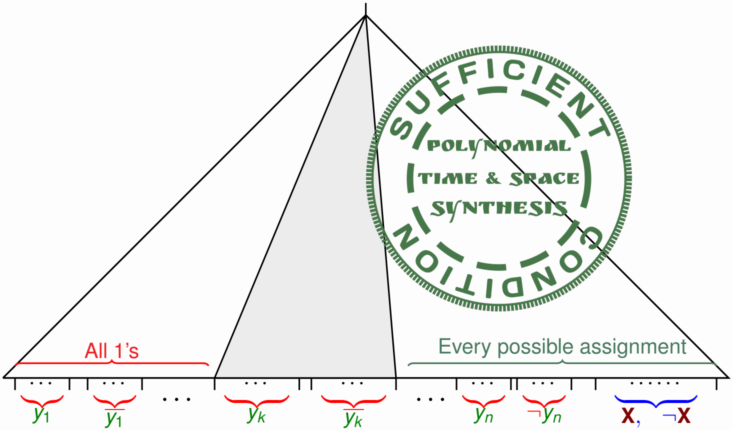
Exploit the property of the reduct!

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



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

$\widehat{\varphi}(\mathbf{X}, \mathbf{Y}) \not\Rightarrow (y_k \wedge \overline{y_k})$



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

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

SynNNF: A negation normal form for efficient synthesis

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

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

SynNNF: A negation normal form for efficient synthesis

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

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

SynNNF: A negation normal form for efficient synthesis

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

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

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

- $\exists y_1, \ldots y_{i-1}\ \varphi(\mathbf{X}, y_1, \ldots y_{i-1}, 1, y_{i+1}, \ldots y_m)$
- 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 $\varphi_1 \wedge \varphi_2 = \bot$.

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.

What more can we do?

What more can we do?

Can we get necessary & sufficient condition?

What more can we do?

Can we get necessary & sufficient condition?

Characterizing poly-time and poly-size BFnS

Does there exist a ”semantically universal” class $\mathcal{C}^\star$ of ckts s.t.:

P1 : BFnS is poly-time for $\mathcal{C}^\star$

What more can we do?

Can we get necessary & sufficient condition?

Characterizing poly-time and poly-size BFnS

Does there exist a ”semantically universal” class $\mathcal{C}^\star$ of ckts s.t.:

- P1 : BFnS is poly-time for $\mathcal{C}^\star$
- P2 : For every class $\mathcal{C}$ of ckts:
  1. BFnS is poly-time for $\mathcal{C}$ iff $\mathcal{C}$ compiles to $\mathcal{C}^\star$ in poly-time.

What more can we do?

Can we get necessary & sufficient condition?

Characterizing poly-time and poly-size BFnS

Does there exist a "semantically universal" class $\mathcal{C}^\star$ of ckts s.t.:

- P1 : BFnS is poly-time for $\mathcal{C}^\star$
- P2 : For every class $\mathcal{C}$ of ckts:
  1. BFnS is poly-time for $\mathcal{C}$ iff $\mathcal{C}$ compiles to $\mathcal{C}^\star$ in poly-time.
  2. BFnS is poly-size for $\mathcal{C}$ iff $\mathcal{C}$ compiles to poly-size ckts in $\mathcal{C}^\star$

What more can we do?

Can we get necessary & sufficient condition?

Characterizing poly-time and poly-size BFnS

Does there exist a "semantically universal" class $\mathcal{C}^\star$ of ckts s.t.:

- P1 : BFnS is poly-time for $\mathcal{C}^\star$
- P2 : For every class $\mathcal{C}$ of ckts:
  1. BFnS is poly-time for $\mathcal{C}$ iff $\mathcal{C}$ compiles to $\mathcal{C}^\star$ in poly-time.
  2. BFnS is poly-size for $\mathcal{C}$ iff $\mathcal{C}$ compiles to poly-size ckts in $\mathcal{C}^\star$

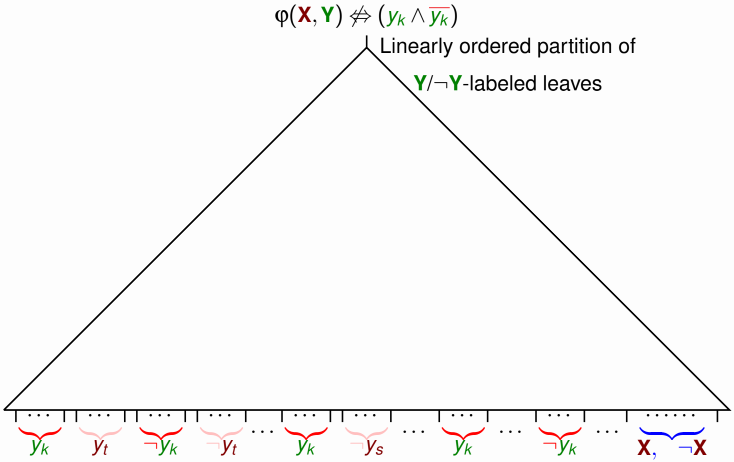
Surprise!

Yes, there exists such a class! Subset-And-Unrealizable Normal Form (SAUNF)

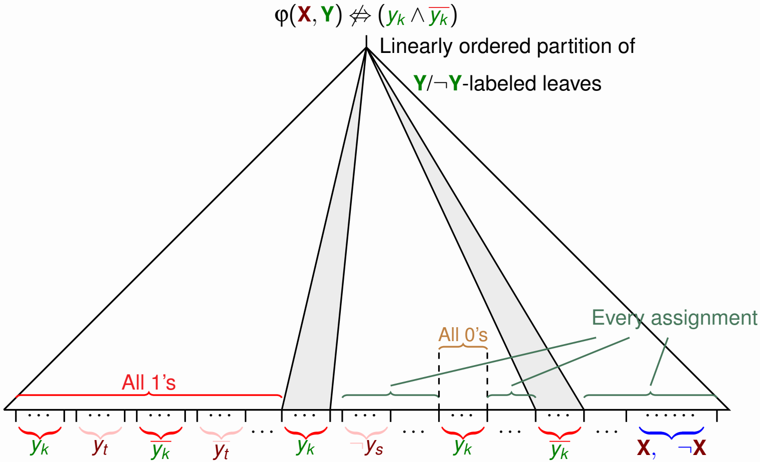
P. Shah, A. Bansal, S. Akshay, S. Chakraborty: A Normal Form Characterization for Efficient Boolean Skolem Function Synthesis. LICS 2021: 1-13

SAUNF: A Very Special Normal Form

Generalizing forbidden **semantics** of SynNNF

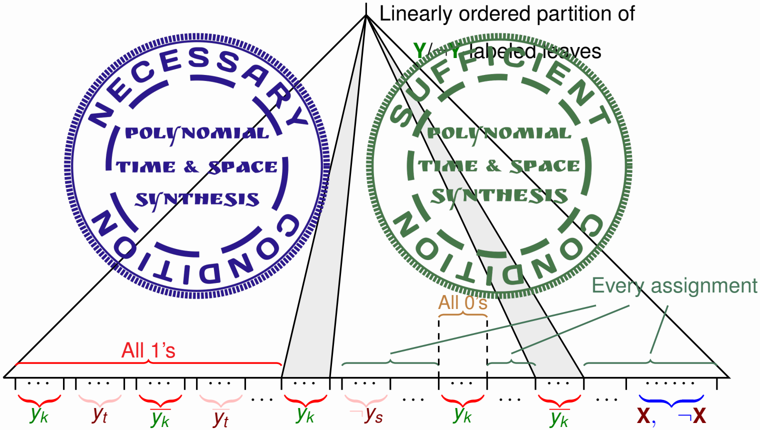


Generalizing forbidden **semantics** of SynNNF



Generalizing forbidden **semantics** of SynNNF

$\varphi(\mathbf{X}, \mathbf{Y}) \not\Leftrightarrow (y_k \wedge \overline{y_k})$



More properties and structure

Checking if a given specification is in SynNNF/SAUNF

- Is Co-NP complete, given linearly ordered variables/partition of **Y**-labeled leaves

More properties and structure

Checking if a given specification is in SynNNF/SAUNF

- Is Co-NP complete, given linearly ordered variables/partition of **Y**-labeled leaves
- Is Co-NP hard and in Σ_2^P, otherwise.

Compiling from CNF to SynNNF/SAUNF

- Algorithms and Prototype implementations exist. (e.g., C2Syn)
- Worst-case exponential-time and space
  - Unavoidable due to hardness results

More properties and structure

Checking if a given specification is in SynNNF/SAUNF

- Is Co-NP complete, given linearly ordered variables/partition of **Y**-labeled leaves
- Is Co-NP hard and in Σ_2^P, otherwise.

Compiling from CNF to SynNNF/SAUNF

- Algorithms and Prototype implementations exist. (e.g., C2Syn)
- Worst-case exponential-time and space
  - Unavoidable due to hardness results

Algorithms for compositions and operations

Given $\varphi_1(\mathbf{X}, \mathbf{Y})$ and $\varphi_2(\mathbf{X}, \mathbf{Y})$ in SynNNF/SAUNF

- Computing $\varphi_1 \vee \varphi_2$ in SynNNF/SAUNF takes constant time.

More properties and structure

Checking if a given specification is in SynNNF/SAUNF

- Is Co-NP complete, given linearly ordered variables/partition of **Y**-labeled leaves
- Is Co-NP hard and in Σ_2^P, otherwise.

Compiling from CNF to SynNNF/SAUNF

- Algorithms and Prototype implementations exist. (e.g., C2Syn)
- Worst-case exponential-time and space
  - Unavoidable due to hardness results

Algorithms for compositions and operations

Given $\varphi_1(\mathbf{X}, \mathbf{Y})$ and $\varphi_2(\mathbf{X}, \mathbf{Y})$ in SynNNF/SAUNF

- Computing $\varphi_1 \vee \varphi_2$ in SynNNF/SAUNF takes constant time.
- Computing $\varphi_1 \wedge \varphi_2$ can take super-polynomial time.

More properties and structure

Checking if a given specification is in SynNNF/SAUNF

- Is Co-NP complete, given linearly ordered variables/partition of **Y**-labeled leaves
- Is Co-NP hard and in Σ_2^P, otherwise.

Compiling from CNF to SynNNF/SAUNF

- Algorithms and Prototype implementations exist. (e.g., C2Syn)
- Worst-case exponential-time and space
  - Unavoidable due to hardness results

Algorithms for compositions and operations

Given $\varphi_1(\mathbf{X}, \mathbf{Y})$ and $\varphi_2(\mathbf{X}, \mathbf{Y})$ in SynNNF/SAUNF

- Computing $\varphi_1 \vee \varphi_2$ in SynNNF/SAUNF takes constant time.
- Computing $\varphi_1 \wedge \varphi_2$ can take super-polynomial time.
- Existential quantification is easy.

End of this deep dive

Some takeaways

- Nice normal forms exist for Boolean Functional Synthesis!

End of this deep dive

Some takeaways

- Nice normal forms exist for Boolean Functional Synthesis!
- Knowledge representations and compilation is key.

End of this deep dive

Some takeaways

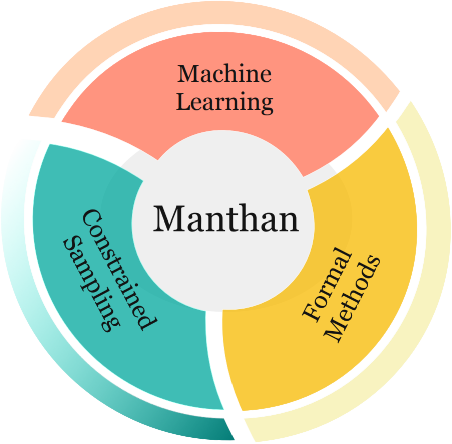
- Nice normal forms exist for Boolean Functional Synthesis!
- Knowledge representations and compilation is key.
- Explains performance of existing tools on some benchmarks.

End of this deep dive

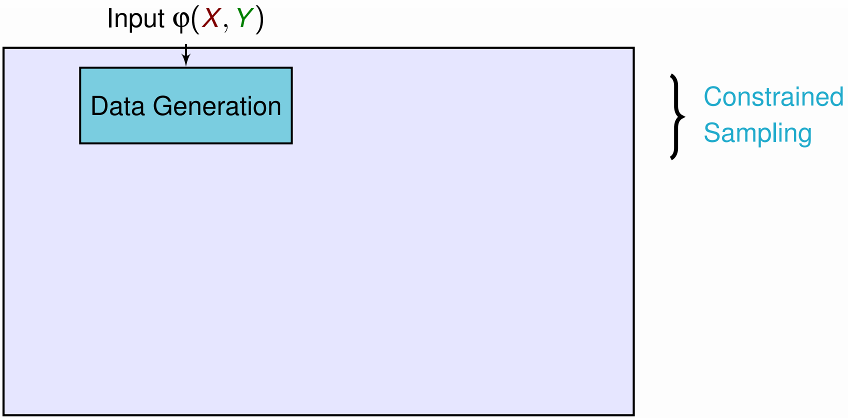
Some takeaways

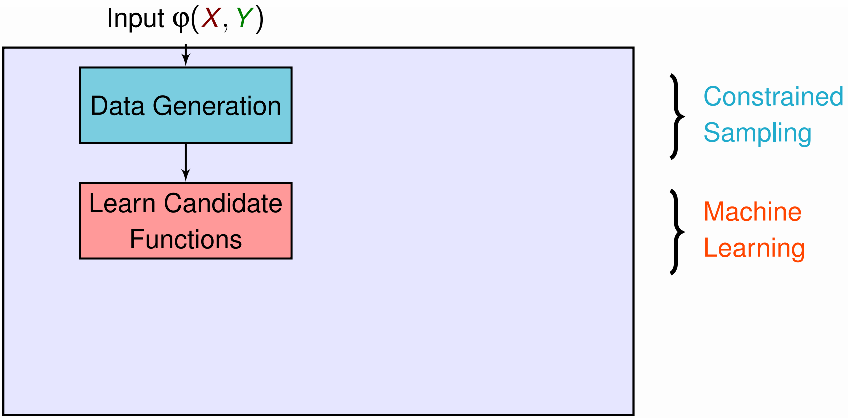
- Nice normal forms exist for Boolean Functional Synthesis!
- Knowledge representations and compilation is key.
- Explains performance of existing tools on some benchmarks.
- More in concluding remarks...

Another Flavour of Guess-Check-Repair

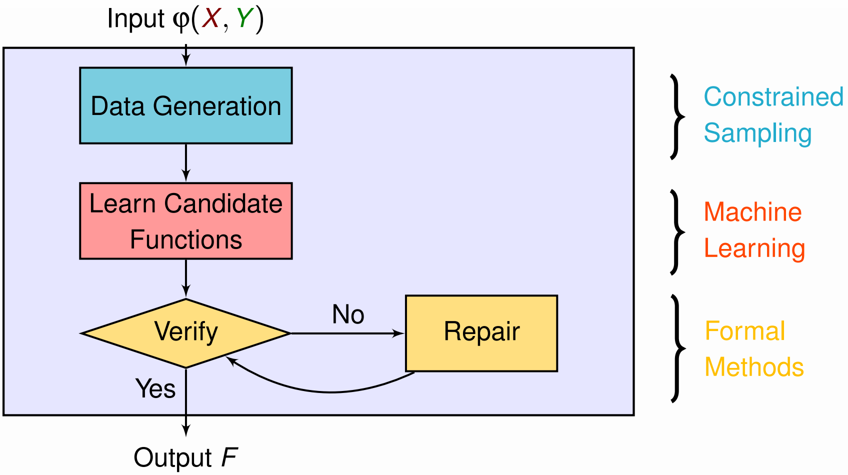


A data-driven approach for Skolem function synthesis


Input φ(X, Y)
Data Generation
Constrained
Sampling


Input φ(X, Y)
Data Generation
Learn Candidate Functions
Constrained Sampling
Machine Learning

Manthan

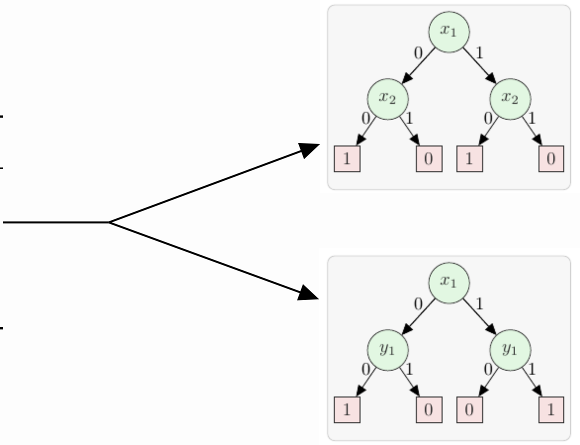

Input φ(X, Y)
Data Generation
Learn Candidate Functions
Verify
No
Repair
Yes
Output F
Constrained Sampling
Machine Learning
Formal Methods

$\varphi(x_1, x_2, y_1, y_2)$ ⟶

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

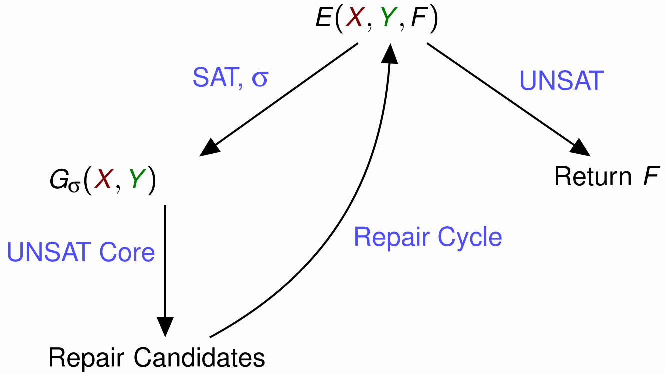
Learn Candidate Functions

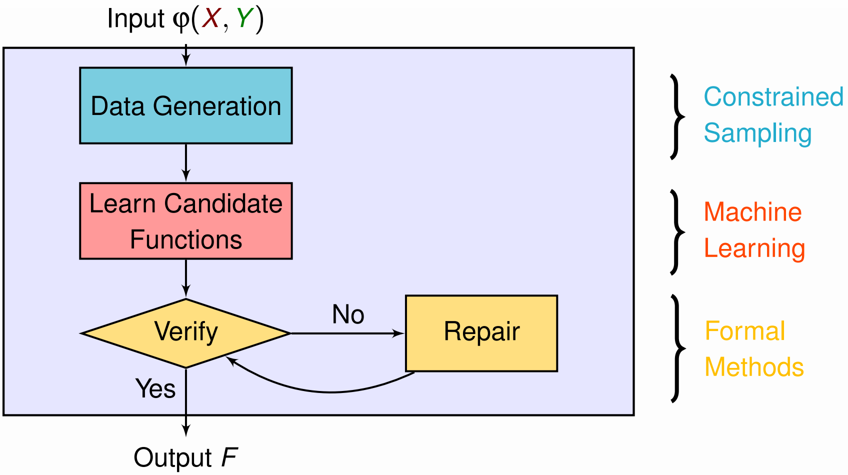
x_1	x_2	y_1	y_2
0	0	1	0
0	1	0	1
1	0	1	1
1	1	0	0



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


E(X, Y, F)
SAT, σ
UNSAT
G_σ(X, Y)
Return F
UNSAT Core
Repair Cycle
Repair Candidates


Input φ(X, Y)
Data Generation
Learn Candidate Functions
Verify
No
Repair
Yes
Output F
Constrained Sampling
Machine Learning
Formal Methods

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

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

Data Generation

$$\varphi(x_1, x_2, y_1, y_2) : (x_1 \lor x_2 \lor y_1) \land (\neg x_1 \lor \neg x_2 \lor \neg y_2)$$

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:
  - $f_1(x_1, x_2) = \neg(x_1 \lor x_2)$
  - $f_2(x_1, x_2) = \neg(x_1 \land x_2)$

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:
  - $f_1(x_1, x_2) = \neg(x_1 \vee x_2)$ $\quad f_1(x_1, x_2) = \neg x_1$ $\quad f_1(x_1, x_2) = \neg x_2$ $\quad f_1(x_1, x_2) = 1$
  - $f_2(x_1, x_2) = \neg(x_1 \wedge x_2)$ $\quad f_2(x_1, x_2) = \neg x_1$ $\quad f_2(x_1, x_2) = \neg x_2$ $\quad f_2(x_1, x_2) = 0$

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.

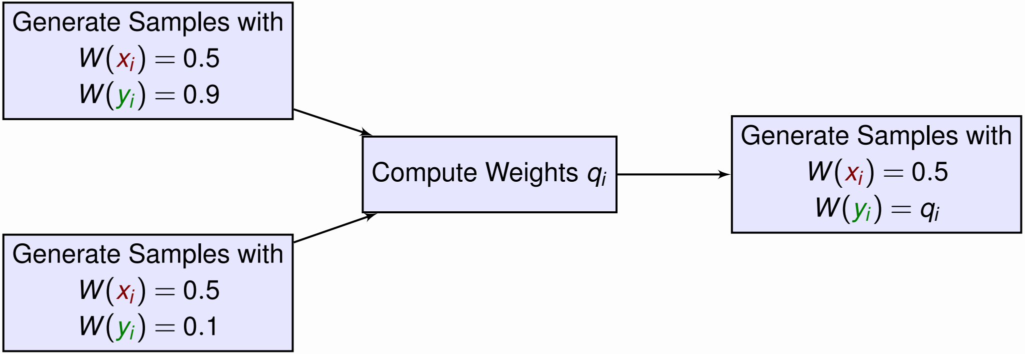
$$W(\sigma) = \prod_{\sigma(z_i)=1} W(z_i) \prod_{\sigma(z_i)=0} (1 - W(z_i))$$

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

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

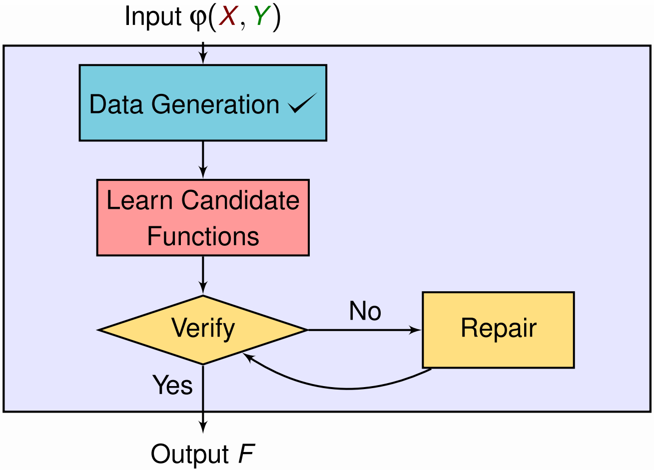
- Uniform sampling is a special case where all variables are assigned weight of 0.5.

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)


Input φ(X, Y)
Data Generation ✓
Learn Candidate Functions
Verify
No
Repair
Yes
Output F

Learn Candidate Function: Decision Tree Classifier

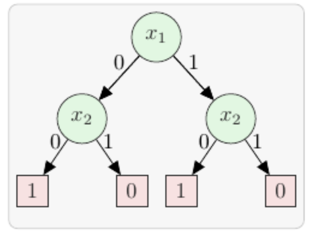
$$\varphi(x_1, x_2, y_1, y_2) : (x_1 \vee x_2 \vee y_1) \wedge (\neg x_1 \vee \neg x_2 \vee \neg y_2)$$

- 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

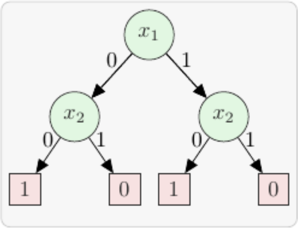
x_1	x_2	y_1	y_2
0	0	1	0
0	1	0	1
1	0	1	1
1	1	0	0



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

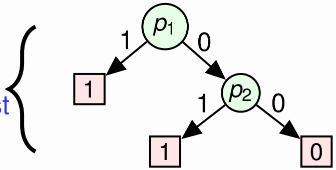
Learning Candidate Functions

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



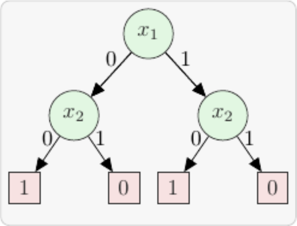
$p_1 := (\neg x_1 \wedge \neg x_2)$,
$p_2 := (x_1 \wedge \neg x_2)$
f_1 = if p_1 then 1
elif p_2 then 1
else 0

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

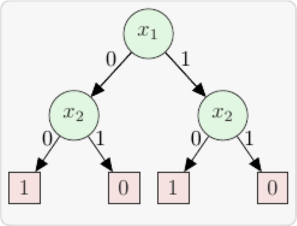


$p_1 := (\neg x_1 \wedge \neg x_2)$,
$p_2 := (x_1 \wedge \neg x_2)$
f_1 = if p_1 then 1
elif p_2 then 1
else 0

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

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

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



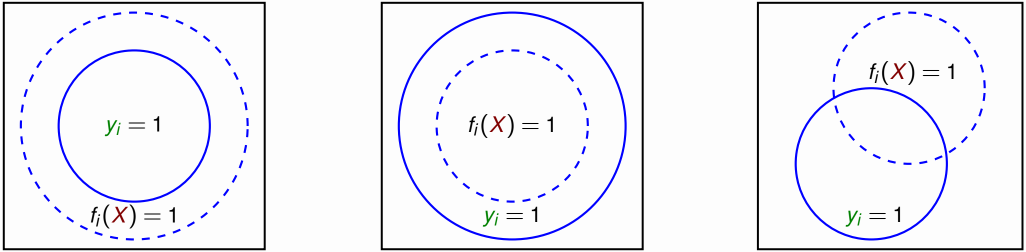
$p_1 := (\neg x_1 \wedge \neg x_2)$,
$p_2 := (x_1 \wedge \neg x_2)$
f_1 = if p_1 then 1
elif p_2 then 1
else 0

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

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

Learn with Errors: Approximations not Abstractions

Abstraction vs Approximation



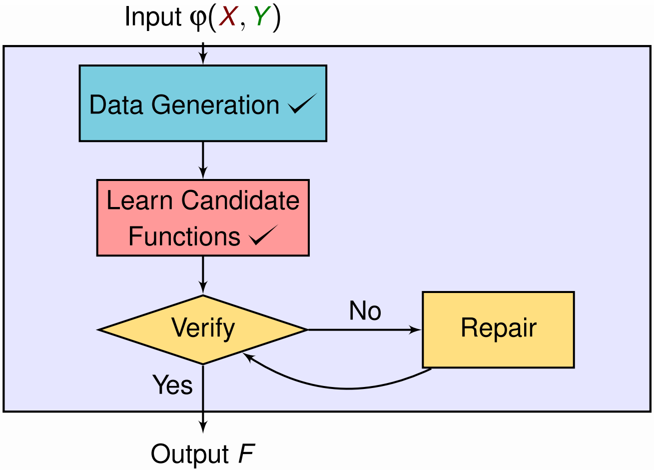
$y_i \rightarrow f_i(X)$ $\qquad$ $f_i(X) \rightarrow y_i$

Abstraction

Approximation

$y_i = 1, f_i(X) = 0$

$y_i = 0, f_i(X) = 1$


Input φ(X, Y)
Data Generation ✓
Learn Candidate Functions ✓
Verify
No
Repair
Yes
Output F

Verification of Candidate Functions

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

(JSCTA'15)

- If $E(X, Y, Y')$ is UNSAT: $\exists Y \varphi(X, Y) \equiv \varphi(X, F(X))$
  - Return F
- If $E(X, Y, Y')$ is SAT: $\exists Y \varphi(X, Y) \not\equiv \varphi(X, F(X))$
  - Let $\sigma \models E(X, Y, Y')$ be a counterexample to fix.

Repair 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.

- Let $\sigma := \{x_1 \mapsto 1, x_2 \mapsto 1, y_1 \mapsto 1, y_2 \mapsto 1, y_1' \mapsto 0, y_2' \mapsto 0\}$.
- Potential repair candidates: All y_i where $\sigma[y_i] \neq \sigma[y_i']$.

Repair 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.

- Let $\sigma := \{x_1 \mapsto 1, x_2 \mapsto 1, y_1 \mapsto 1, y_2 \mapsto 1, y'_1 \mapsto 0, y'_2 \mapsto 0\}$.
- Potential repair candidates: All y_i where $\sigma[y_i] \neq \sigma[y'_i]$.
- $\varphi(X, Y)$ is Boolean Relation.
  - So it can be $\hat{\sigma} = \{x_1 \mapsto 1, x_2 \mapsto 1, y_1 \mapsto 0, y_2 \mapsto 1, y'_1 \mapsto 0, y'_2 \mapsto 0\}$
  - We would not repair f_1.

Repair Candidate Identification

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

$$\sigma \models E(X,Y,Y') \text{ be a counterexample to fix.}$$

- Let $\sigma := \{x_1 \mapsto 1, x_2 \mapsto 1, y_1 \mapsto 1, y_2 \mapsto 1, y'_1 \mapsto 0, y'_2 \mapsto 0\}$.
- Potential repair candidates: All y_i where $\sigma[y_i] \neq \sigma[y'_i]$.
- $\varphi(X,Y)$ is Boolean Relation.
  - So it can be $\hat{\sigma} = \{x_1 \mapsto 1, x_2 \mapsto 1, y_1 \mapsto 0, y_2 \mapsto 1, y'_1 \mapsto 0, y'_2 \mapsto 0\}$
  - We would not repair f_1.
- MaxSAT-based Identification of *nice counterexamples*:
  - Hard Clauses $\varphi(X,Y) \land (X \leftrightarrow \sigma[X])$.
  - Soft Clauses $(Y \leftrightarrow \sigma[Y'])$.
- Candidates to repair: Y variables in the violated soft clauses

Repairing Approximations

- $\sigma = \{x_1 \mapsto 1, x_2 \mapsto 1, y_1 \mapsto 0, y_2 \mapsto 1, y_1' \mapsto 0, y_2' \mapsto 0\}$, and we want to repair f_2.

- Potential Repair: If $\underbrace{x_1 \wedge x_2 \wedge \neg y_1}_{\beta=\{x_1, x_2, \neg y_1\}}$ then $y_2 = 1$

- Would be nice to have $\beta = \{x_1, x_2\}$ or even $\beta = \{x_1\}$

- Challenge: How do we find small β?
  - $G_\sigma(X, Y) := \varphi(X, Y) \wedge x_1 \wedge x_2 \wedge \neg y_1 \wedge \neg y_2$
  - $\beta :=$ Literals in UNSAT Core of $G_\sigma(X, Y)$

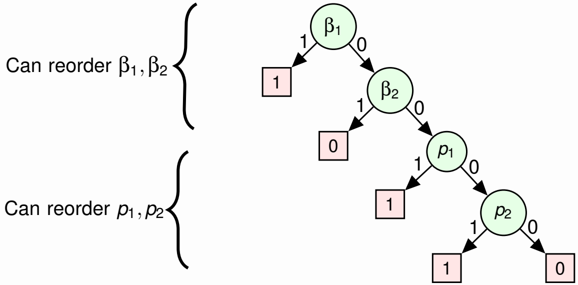
Repair: Adding Level to Decision List

- Candidates are from one level decision list:
  - Say we have paths p_1, p_2 with the leaf node label as 1.
  - Learned decision tree: If p_1 then 1, elif p_2 then 1, else 0.
  - p_1, p_2 can be reordered.

Repair: Adding Level to Decision List

- Candidates are from one level decision list:
  - Say we have paths p_1, p_2 with the leaf node label as 1.
  - 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, $\ldots \beta_2$ then 0

  ……..
- β_1 and β_2 can be reordered.
- From one-level decision list to two-level decision list.



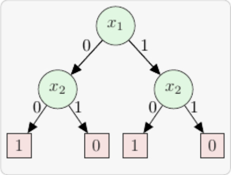
Manthan

$\varphi(X, Y)$
$X = \{x_1, x_2\}$
$Y = \{y_1, y_2\}$

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 → $G_\sigma(X, Y)$

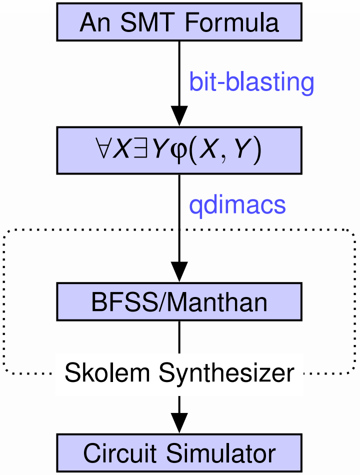
UNSAT Core-based Repair

UNSAT ↓

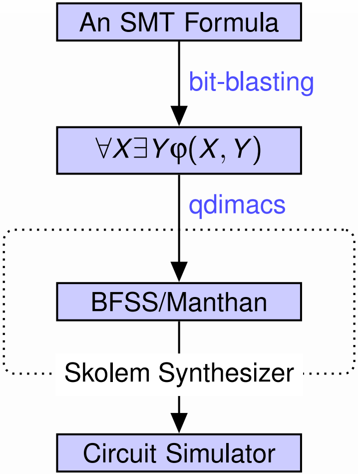
Return F

Outline

Tool Demo: Pipeline



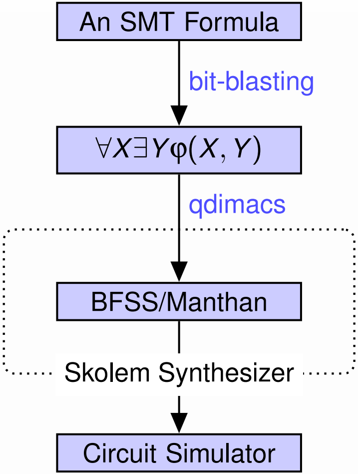
Tool Demo: Pipeline




```
(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)) (= inp3 (out inp1 inp2) )))
(check-sat)
```

An SMT formula

Tool Demo: Pipeline



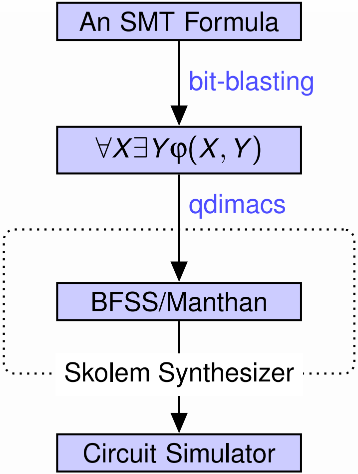

```
(set-logic BV)
;; out function with two 2 bit arguments
(declare-fun out ( (_ BitVec 2) (_ BitVec 2)) (_ BitVec 2))
;; declaring the constant
(declare-const inp1 (_ BitVec 2))
(declare-const inp2 (_ BitVec 2))
;; output of out function should be greater than or equal to first input
(assert (bvuge (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)) (= inp3 (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

Tool Demo: Pipeline

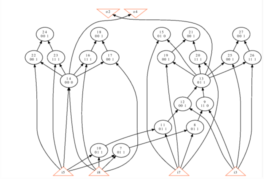



```
(set-logic BV)
;; out function with two 2 bit arguments
(declare-fun out ( (_ BitVec 2) (_ BitVec 2)) (_ BitVec 2))
;; declaring the constant
(declare-const inp1 (_ BitVec 2))
(declare-const inp2 (_ BitVec 2))
;; output of out function should be greater than or equal to first input
(assert (bvuge (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)) (= inp3 (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

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 envlope
- An exciting and diverse set of approaches
  - Knowledge compilation
  - Guess, check, and repair
- 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?

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

Additional Slides

A Quick Aside

Many questions required solving QBF.

But how do we go from QBF to 2-QBF?

Application Domain 2: Games and planning from QBF to 2-QBF/dep-QBF

A Quick Aside

Many questions required solving QBF.

But how do we go from QBF to 2-QBF?

Two simple ways

1. QBF to 2-QBF by repeated substitutions of Skolem functions!

1. Remove inner most quantifier alternation.
2. Substitute Skolem function.

- E.g., if $\Psi = \forall \mathbf{X}_1 \exists \mathbf{Y}_1 \forall \mathbf{X}_2 \exists \mathbf{Y}_2 \varphi$, consider 2-QBF formula $\Psi' = \forall \mathbf{X}_1 \forall \mathbf{Y}_1 \forall \mathbf{X}_2 \exists \mathbf{Y}_2 \varphi$
- Synthesize Skolem fns F for $\mathbf{Y}_2$ in terms of $\mathbf{X}_1, \mathbf{Y}_1, \mathbf{X}_2$. Let $\varphi_1 = \varphi[\mathbf{Y}_2 \mapsto F]$.

1. QBF to 2-QBF by repeated substitutions of Skolem functions!

1. Remove inner most quantifier alternation.
2. Substitute Skolem function.

- E.g., if $\Psi = \forall \mathbf{X}_1 \exists \mathbf{Y}_1 \forall \mathbf{X}_2 \exists \mathbf{Y}_2 \varphi$, consider 2-QBF formula $\Psi' = \forall \mathbf{X}_1 \forall \mathbf{Y}_1 \forall \mathbf{X}_2 \exists \mathbf{Y}_2 \varphi$
- Synthesize Skolem fns F for $\mathbf{Y}_2$ in terms of $\mathbf{X}_1, \mathbf{Y}_1, \mathbf{X}_2$. Let $\varphi_1 = \varphi[\mathbf{Y}_2 \mapsto F]$.
- Observe: $\Psi \equiv \forall \mathbf{X}_1 \exists \mathbf{Y}_1 \forall \mathbf{X}_2 \varphi_1$

1. QBF to 2-QBF by repeated substitutions of Skolem functions!

1. Remove inner most quantifier alternation.
2. Substitute Skolem function.
3. Use Double Negation

- E.g., if $\Psi = \forall \mathbf{X}_1 \exists \mathbf{Y}_1 \forall \mathbf{X}_2 \exists \mathbf{Y}_2 \varphi$, consider 2-QBF formula $\Psi' = \forall \mathbf{X}_1 \forall \mathbf{Y}_1 \forall \mathbf{X}_2 \exists \mathbf{Y}_2 \varphi$
- Synthesize Skolem fns F for $\mathbf{Y}_2$ in terms of $\mathbf{X}_1, \mathbf{Y}_1, \mathbf{X}_2$. Let $\varphi_1 = \varphi[\mathbf{Y}_2 \mapsto F]$.
- Observe: $\Psi \equiv \forall \mathbf{X}_1 \exists \mathbf{Y}_1 \forall \mathbf{X}_2 \varphi_1 \equiv \exists \mathbf{X}_1 \forall \mathbf{Y}_1 \exists \mathbf{X}_2 \neg \varphi_1$

Application Domain 2: Games and planning from QBF to 2-QBF/dep-QBF

1. QBF to 2-QBF by repeated substitutions of Skolem functions!

1. Remove inner most quantifier alternation.
2. Substitute Skolem function.
3. Use Double Negation
4. Continue

- E.g., if $\Psi = \forall \mathbf{X}_1 \exists \mathbf{Y}_1 \forall \mathbf{X}_2 \exists \mathbf{Y}_2 \varphi$, consider 2-QBF formula $\Psi' = \forall \mathbf{X}_1 \forall \mathbf{Y}_1 \forall \mathbf{X}_2 \exists \mathbf{Y}_2 \varphi$
- Synthesize Skolem fns F for $\mathbf{Y}_2$ in terms of $\mathbf{X}_1, \mathbf{Y}_1, \mathbf{X}_2$. Let $\varphi_1 = \varphi[\mathbf{Y}_2 \mapsto F]$.
- Observe: $\Psi \equiv \forall \mathbf{X}_1 \exists \mathbf{Y}_1 \forall \mathbf{X}_2 \varphi_1 \equiv \exists \mathbf{X}_1 \forall \mathbf{Y}_1 \exists \mathbf{X}_2 \neg \varphi_1$
- Synthesize Skolem fns G for $\mathbf{X}_2$, let $\varphi_2 = \neg \varphi_1[\mathbf{X}_2 \mapsto G]$.

1. QBF to 2-QBF by repeated substitutions of Skolem functions!

1. Remove inner most quantifier alternation.
2. Substitute Skolem function.
3. Use Double Negation
4. Continue

- E.g., if $\Psi = \forall \mathbf{X}_1 \exists \mathbf{Y}_1 \forall \mathbf{X}_2 \exists \mathbf{Y}_2 \varphi$, consider 2-QBF formula $\Psi' = \forall \mathbf{X}_1 \forall \mathbf{Y}_1 \forall \mathbf{X}_2 \exists \mathbf{Y}_2 \varphi$
- Synthesize Skolem fns F for $\mathbf{Y}_2$ in terms of $\mathbf{X}_1, \mathbf{Y}_1, \mathbf{X}_2$. Let $\varphi_1 = \varphi[\mathbf{Y}_2 \mapsto F]$.
- Observe: $\Psi \equiv \forall \mathbf{X}_1 \exists \mathbf{Y}_1 \forall \mathbf{X}_2 \varphi_1 \equiv \exists \mathbf{X}_1 \forall \mathbf{Y}_1 \exists \mathbf{X}_2 \neg \varphi_1$
- Synthesize Skolem fns G for $\mathbf{X}_2$, let $\varphi_2 = \neg \varphi_1[\mathbf{X}_2 \mapsto G]$.
- Then: $\Psi \equiv \exists \mathbf{X}_1 \forall \mathbf{Y}_1 \varphi_2 \equiv \forall \mathbf{X}_1 \exists \mathbf{Y}_1 \neg \varphi_2$ which is in 2-QBF

Application Domain 2: Games and planning from QBF to 2-QBF/dep-QBF

1. QBF to 2-QBF by repeated substitutions of Skolem functions!

1. Remove inner most quantifier alternation.
2. Substitute Skolem function.
3. Use Double Negation
4. Continue

- E.g., if $\Psi = \forall \mathbf{X}_1 \exists \mathbf{Y}_1 \forall \mathbf{X}_2 \exists \mathbf{Y}_2 \varphi$, consider 2-QBF formula $\Psi' = \forall \mathbf{X}_1 \forall \mathbf{Y}_1 \forall \mathbf{X}_2 \exists \mathbf{Y}_2 \varphi$
- Synthesize Skolem fns F for $\mathbf{Y}_2$ in terms of $\mathbf{X}_1, \mathbf{Y}_1, \mathbf{X}_2$. Let $\varphi_1 = \varphi[\mathbf{Y}_2 \mapsto F]$.
- Observe: $\Psi \equiv \forall \mathbf{X}_1 \exists \mathbf{Y}_1 \forall \mathbf{X}_2 \varphi_1 \equiv \exists \mathbf{X}_1 \forall \mathbf{Y}_1 \exists \mathbf{X}_2 \neg \varphi_1$
- Synthesize Skolem fns G for $\mathbf{X}_2$, let $\varphi_2 = \neg \varphi_1 [\mathbf{X}_2 \mapsto G]$.
- Then: $\Psi \equiv \exists \mathbf{X}_1 \forall \mathbf{Y}_1 \varphi_2 \equiv \forall \mathbf{X}_1 \exists \mathbf{Y}_1 \neg \varphi_2$ which is in 2-QBF

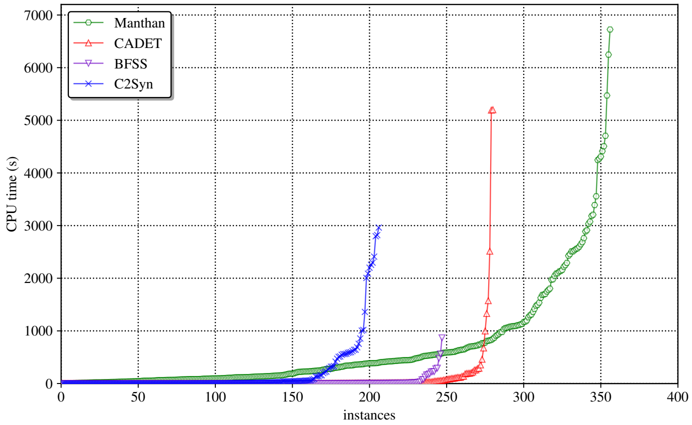
2. QBF to Dep-QBF by exploiting dependencies!

Every QBF formula is equivalent to a (2-)dep-QBF formula!
E.g., $\forall \mathbf{X}_1 \exists \mathbf{Y}_1 \forall \mathbf{X}_2 \exists \mathbf{Y}_2 \forall \mathbf{X}_3 \exists \mathbf{Y}_3 \varphi \equiv \forall \mathbf{X}_1 \forall \mathbf{X}_2 \forall \mathbf{X}_3 \exists^{\{x_1\}} \mathbf{Y}_1 \exists^{\{x_1, y_1, x_2\}} \mathbf{Y}_2 \exists^{\{x_1, y_1, x_2, y_2, x_3\}} \mathbf{Y}_3$.

Experimental Evaluations

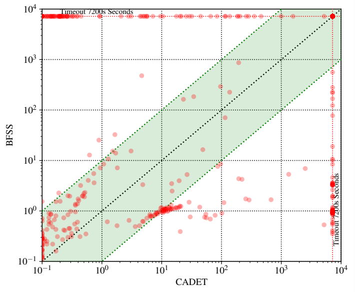
- 609 Benchmarks from:
  - QBFEval competition (`http://www.qbflib.org/`)
  - Arithmetic functions (Tabajara, Vardi,'2017)
  - Disjunctive decomposition (Akshay et al. '2017)
  - Factorization(Akshay et al. '2017)
- Compared among different state-of-the-art tools:
  - CADET (Rabe et al.'2019)
  - C2Syn (Chakraborty et al.' 2019)
  - BFSS (Akshay et al. '2018)
  - Manthan (Golia et al.' '2020,'2021).
- Timeout: 7200 seconds.

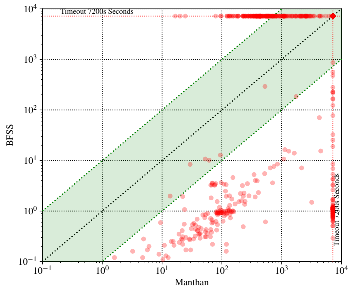
Experimental Evaluations: SOTA'20

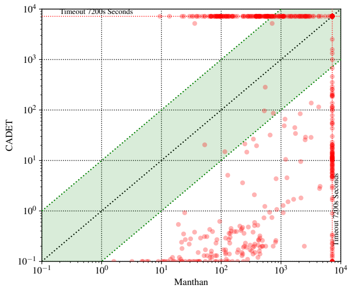


C2Syn	BFSS	CADET	Manthan
206	247	280	356

Experimental Evaluations: SOTA'20





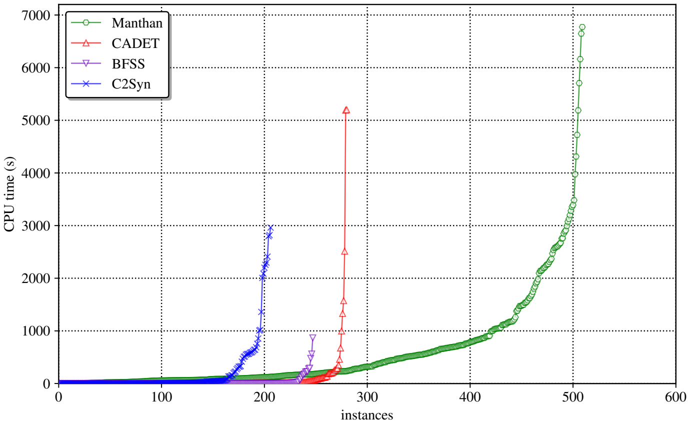


- BFSS \ CADET = 67
- CADET \ BFSS = 100

- BFSS \ Manthan = 85
- Manthan \ BFSS = 194

- CADET \ Manthan = 111
- Manthan \ CADET = 187

Experimental Evaluations: SOTA'21



C2Syn	BFSS	CADET	Manthan
206	247	280	509