УНИВЕРСИТЕТ ИТМО

Синтез на основе синтаксиса (Syntax-guided synthesis)

Чухарев Константин // Университет ИТМО

Санкт-Петербург, 2020

Сириус
Образовательный центр

ITMO UNIVERSITY

Saint Petersburg, Russia

Syntax-guided synthesis (SyGuS)

Chukharev Konstantin // ITMO University

This logo should also be in English...

Сириус

Образовательный центр

Saint Petersburg, 2020

Syntax-Guided Program Synthesis

$$\exists f \in \mathrm{Exp}.\, \forall x.\, \mathrm{Spec}(f, x)$$

Spec – logical formula that captures the desired functionality of ***f***

$$(f(x,y) \geq x) \wedge (f(x,y) \geq y) \wedge (f(x,y) \in \{x, y\})$$

[Semantics]

Exp – set of expressions specified by a context-free grammar that captures the candidate implementations of ***f***

$$\begin{aligned}
\texttt{fInt} &\ \texttt{::=}\ x \mid 0 \mid 1 \mid +(\texttt{fInt}, \texttt{fInt}) \mid \texttt{ite}(\texttt{fBool},\texttt{fInt},\texttt{fInt})\\
\texttt{fBool} &\ \texttt{::=}\ >(\texttt{fInt}, \texttt{fInt}) \mid =(\texttt{fInt}, \texttt{fInt}) \mid \neg(\texttt{fBool})
\end{aligned}$$

[Syntax]

Applications

- ➢ Programming by examples (PBE)
  - ↳ Given some inputs and outputs, find a desired function
- ➢ Superoptimizing compiler
  - ↳ Replace code with more efficient (yet equivalent!) code
- ➢ Side-channel attacks on cryptographic circuits
  - ↳ Auto-synthesize FSA-attack-resilient cryptographic circuits
- ➢ Template-guided invariant generation
- ➢ … many more …

Rajeev Alur, Rishabh Singh, Dana Fisman, and Armando Solar-lezama. Search-based Program Synthesis, 2018. DOI: 10.1145/3208071. Review article URL: https://sygus.org/assets/pdf/CACM'18_Search-based_Program_Synthesis.pdf

Programming by Examples (PBE)

```
A ::= 0 | 1 | x | (bvnot A)
    | (shl1 A) | (shr1 A) | (shr4 A) |(shr16 A)
    | (bvand A A) | (bvor A A) | (bvxor A A)
    | (bvadd A A) | (if0 A A A)
```

Grammar

Examples
(inputs and outputs)

```
∃f.
 f(#x28085a970e13e12c) = #x28085a970e13e12d
 f(#xbe5341bebd2a0749) = #xbe5341bebd2a0749
 …
 f(#xcd67bd5beaac575e) = #xcd67bd5beaac575e
```

Solution
(size = 29)

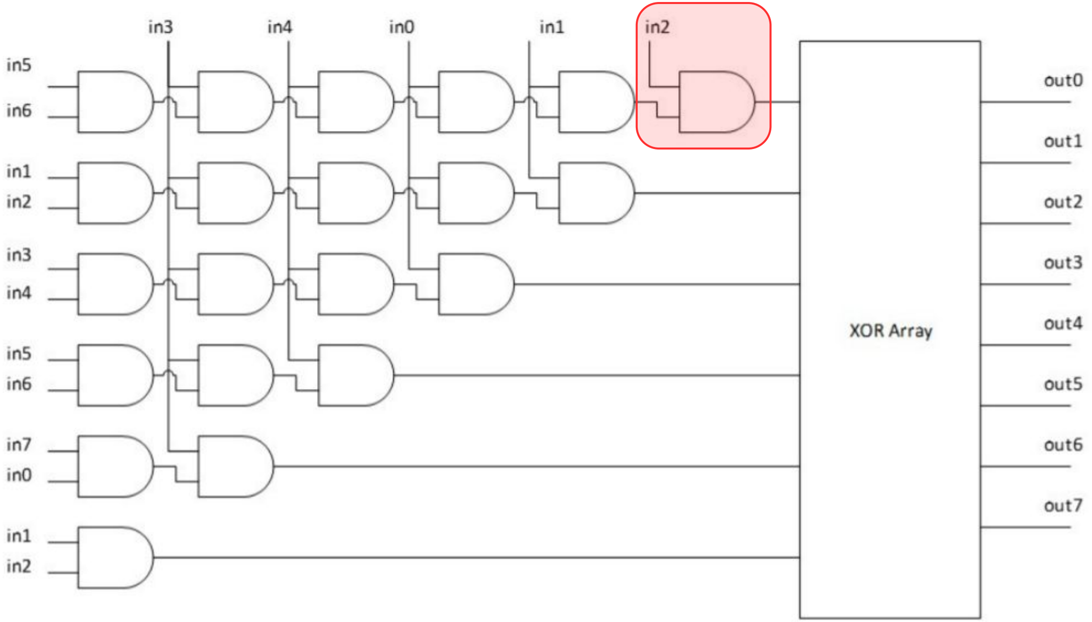
```
f := λx.
(if0 (bvand (bvnot x) #x0000000000000001)
(if0 (bvand (bvnot (shr4 x)) #x0000000000000001)
(if0 (bvand (shr1 (shr16 x)) #x0000000000000001)
(if0 (bvand (bvnot (shr1 x)) #x0000000000000001) (bvor #x0000000000000001 x) x)
(if0 (bvand (bvnot (shr1 x)) #x0000000000000001) x
(if0 (bvand (shr16 x) #x0000000000000001) x (bvor #x0000000000000001 x))))
(if0 (bvand (shr1 (shr16 x)) #x0000000000000001) (bvor #x0000000000000001 x) x))
(bvor #x0000000000000001 x))
```

Superoptimizing Compiler

```
multiply (x[1,n], y[1,n]) {
  x1 = x[1,n/2];
  x2 = x[n/2+1, n];
  y1 = y[1, n/2];
  y2 = y[n/2+1, n];
  a = x1 * y1;
  b = shift( x1 * y2, n/2);
  c = shift( x2 * y1, n/2);
  d = shift( x2 * y2, n);
  return ( a + b + c + d)
}
```

Replace with equivalent code with only 3 multiplications

Side-channel Attacks on Cryptographic Circuits



Countermeasures:

1. Add delays to inputs
   ↳ Verification problem
2. Auto-synthesize
   ↳ the equivalent circuit **(semantic constraint)**,
   ↳ where all input-to-output paths have the same length **(syntactic constraint)**

Ghalaty, N.F. Analyzing and eliminating the causes of fault sensitivity analysis, 2014. DOI: 10.7873/date.2014.217.

Syntax-Guided Program Synthesis

$$\exists f \in \mathrm{Exp}.\, \forall x.\, \mathrm{Spec}(f, x)$$

Spec – logical formula that captures the desired functionality of ***f***

$$(f(x,y) \geq x) \wedge (f(x,y) \geq y) \wedge (f(x,y) \in \{x,y\})$$

[Semantics]

Exp – set of expressions specified by a context-free grammar that captures the candidate implementations of ***f***

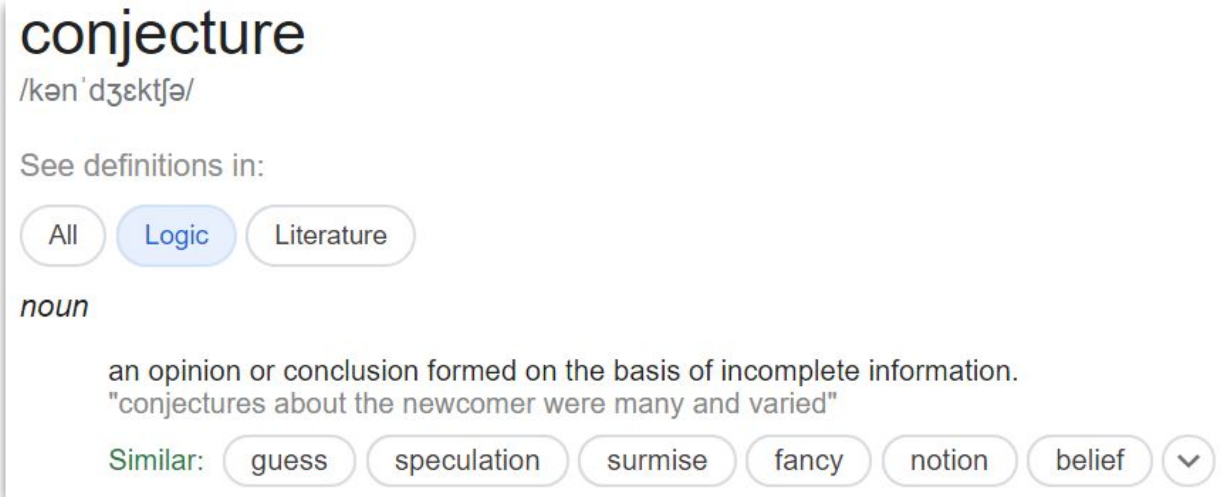
$$\texttt{fInt} ::= x \mid 0 \mid 1 \mid +(\texttt{fInt},\texttt{fInt}) \mid \texttt{ite}(\texttt{fBool},\texttt{fInt},\texttt{fInt})$$
$$\texttt{fBool} ::= >(\texttt{fInt},\texttt{fInt}) \mid =(\texttt{fInt},\texttt{fInt}) \mid \neg(\texttt{fBool})$$

[Syntax]

Synthesis Conjectures

$$\exists f.\, \forall x.\, P(f, x)$$

“There exists a function f for which property P holds for all x”



conjecture

/kənˈdʒɛktʃə/

See definitions in:

noun

an opinion or conclusion formed on the basis of incomplete information.

"conjectures about the newcomer were many and varied"

Similar: guess speculation surmise fancy notion belief

Synthesis Conjectures Modulo *T*

$$\exists f.\, \forall x.\, P(f, x)$$

"There exists a function f for which property P holds for all x"

Property is in some **background theory *T***, for example:

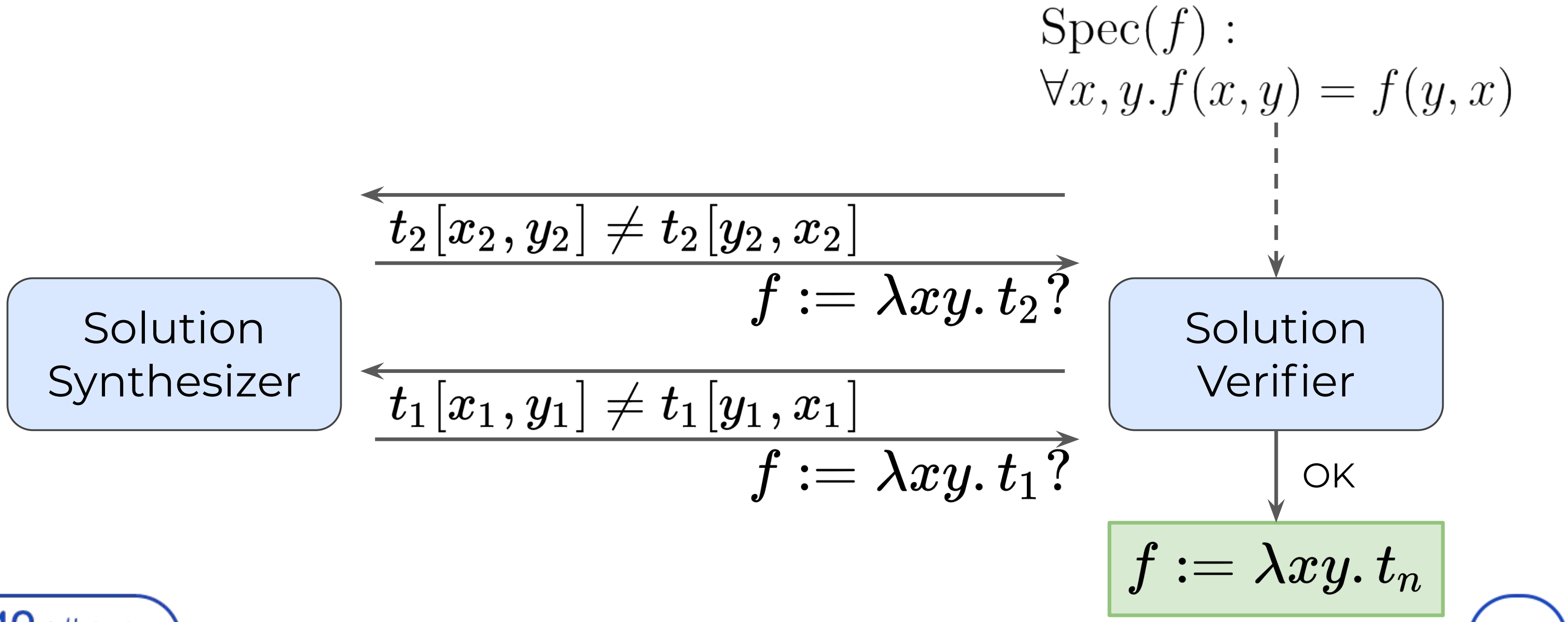
- Linear or non-linear arithmetics (LIA, NIA)
- Fixed-width bit-vectors (BV)
- Strings
- …

⇒ Satisfiability Modulo Theories (SMT)

Approaches to Synthesis

- Naïve
  - Let f – free uninterpreted function (UF)
  - Use SMT solvers to find a model for f
  - Hard for SMT solvers :c
- Counterexample-guided Inductive Synthesis (CEGIS)
  - Enumerative
  - Symbolic
  - Stochastic
- Counterexample-guided quantifier instantiation (CEGQI)
  - https://doi.org/10.1007/978-3-319-21668-3_12

Counterexample-guided Inductive Synthesis (CEGIS)



$\text{Spec}(f):$
$\forall x, y. f(x, y) = f(y, x)$
$t_2[x_2, y_2] \neq t_2[y_2, x_2]$
$f := \lambda xy.\, t_2?$
Solution Synthesizer
Solution Verifier
$t_1[x_1, y_1] \neq t_1[y_1, x_1]$
$f := \lambda xy.\, t_1?$
OK
$f := \lambda xy.\, t_n$

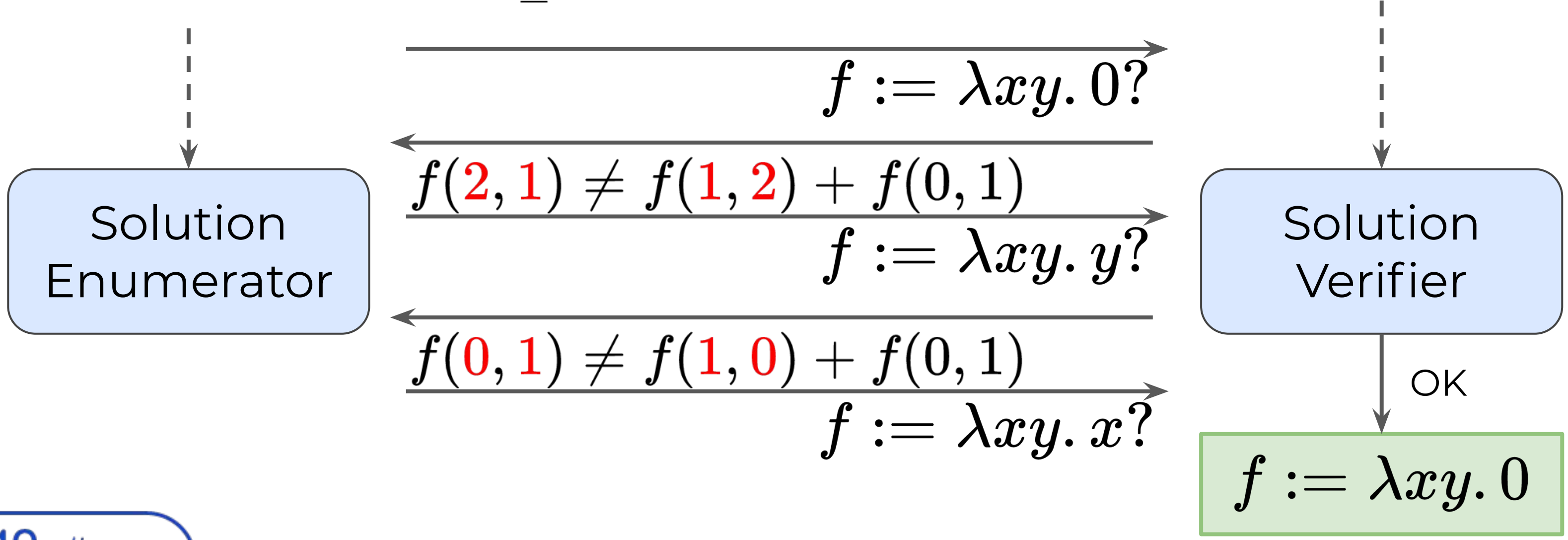
Enumerative CEGIS

$\mathrm{Syntax}(f)$:

```
A ::= A+A | -A | x | y | 0 | 1 | ite(B,A,A)
B ::= B∧B | ¬B | A=A | A≥A | ⊥
```

$\mathrm{Spec}(f)$:

$\forall x, y. f(x,y) = f(y,x) + f(0,1)$



Solution Enumerator

$f := \lambda xy.\, 0?$

$f({\color{red}2},{\color{red}1}) \neq f({\color{red}1},{\color{red}2}) + f(0,1)$

$f := \lambda xy.\, y?$

$f({\color{red}0},{\color{red}1}) \neq f({\color{red}1},{\color{red}0}) + f(0,1)$

$f := \lambda xy.\, x?$

Solution Verifier

OK

$f := \lambda xy.\, 0$

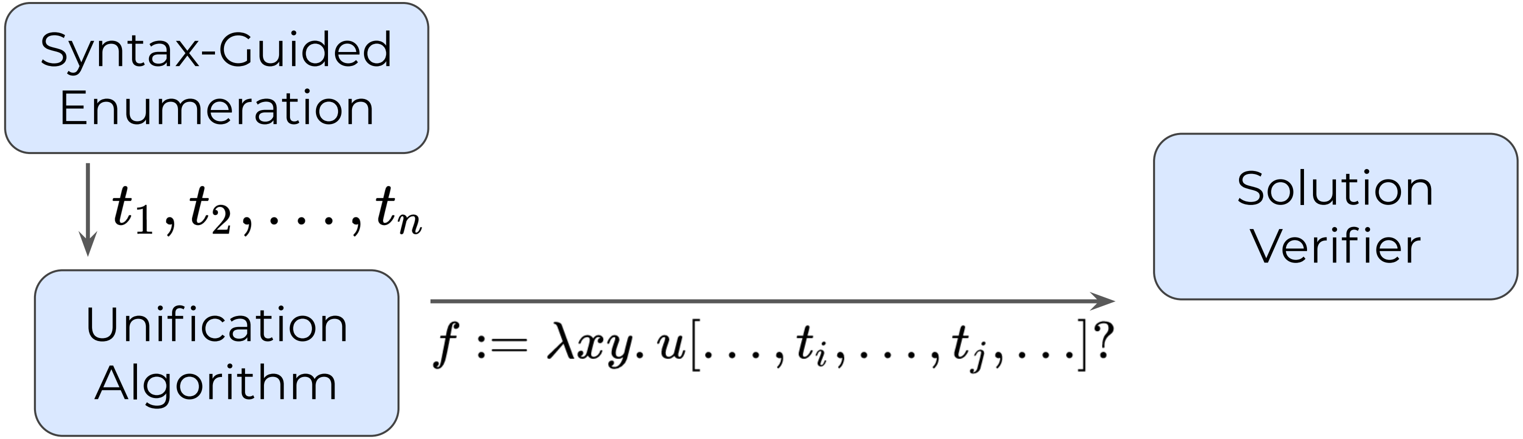
Enumerative CEGIS + Unification

$\mathrm{Syntax}(f)$:

```
A ::= A+A | -A | x | y | 0 | 1 | ite(B,A,A)
B ::= B∧B | ¬B | A=A | A≥A | ⊥
```

$\mathrm{Spec}(f)$:

$\forall x, y. f(x, y) = f(y, x)$



Syntax-Guided Enumeration

$t_1, t_2, \ldots, t_n$

Unification Algorithm

$f := \lambda xy.\, u[\ldots, t_i, \ldots, t_j, \ldots]?$

Solution Verifier

CEGIS using SMT Solver

$\text{Syntax}(f):$

```
A ::= A+A | -A | x | y | 0 | 1 | ite(B,A,A)
B ::= B∧B | ¬B | A=A | A≥A | ⊥
```

$\text{Spec}(f):$

$\forall x, y. f(x, y) = f(y, x)$

Syntax-Guided Enumeration

Unification Algorithm

Solution Verifier

SMT Solver

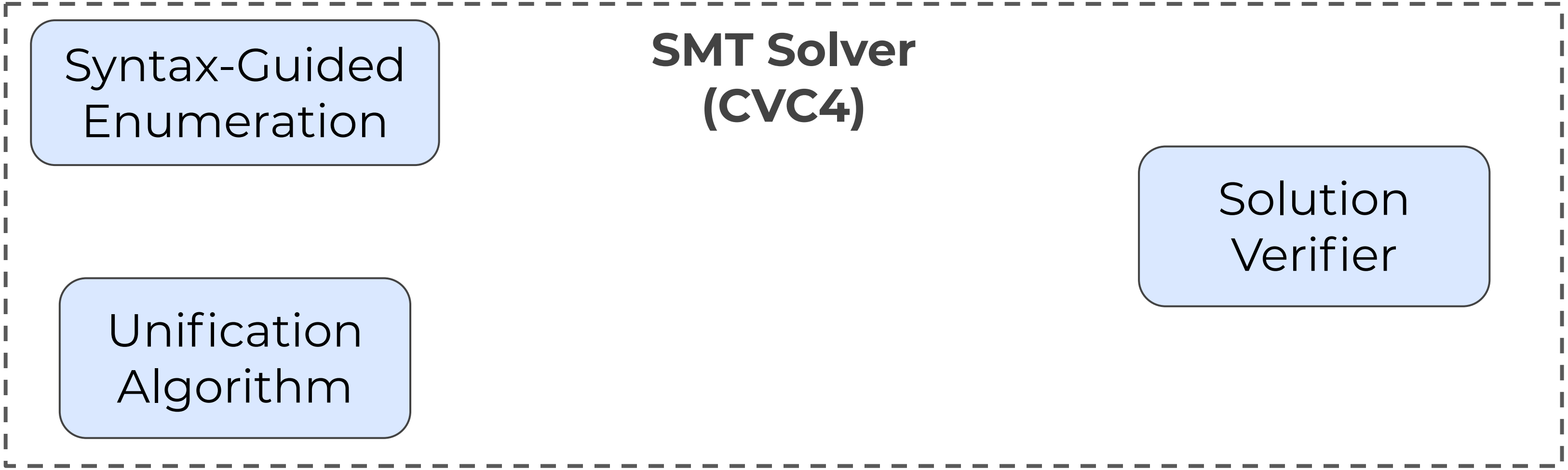
CEGIS inside an SMT Solver

$\text{Syntax}(f):$

```
A ::= A+A | -A | x | y | 0 | 1 | ite(B,A,A)
B ::= B∧B | ¬B | A=A | A≥A | ⊥
```

$\text{Spec}(f):$

$$\forall x, y. f(x,y) = f(y,x)$$

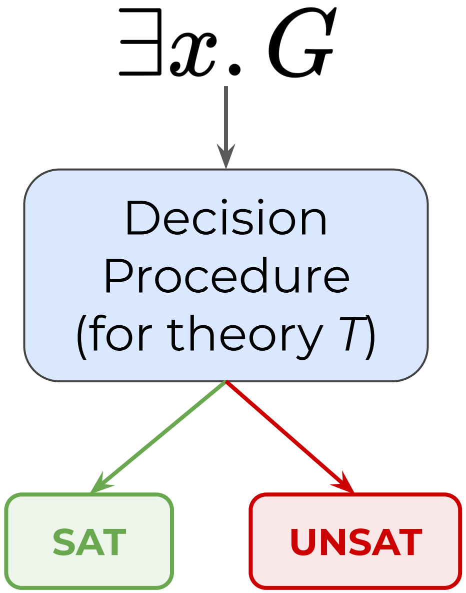


Enumerative Synthesis in SMT Solvers

`synth(∃ƒ.∀x.P(f,x))`:

1. Syntax-guided enumeration
   - ↳ Construct a set of “interesting” terms {t_1 … t_n}
2. Unification algorithms (https://doi.org/10.1007/978-3-662-54577-5_18)
   - ↳ From {t_1 … t_n}, construct candidate *u* for *ƒ*
3. (Decision) Procedures
   - ↳ Check whether *u* satisfies the specification:
     “Is ¬`∀x.P(u,x)` *T*-unsatisfiable?” ⇒ ¬`P(u,`x_m`)`

From Decision Procedures to Synthesis Procedures



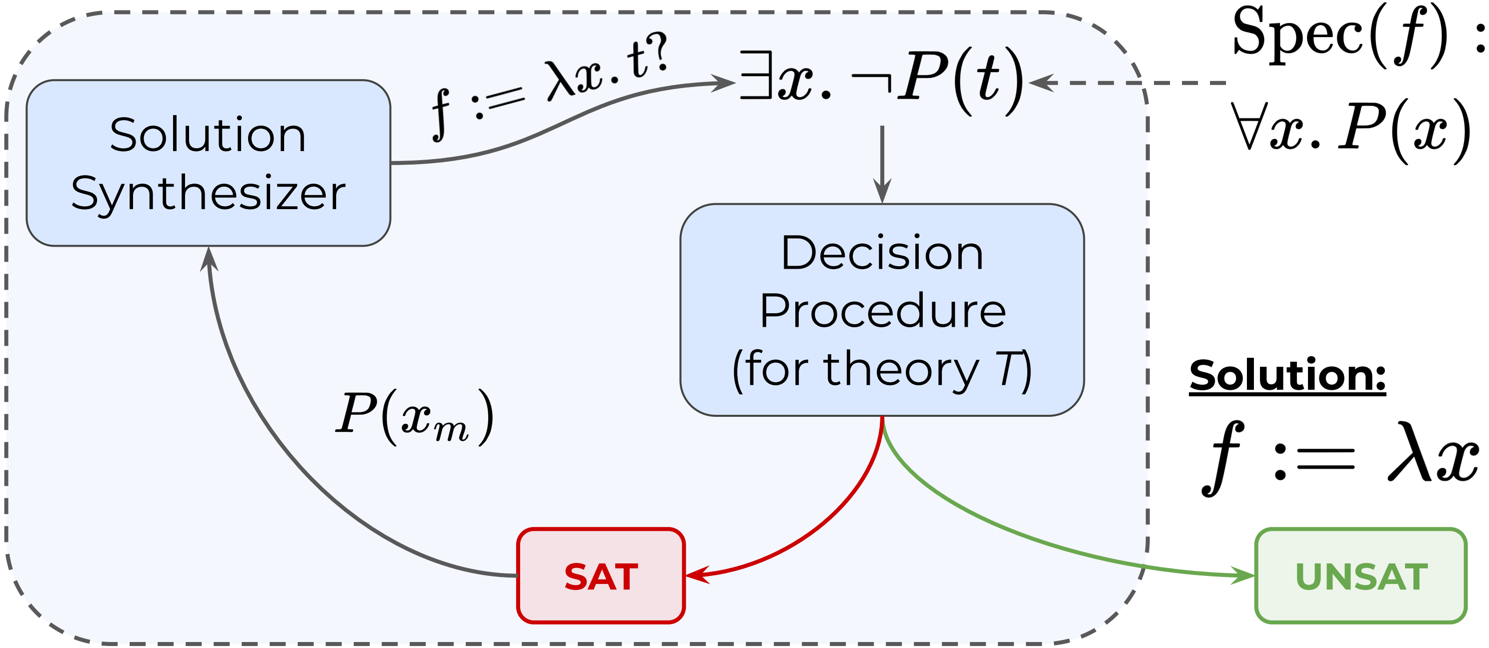
$$\exists x.\, G$$

Decision Procedure (for theory *T*)

Model:

$$x = x_m \models G$$

From Decision Procedures to Synthesis Procedures



$\text{Spec}(f)$:

$\forall x.\, P(x)$

Solution:

$f := \lambda x.\, t$

Ufff... Enough theory.

Demo time

Links

- https://sygus.org/
  - SyGuS specification: https://sygus.org/assets/pdf/SyGuS-IF_2.0.pdf
- https://cvc4.github.io/
  - Online playground: https://cvc4.github.io/app
- http://smtlib.cs.uiowa.edu/
  - Logics: http://smtlib.cs.uiowa.edu/logics.shtml
  - SMT specification: http://smtlib.cs.uiowa.edu/language.shtml
- ...

(Minimal) Boolean Formula Synthesis

```
(synth-fun f ((x Bool) (y Bool) (z Bool)) Bool
    ((Start Bool)) (
    (Start Bool (
        x y z
        (not Start)
        (and Start Start)
        (or Start Start)
    ))
))
```

https://gist.github.com/Lipen/1bbf6586ad180ef8334a5e5a7efa1576

```
; f(x,y,z) = x|(y&z) = 00011111
(constraint (= (f’ 0 0 0) 0 ))
(constraint (= (f’ 0 0 1) 0 ))
(constraint (= (f’ 0 1 0) 0 ))
(constraint (= (f’ 0 1 1) 1 ))
(constraint (= (f’ 1 0 0) 1 ))
(constraint (= (f’ 1 0 1) 1 ))
(constraint (= (f’ 1 1 0) 1 ))
(constraint (= (f’ 1 1 1) 1 ))

; Solution: (or x (and y z))
```

Thanks for your attention.

kchukharev@itmo.ru

Lipen

ITsMOre than a
UNIVERSITY