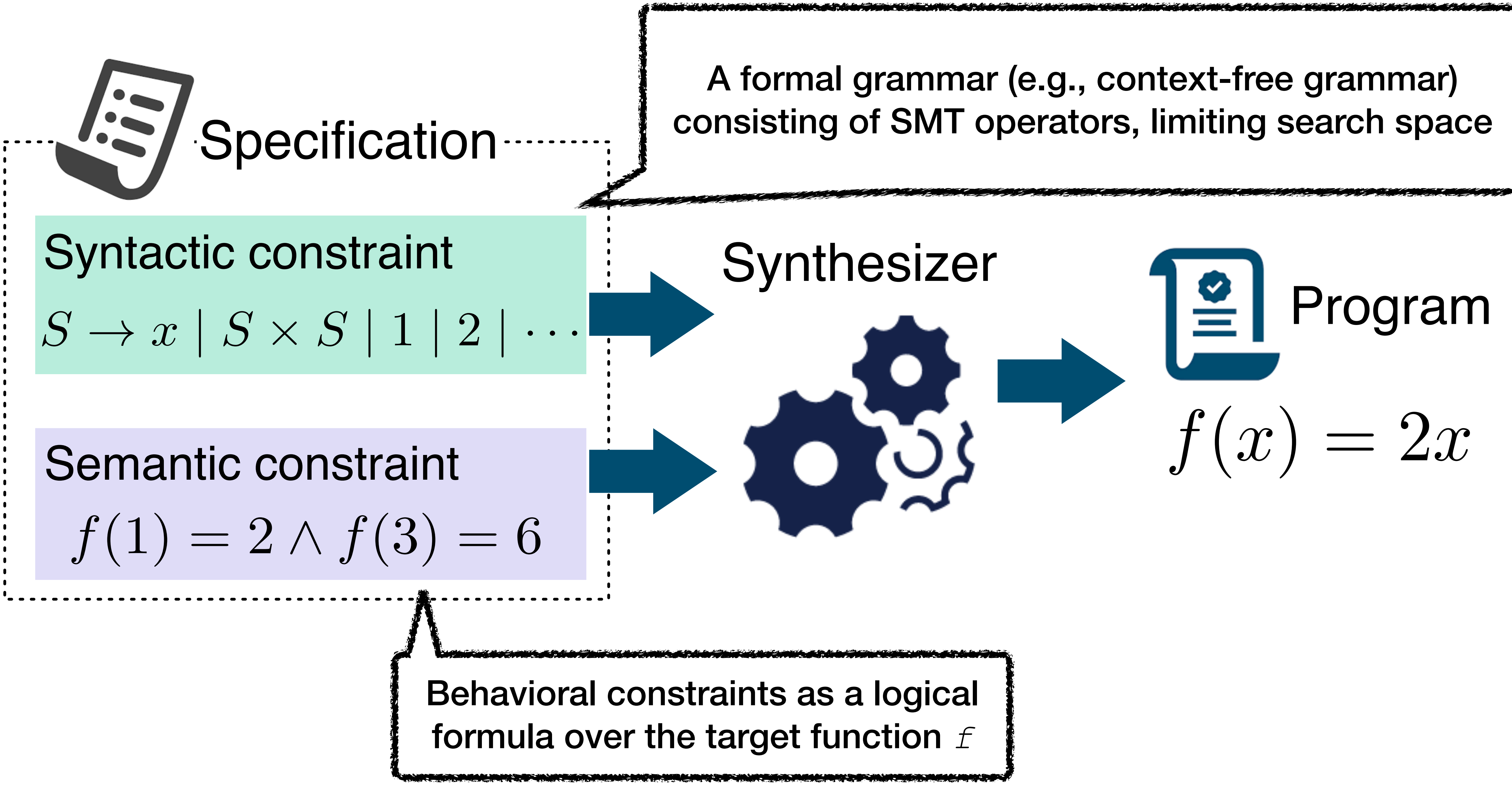
Syntax-Guided Synthesis

Woosuk Lee

CSE9116 SPRING 2024

Hanyang University

Syntax-Guided Program Synthesis (SyGuS)†



†http://www.sygus.org

Preliminaries for Understanding SyGuS

- SAT / SMT
- Context free grammar / Regular tree grammar
- General *decidability*

Propositional Logic

- True (T), False (F), Boolean variables (p, q, r, …)
- Connectives: $\neg$ (not), $\wedge$ (and), $\vee$ (or)
  - e.g., $(\neg p \wedge T) \vee (q \Rightarrow F)$

$\Rightarrow$: $a \Rightarrow b \equiv \neg a \vee b$

- Logic for determining satisfiability of Boolean formulas

Interpretation

- Interpretation I for Boolean formula Q: Boolean assignments to variables in Q
  - e.g., I : { p ↦ T, q ↦ F , … }
- Satisfying interpretation I for Q (denoted Q ⊧ F): Interpretation of Q that makes Q true
  - What is a satisfying interpretation of $Q \equiv (\neg p \wedge T) \vee (\neg q \vee F)$?

Satisfiability and *Validity*

- Q is *satisfiable* if and only if
  - A satisfying interpretation of Q exists (i.e., $\exists I.\ I \models Q$)
- Q is *valid* if and only if
  - All interpretations of Q are satisfying (i.e., $\forall I.\ I \models Q$)

Satisfiability and *Validity*

- Satisfiability and validity are dual
  - “Q is valid” ≡ “¬ Q is *unsatisfiable*”

Boolean Satisfiability Problem (SAT)

- For a given Boolean formula Q, determine if there exists a satisfying interpretation of Q
  - e.g., $Q \equiv (\neg p \wedge T) \vee (\neg q \vee F) \quad \rightarrow \quad I: \{ p \mapsto F, q \mapsto F \}$
- Algorithms for solving SAT:
  - DPLL, CDCL(conflict-driven clause learning), WalkSAT(stochastic local search), …
- NP-complete (hardest among problems that are solvable in polynomial time when you are very lucky)

First-Order Logic

- Formulas may contain
  - Quantifiers (∀ forall, ∃ exist)
  - Constants and variables of certain types (e.g,. integer)
  - N-ary functions taking N arguments

Example

- $x \leq 3 \land 0 \leq x$
- $\exists\, x.\ x >> 0x01 = 0x02$
- $\forall\, k\ .\ 0 \leq k < n \Rightarrow A[k] \leq A[k+1]$

Satisfiability Modulo Theory (SMT)

- SAT: determining satisfiability of propositional logic formula
  - Interpretation: variables → {T, F}
- SMT: determining satisfiability of first-order logic formula
  - Interpretation: variables → domain of discourse (e.g., integer, rational, bit-vector, …)

Example

$$i = j + 3 \wedge f(i + 3) \neq f(j + 6)$$

Example

$$i = j + 3 \wedge f(i+3) \neq f(j+6)$$

$$unsat$$

Example

$$x = y - 4 \wedge f(x+2) \neq f(y-1)$$

Example

$$x = y - 4 \land f(x+2) \neq f(y-1)$$

$$sat \quad \text{if} \quad \begin{array}{l} x = -2 \\ y = 2 \\ f(0) = 1 \\ f(1) = 3 \end{array}$$

Algorithms for Solving SMT

- Basically relying on SAT solvers
- + a theory about a topic

Theory ?

- *signature* + *axiom*
- Theory of equality with uninterpreted function
  - *signature:* $\{ f, g, h, \ldots, = \}$
  - *axiom:*
    - $\forall x.\ x = x$
    - $\forall x, y.\ x = y \rightarrow y = x$
    - $\forall x, y, z.\ x = y \wedge y = z \rightarrow x = z$
    - $\forall x_1, \ldots, x_n, y_1, \ldots, y_n.\ (x_1 = y_1 \wedge \ldots \wedge x_n = y_n) \rightarrow (f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n))$

Theory of Linear Integer Arithmetic

- Signature: { 0, 1, +, −, ×, ≤ }
- Axiom:
  - For every natural number a, a x 0 = 0
  - For every natural number a, a + 0 = a
  - For every natural number a, b, a + b = b + a
  - …

Theory of Arrays

- Signature: { read(…), write(…), =, … }
- Axiom
  - $\forall a, i, v.\ read(write(a, i, v), i) = v$
  - $\forall a, i, j, v.\ \neg(i = j) \rightarrow (read(write(a, i, v), j) = read(a, j))$
  - $\forall a, b.\ (\forall i.\ read(a, i) = read(b, i)) \rightarrow a = b$

Theory of Fixed-width Bitvectors

- Signature: =, variables, and
  - Arithmetic operators: bvadd, bvsub, bvmul, …
  - Logical oeprators: bvand, bvor, bvnot, …
- Axiom:
  - For every bitvector a, bvmul(a, 0x00) = 0x00
  - …

Theory of Strings

- Signature: =, variables, and
    - str.++ (string concatenation), str.len (string length), str.substr (substring extraction), str.replace (string replacement), str.indexof, …
    - Type casting operators: str.to.int, int.to.str, …
- Axiom:
    - For every strings a and b, str.++(a,b) = ab
    - …

Solving SMT

- Two approaches: eager / lazy
- Example for *theory of uninterpreted functions*
  - Let's suppose we want to solve the following SMT problem

$$g(a) = c \wedge g(b) = d \wedge a = b \wedge c \neq d$$

Eager Approach

- SMT formula → propositional logic formula
- Remove functions
  - $g(a) \rightarrow A, g(b) \rightarrow B, g(c) \rightarrow C$
- Add the following clauses (from axiom)
  - $(a = b \Rightarrow A = B) \wedge (a = c \Rightarrow A = C) \wedge (b = c \Rightarrow B = C)$
- $A = B \rightarrow (A \Rightarrow B) \wedge (B \Rightarrow A)$

Eager Approach

$$\underbrace{g(a) = c}_{1} \wedge \underbrace{g(b) = d}_{2} \wedge \underbrace{a = b}_{3} \wedge \underbrace{c \neq d}_{4}$$

$$\underbrace{A = c}_{1} \wedge \underbrace{B = d}_{2} \wedge \underbrace{a = b}_{3} \wedge \underbrace{c \neq d}_{4} \wedge \underbrace{(a = b \rightarrow A = B)}_{5}$$

CNF

$$\underbrace{\boxed{A = c}}_{1} \wedge \underbrace{B = d}_{2} \wedge \underbrace{a = b}_{3} \wedge \underbrace{c \neq d}_{4} \wedge \underbrace{(a \neq b \vee A = B)}_{5}$$

$$(A \implies c) \wedge (c \implies A) \quad \ldots$$

Invoke a SAT solver ⇨ UNSAT!

Lazy Approach

$$\underbrace{g(a)=c}_{1} \wedge (\underbrace{f(g(a)) \neq f(c)}_{2} \vee \underbrace{g(a)=d}_{3}) \wedge \underset{4}{c \neq d}$$

1. Solve$1 \wedge (2 \vee 3) \wedge 4$ and suppose we get a solution $\{1,2,4\}$

2. The theory solver tells us it cannot be a solution ($\because$ $1 \wedge 2$ violates an axiom)

3. Solve $1 \wedge (2 \vee 3) \wedge 4 \wedge \neg(1 \wedge 2 \wedge 4)$ and get $\{1, \neg 2,3,4\}$

4. The theory solver tells us it cannot be a solution ($\because$ $1 \wedge 3 \wedge 4$ violates an axiom)

5. Solve $1 \wedge (2 \vee 3) \wedge 4 \wedge \neg(1 \wedge 2 \wedge 4) \wedge \neg(1 \wedge \neg 2 \wedge 3 \wedge 4)$ and get UNSAT

SAT / SMT

- More details can be found at
  - Aaron Bradley, Zohar Manna, The Calculus of Computation
  - Armin Biere et al., Handbook of satisfiability

Formal grammar

- How to form strings from a language's *alphabet*

- E.g., For a set of alphabets {a, b} grammar G

1. $S \rightarrow aSb$

2. $S \rightarrow ba$

- defines a set of strings (denoted L(G))

{ba, abab, aababb, aaababbb, $\ldots$ }

Formal grammar

- How to form strings from a language's *alphabet*
- E.g., For a set of alphabets {a, b} grammar G

1. $S \rightarrow aSb$
2. $S \rightarrow ba$

- defines a set of strings (de

{ba, abab, aababb, aaababbb, … }

$$S \Rightarrow ba$$
$$S \Rightarrow aSb \Rightarrow abab$$
$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aababb$$

Formal grammar

- How to form strings from a language's *alphabet*

production rule

- ~~F~~or a set of alphabets {a, b} grammar G

1. $S \rightarrow aSb$
2. $S \rightarrow ba$

terminal symbol

(Starting) *nonterminal symbol*

(Leftmost) *derivation*

$$S \Rightarrow ba$$

$$S \Rightarrow aSb \Rightarrow abab$$

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aababb$$

- defines a set of strings (de

{ba, abab, aababb, aaababbb, … }

Context-free grammar

- Grammar with production rules of the following form
  - $A \to \alpha$
    (A : nonterminal symbol
    α : mixture of terminal/nonterinal symbols, i.e., *sentential form*)
- E.g., production rules for matched parentheses
  - $S \to SS$
  - $S \to (S)$
  - $S \to ()$

What is a SyGuS Problem?

- A synthesis problem formulated with
  - An SMT Theory
  - Correctness specification ϕ as a formula in the theory
  - Target function f
  - A context-free grammar

What is a SyGuS Problem?

- For given
  - Background SMT Theory T
  - Target function f and its type
  - First-order logic formula ϕ involving f (all variables are considered ∀-quantified)
  - Context-free grammar G
- Find expression $e \in L(G)$ such that
  - ϕ[e / f] is *valid* modulo theory T

Replace e with f in ϕ

What is a SyGuS Problem?

- For given
  - Background SMT Theory T

> Nowadays, a *regular tree grammar* is used instead
> (less expressive than a CFG)
> (e.g., impossible to use multiple non-terminals in series)

∀-qu ified)

  - Context-free grammar G
- Find expression e ∈ L(G) such that
  - ϕ[e / f] is *valid* modulo theory T

Example

- Goal: function `f` that takes integers x, y (`f : int x int → int`)
- Specification

Syntactic:

$$S \to S + S \mid S \times S \mid x \mid y \mid 0 \mid 1$$

Semantic:

$$\varphi : f(x,y) = f(y,x) \wedge f(x,y) \geq x$$

All variables are universally quantified: $\forall x, y.\ \varphi$

- Solution:

$$f(x,y) = x + y$$

Example

- Goal: function that returns 3x + 9 given x (`f : int → int`)
- Specification

Syntactic:

$$S \rightarrow S + S \mid S \times S \mid x \mid 1$$

Semantic:

$$f(2) = 15 \wedge f(3) = 18$$

- Solution:

$$f(x) = (1 + 1 + 1) \times (x + (1 + 1 + 1))$$

Complexity of SyGuS

- Even for SyGuS problems with the theory of uninterpreted functions (EUF) and a restricted grammar† the *worst-case complexity* is EXPTIME-complete.

$$P \subseteq NP \subseteq PSPACE \subseteq \textbf{EXPTIME} \subseteq NEXPTIME \subseteq EXPSPACE$$

- i.e., <u>hardest among problems solvable in exponential time</u>

- Since EUF is the simplest theory, it is conjectured more complex SyGuS problems will be even more difficult.

† called Regular-EUF (Caulfield et al., “What’s Decidable about Syntax-Guided Synthesis?”). Regular-EUF is a class of SyGuS problems where conditionals are not allowed and the semantic constraint is in a limited form.