EECS 598-008 & EECS 498-008: Intelligent Programming Systems

Lecture 6

Announcements

- **Remote OH** Friday 3-4pm September 17 (tomorrow)
  - “Video tutorial” of Z3 will be released today
  - Come to OH with your questions!
- A2 out yesterday, **due midnight Monday September 27**
  - A2 involves (1) learning Z3 and (2) applying Z3 for synthesis
- CFPP (call for paper presentations) will be out next Monday, due September 28
  - PC invites already sent out
  - Create HotCRP account

Today’s Agenda

- **Wrap up Logics**
- Deduction-based Pruning Techniques in Top-Down Search
- Observational Equivalence Reduction in Bottom-Up Search

First-Order Theories

- So far, propositional logic and first-order logic
  - Propositional logic is limited in expressiveness
  - FOL is more expressive, but functions are uninterpreted (can assign any meaning)

- In many cases, we want functions to have certain meanings (e.g., +, =, >)
- **Theories assign meanings to symbols**

First-Order Theories Syntax

- A first-order theory has
  - object/function/relation constants, variables, quantifiers, logical connectives (FOL)
  - **axioms (new!)**

- E.g., let's make up a first-order theory — theory of heights T_H
  - T_H has only one relation constant called $taller$ and no other constants
  - T_H has one axiom $\forall x, y \,.\, (taller(x, y) \rightarrow \neg taller(y, x))$
  - Is $\forall x \,.\, \exists y \,.\, taller(y, x)$ in T_H?
  - Is $\forall x \,.\, taller(Jack, x)$ in T_H?

First-Order Theories Semantics

- **Axioms assign meaning to symbols**
- That means: some universes/interpretations may not be consistent with axioms
  - E.g., $U = \{A, B\}, I(taller) = \{\langle A, B\rangle, \langle B, A\rangle\}$ is not consistent with the axiom $\forall x, y \,.\, (taller(x, y) \rightarrow \neg taller(y, x))$ in T_H
  - **We are only interested in those interpretations that are consistent!**
- Given U, I, formula F can be evaluated in the same way as in FOL, but we only consider interpretations that are consistent with axioms
  - … which means some formulas not valid in FOL may be valid in first-order theories

Satisfiability and Validity Modulo Theory T

- "modulo" $\approx$ "in terms of"

- Formula F is **satisfiable modulo T** if **there exists** a universe U and an interpretation I, such that (1) U, I is consistent with axioms in T, and (2) $U, I \models F$
- Formula F is **valid modulo T** if **for all** universes U and interpretations I, if U, I is consistent with axioms in T then we have $U, I \models F$

- Satisfiability Modulo Theory (SMT) solvers: Microsoft z3, CVC4, …

Satisfiability and Validity Modulo Theory T

- If F is valid in FOL, is it also valid modulo T ?
- If F is not valid in FOL, is it also not valid modulo T ?
- If F is satisfiable in FOL, is it also satisfiable modulo T ?
- If F is not satisfiable in FOL, is it also not satisfiable modulo T ?

- If F is valid modulo T, is it also valid in FOL?
- If F is not valid modulo T, is it also not valid in FOL?
- If F is satisfiable modulo T, is it also satisfiable in FOL?
- If F is not satisfiable modulo T, is it also not satisfiable in FOL?

Theory of Equality

- **Extend** FOL to include a “built-in” predicate =

- Axioms assign meaning to =

$\forall x \,.\, x = x$ (reflexivity)

$\forall x, y \,.\, (x = y \rightarrow y = x)$ (symmetry)

$\forall x, y, z \,.\, (x = y \wedge y = z \rightarrow x = z)$ (transitivity)

$\forall x_1, \ldots, x_n, y_1, \ldots, y_n \,.\, \bigwedge_i x_i = y_i \rightarrow f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)$ (function congruence)

$\forall x_1, \ldots, x_n, y_1, \ldots, y_n \,.\, \bigwedge_i x_i = y_i \leftrightarrow p(x_1, \ldots, x_n) = p(y_1, \ldots, y_n)$ (predicate congruence)

Theory of Equality

- Extend FOL to include a “built-in” predicate =
- Is $\forall x, y, z \,.\, \left(x = y \wedge y = z \rightarrow f(x) = f(z) \right)$ in theory of equality?

Theory of Equality

- Extend FOL to include a “built-in” predicate =
- Is $\forall x, y, z \,.\, \left(x = y \wedge y = z \rightarrow f(x) = f(z) \right)$ in theory of equality?
  - Is it satisfiable, unsatisfiable, valid?

Theory of Equality

- Extend FOL to include a “built-in” predicate =
- Is $\forall x, y, z \,.\, \left(x = y \wedge y = z \rightarrow f(x) = f(z) \right)$ in theory of equality?
  - Is it satisfiable, unsatisfiable, valid?
- Is $\forall x, y, z, w \,.\, \left(x = y \wedge z = w \rightarrow f(x + z) = f(y + w) \right)$ in theory of equality?

Theory of Equality

- Extend FOL to include a “built-in” predicate =
- Is $\forall x, y, z \,.\, \left(x = y \wedge y = z \rightarrow f(x) = f(z) \right)$ in theory of equality?
  - Is it satisfiable, unsatisfiable, valid?
- Is $\forall x, y, z, w \,.\, \left(x = y \wedge z = w \rightarrow f(x + z) = f(y + w) \right)$ in theory of equality?

- Undecidable (but quantifier-free fragment is decidable)

Theory of Integers

- Also known as **Linear Arithmetic over Integers**, or **Linear Integer Arithmetic** (LIA)

Theory of Integers

- Also known as **Linear Arithmetic over Integers**, or **Linear Integer Arithmetic** (LIA)
- Symbols that are allowed:
  - Object constants: $\ldots, -2, -1, 0, 1, 2, \ldots$
  - Function constants: $\ldots, -3\cdot, -2\cdot, 2\cdot, 3\cdot, \ldots, +, -$
  - Relation constants: $=, >$
  - Variables: $x, y, z, \ldots$
  - Logical connectives: same as FOL

Theory of Integers

- Also known as **Linear Arithmetic over Integers**, or **Linear Integer Arithmetic** (LIA)
- Symbols that are allowed:
  - Object constants: $\ldots, -2, -1, 0, 1, 2, \ldots$
  - Function constants: $\ldots, -3\cdot, -2\cdot, 2\cdot, 3\cdot, \ldots, +, -$
  - Relation constants: $=, >$
  - Variables: $x, y, z, \ldots$
  - Logical connectives: same as FOL
- Axioms
  - Define meaning of symbols
  - E.g., $\forall x \,.\, x + 0 = x$ (do not show the complete set of axioms here)

Theory of Integers

- Symbols that are allowed:
  - Object constants: $\ldots, -2, -1, 0, 1, 2, \ldots$
  - Function constants: $\ldots, -3\cdot, -2\cdot, 2\cdot, 3\cdot, \ldots, +, -$
  - Relation constants: $=, >$
  - Variables: $x, y, z, \ldots$
  - Logical connectives: same as FOL
- Is $\forall x, y, z, w \,.\, \Big(x = y \wedge z = w \rightarrow f(x + z) = f(y + w)\Big)$ in theory of integers?

Theory of Integers

- Symbols that are allowed:
  - Object constants: $\ldots, -2, -1, 0, 1, 2, \ldots$
  - Function constants: $\ldots, -3\cdot, -2\cdot, 2\cdot, 3\cdot, \ldots, +, -$
  - Relation constants: $=, >$
  - Variables: $x, y, z, \ldots$
  - Logical connectives: same as FOL
- Is $\forall x, y, z, w \,.\, \Big(x = y \wedge z = w \rightarrow f(x + z) = f(y + w)\Big)$ in theory of integers?
- Is $\forall x, y \,.\, \exists z \,.\, x + y = z$ in theory of integers?

Theory of Integers

- Symbols that are allowed:
  - Object constants: $\ldots, -2, -1, 0, 1, 2, \ldots$
  - Function constants: $\ldots, -3\cdot, -2\cdot, 2\cdot, 3\cdot, \ldots, +, -$
  - Relation constants: $=, >$
  - Variables: $x, y, z, \ldots$
  - Logical connectives: same as FOL
- Is $\forall x, y, z, w \,.\, \Big(x = y \wedge z = w \rightarrow f(x + z) = f(y + w)\Big)$ in theory of integers?
- Is $\forall x, y \,.\, \exists z \,.\, x + y = z$ in theory of integers?
- Is $\forall x, y \,.\, \exists z \,.\, x \cdot y = z$ in theory of integers?

Theory of Integers

- Symbols that are allowed:
  - Object constants: $\ldots, -2, -1, 0, 1, 2, \ldots$
  - Function constants: $\ldots, -3\cdot, -2\cdot, 2\cdot, 3\cdot, \ldots, +, -$
  - Relation constants: $=, >$
  - Variables: $x, y, z, \ldots$
  - Logical connectives: same as FOL
- Is $\forall x, y, z, w \,.\, \left(x = y \wedge z = w \rightarrow f(x + z) = f(y + w) \right)$ in theory of integers?
- Is $\forall x, y \,.\, \exists z \,.\, x + y = z$ in theory of integers?
- Is $\forall x, y \,.\, \exists z \,.\, x \cdot y = z$ in theory of integers?
- Decidable

Other Theories

- Peano Arithmetic
- Presburger Arithmetic
- Theory of Rationals
- Theory of Arrays
- …
- You can also combine theories

Today's Agenda

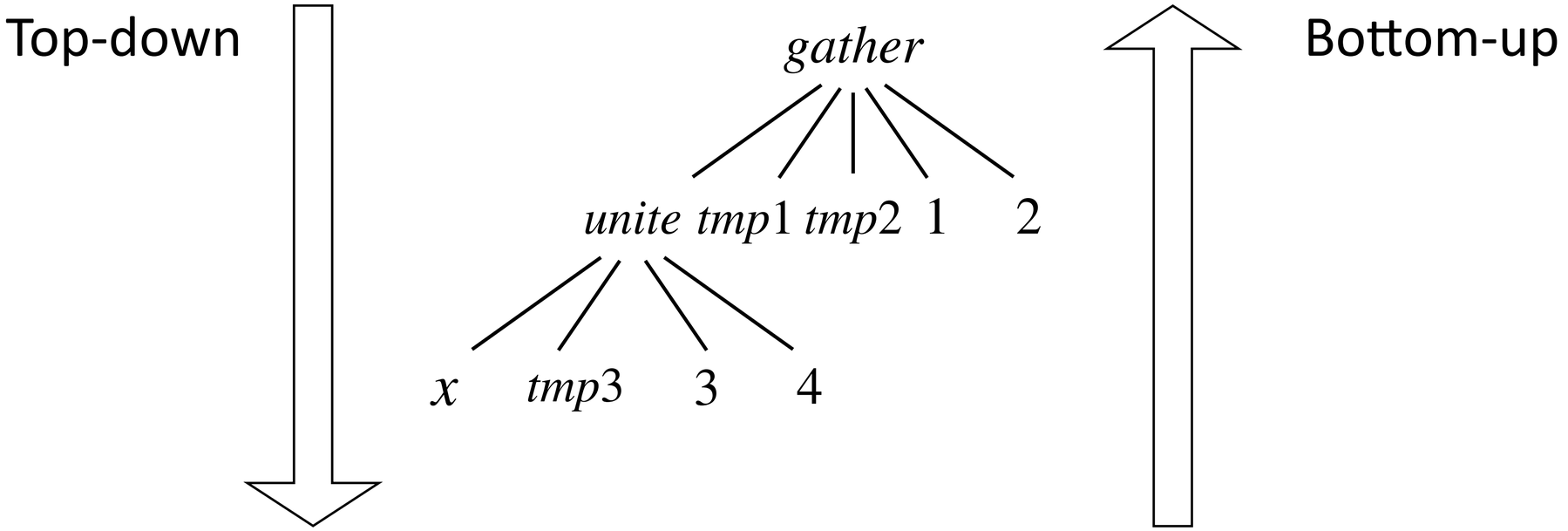
- Wrap up Logics
- **Deduction-based Pruning Techniques in Top-Down Search**
- Observational Equivalence Reduction in Bottom-Up Search

Why?

- **Naive search algorithms are slow**

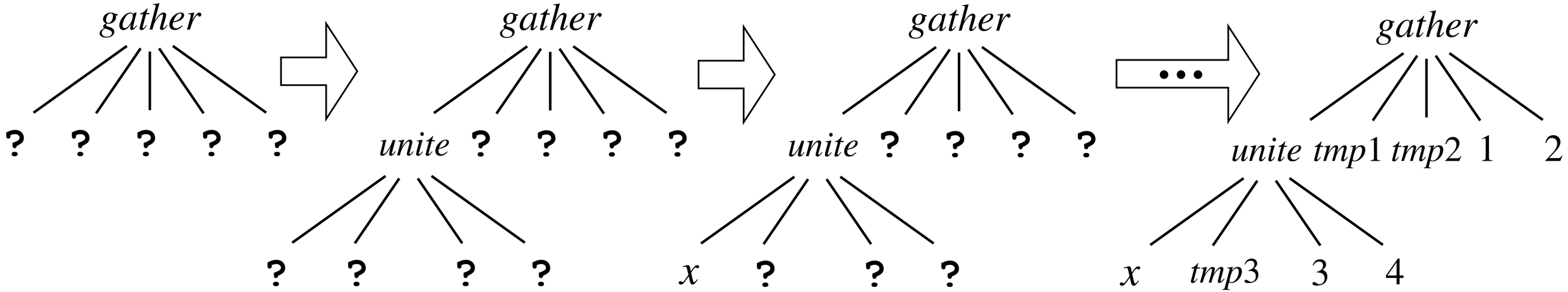
Enumeration-based Approaches

- Two ideas: Top-down and bottom-up



Top-Down Search

- Key idea: A parent node was generated before its children are generated
  - Or, generate high(er) level structures first, then fill it with low(er) level fragments



Top-Down Search Algorithm

- Given a CFG $G = (T, N, P, S)$ and a set E of examples:

  Top-Down-Search ((T, N, P, S), E):

  worklist := { S };

  while (*worklist* **is not empty**):

  AST := *worklist*.remove();

  if (AST **is complete &** AST **satisfies** E): **return** AST;

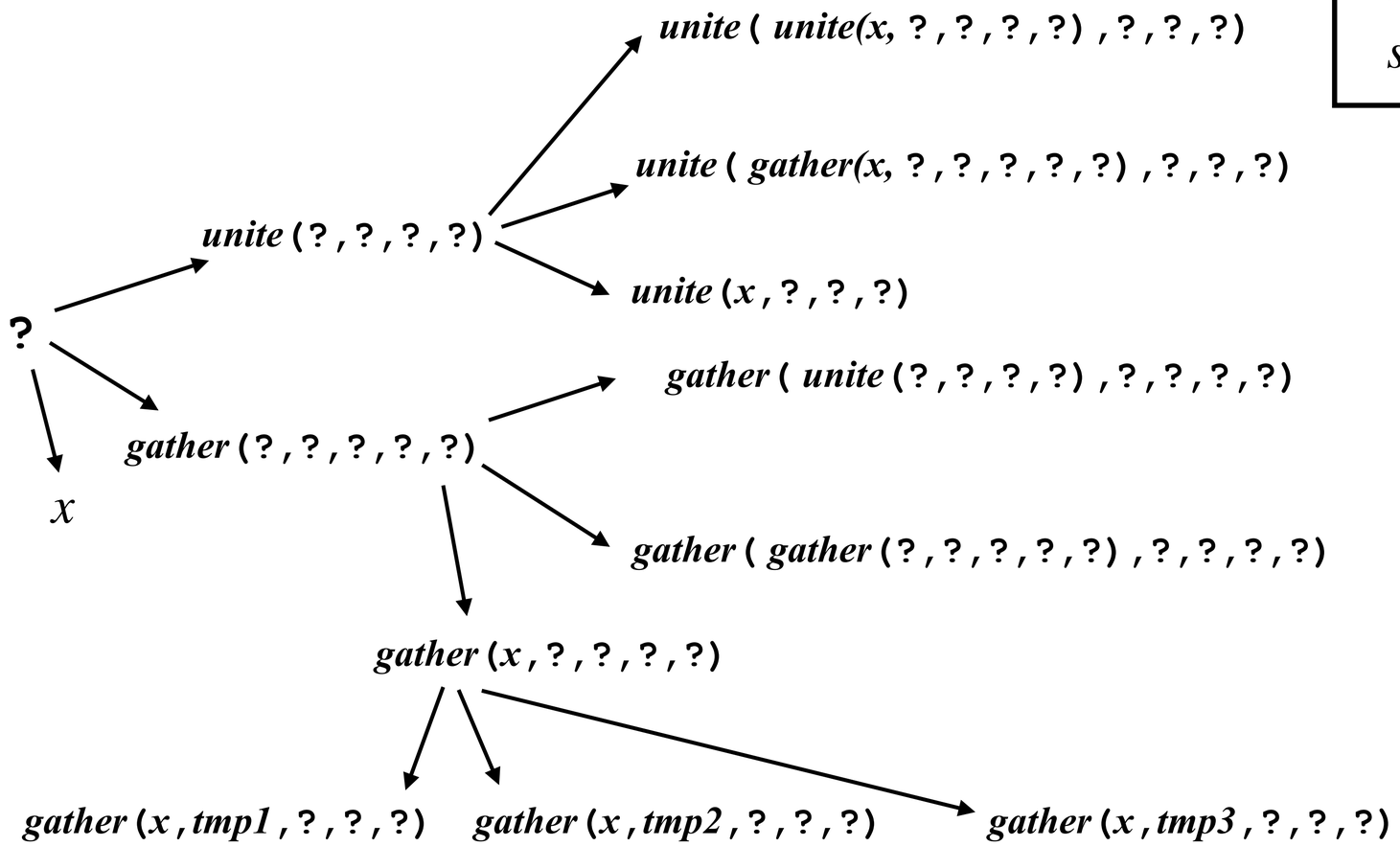
  worklist.addAll(**expand**(AST));

- High-level idea: An iterative algorithm that manipulates ASTs and creates more ASTs

Top-Down Search Algorithm

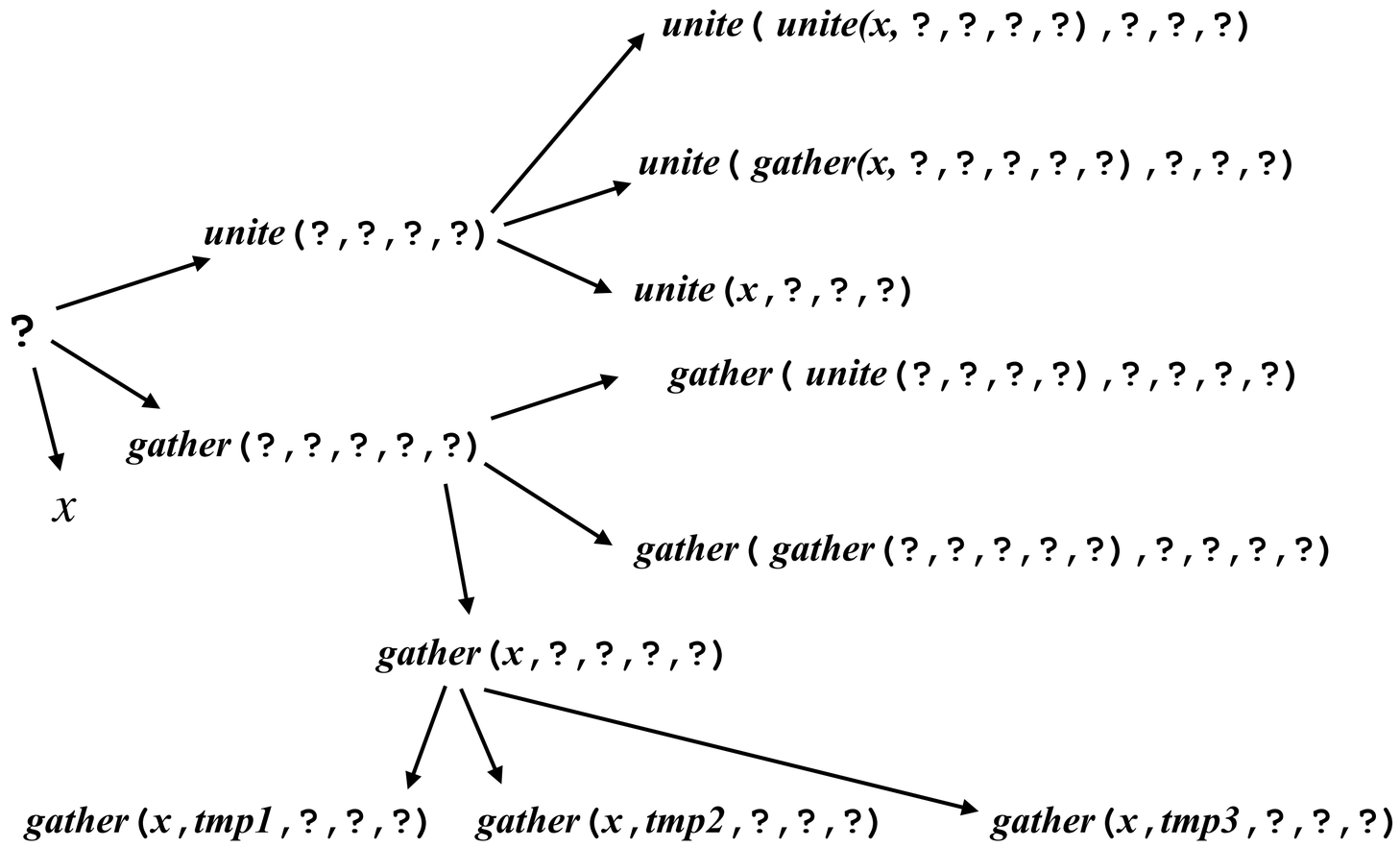
- One way to “visualize” this algorithm:

$$df ::= x \mid gather(df, s, s, k, k) \mid unite(df, s, k, k)$$
$$k ::= 1 \mid 2 \mid 3 \mid 4$$
$$s ::= tmp1 \mid tmp2 \mid tmp3$$



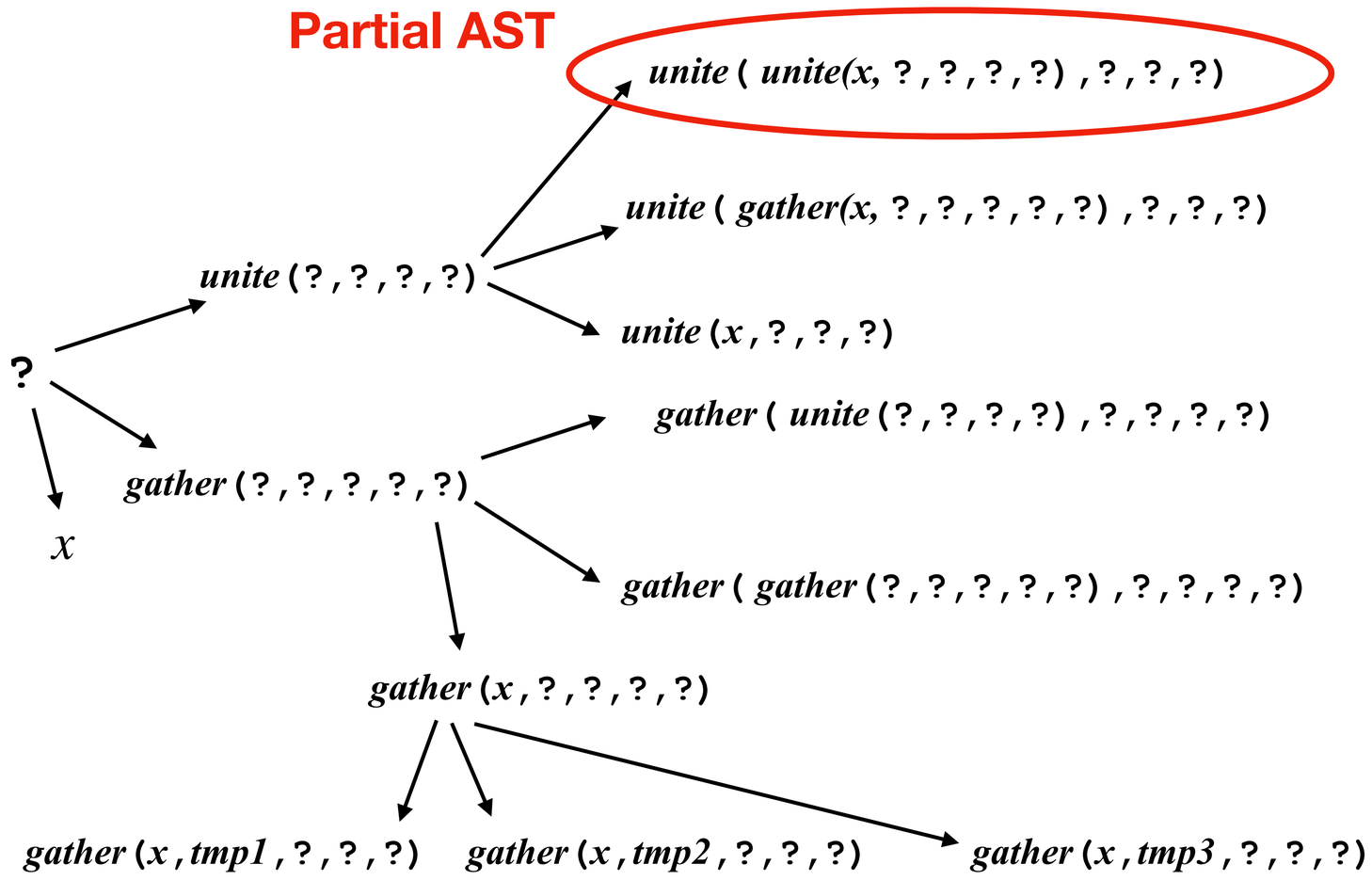
Top-Down Search Algorithm

- A key limitation: **slow!**



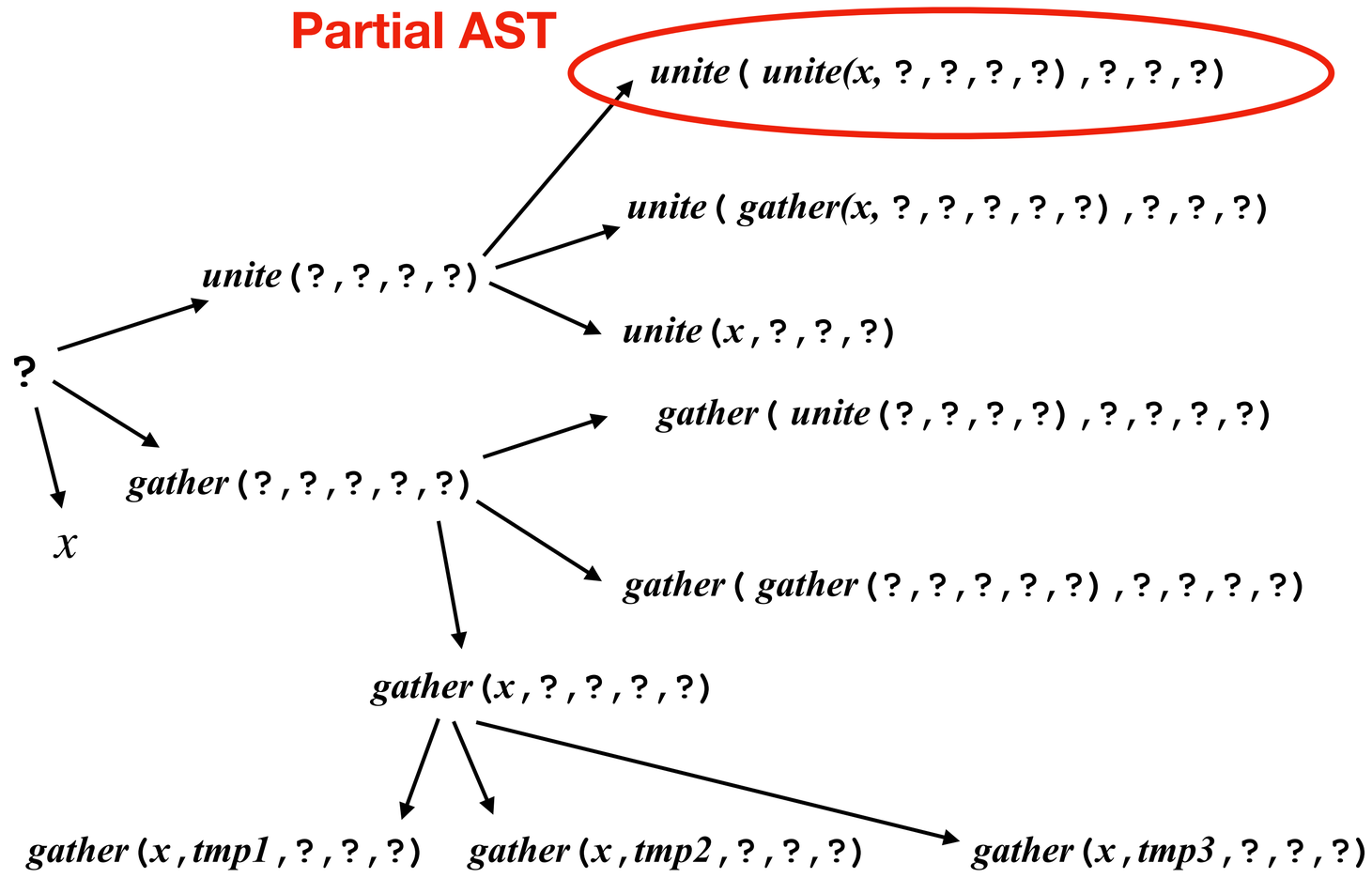
Top-Down Search Algorithm

- A key limitation: **slow!**
- Why slow?
  - Too many partial ASTs



Top-Down Search Algorithm

- A key limitation: **slow!**
- Why slow?
  - Too many partial ASTs
- Is this necessary?



Simple Example

- DSL (of arithmetic expressions)

$$e ::= x \mid c \mid e + e \mid e - e$$

$$c ::= 1 \mid 2 \mid \cdots \mid 10$$

Example: 5 —> 20

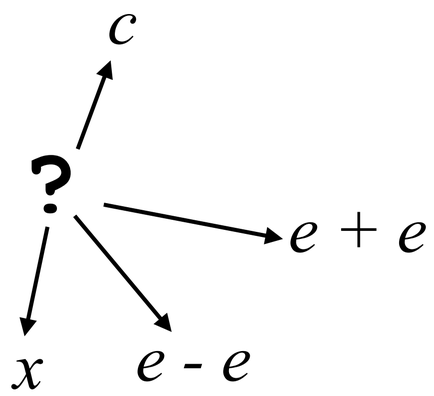
Simple Example

- DSL (of arithmetic expressions)

$$e ::= x \mid c \mid e + e \mid e - e$$

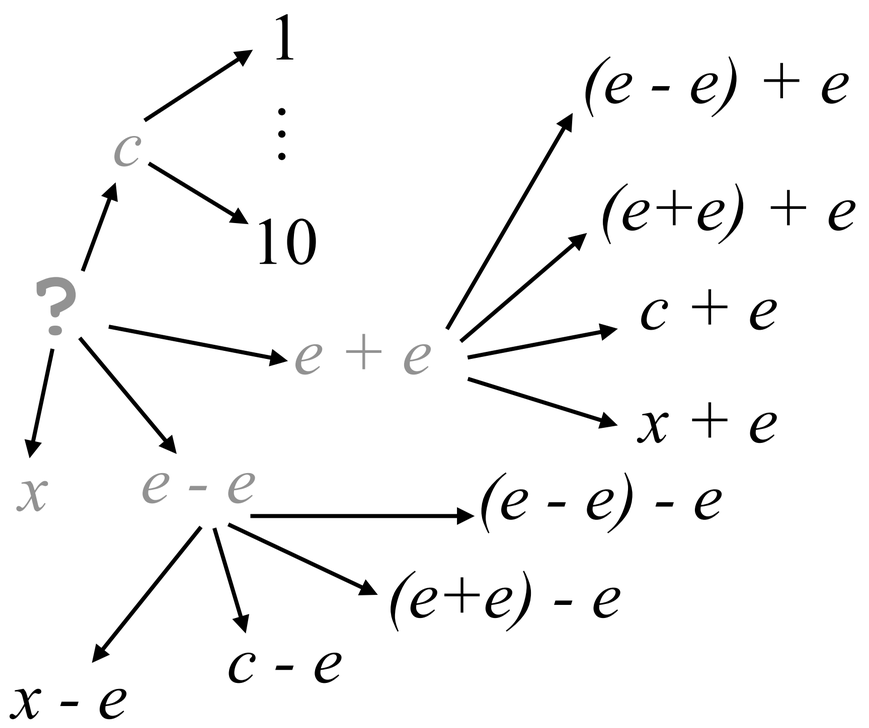
$$c ::= 1 \mid 2 \mid \cdots \mid 10$$

Example: 5 —> 20



Simple Example

- DSL (of arithmetic expressions)



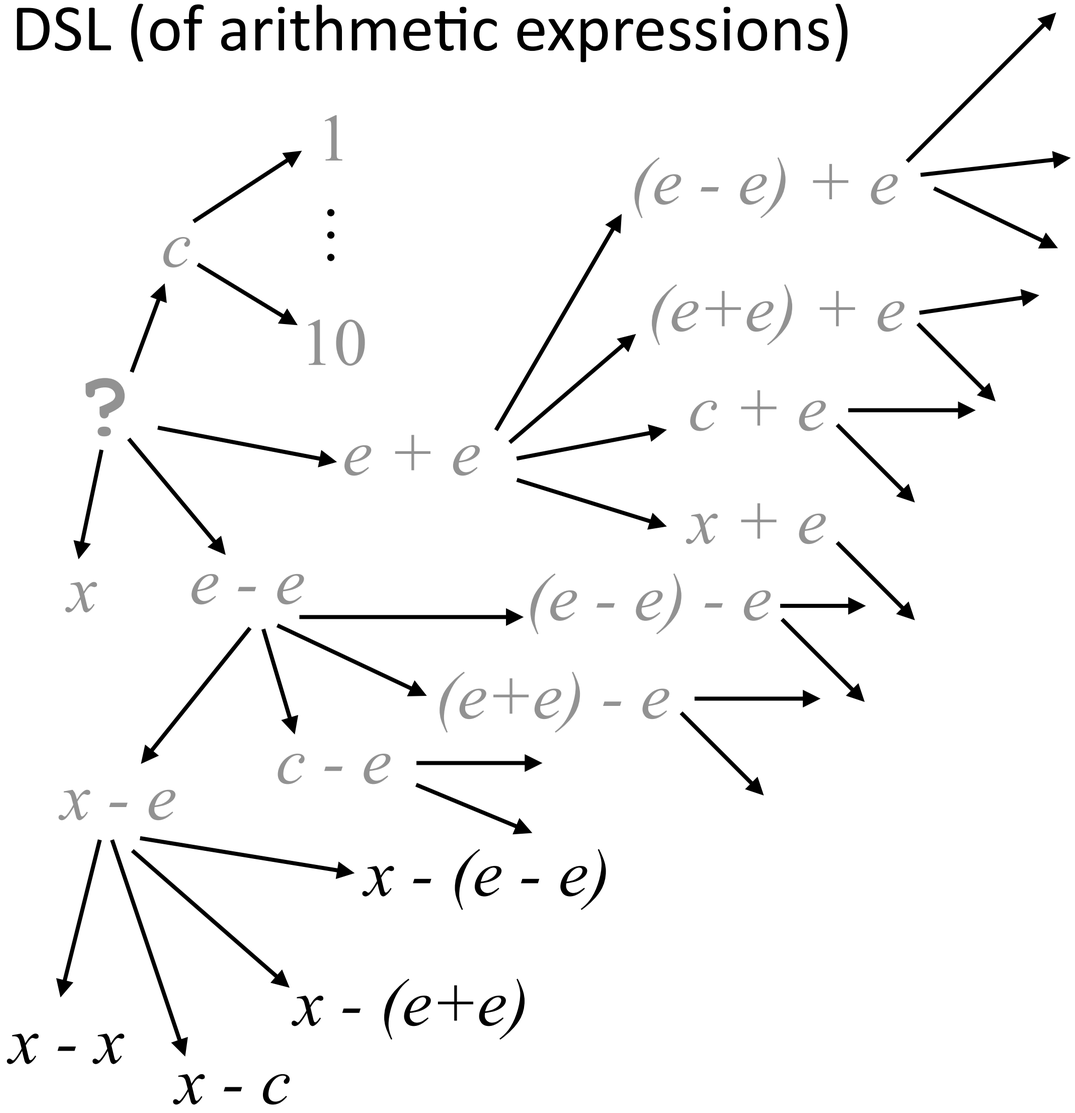
$e ::= x \mid c \mid e + e \mid e - e$

$c ::= 1 \mid 2 \mid \cdots \mid 10$

Example: 5 —> 20

Simple Example

- DSL (of arithmetic expressions)



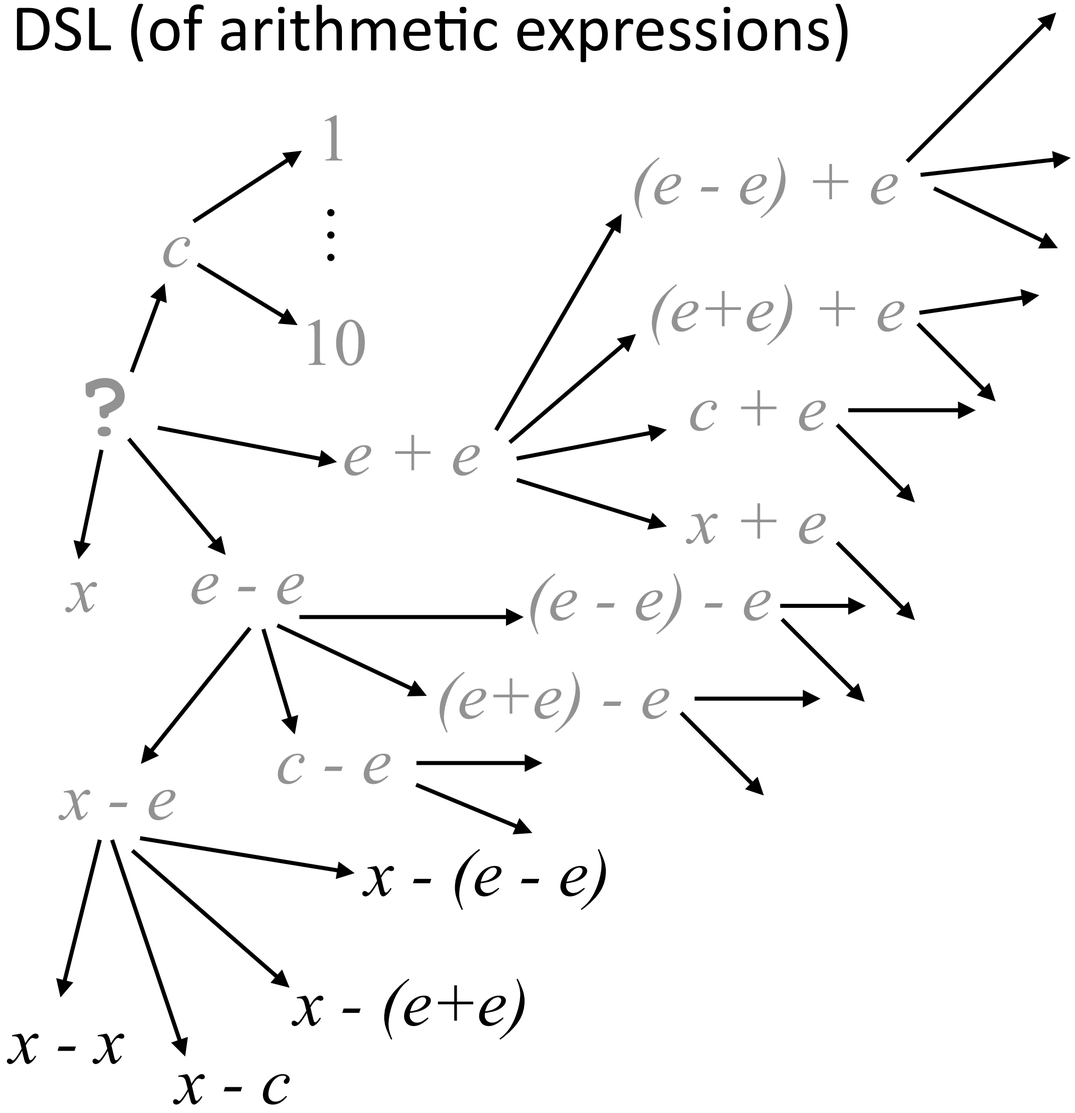
$e ::= x \mid c \mid e + e \mid e - e$

$c ::= 1 \mid 2 \mid \cdots \mid 10$

Example: 5 —> 20

Simple Example

- DSL (of arithmetic expressions)



$e ::= x \mid c \mid e + e \mid e - e$

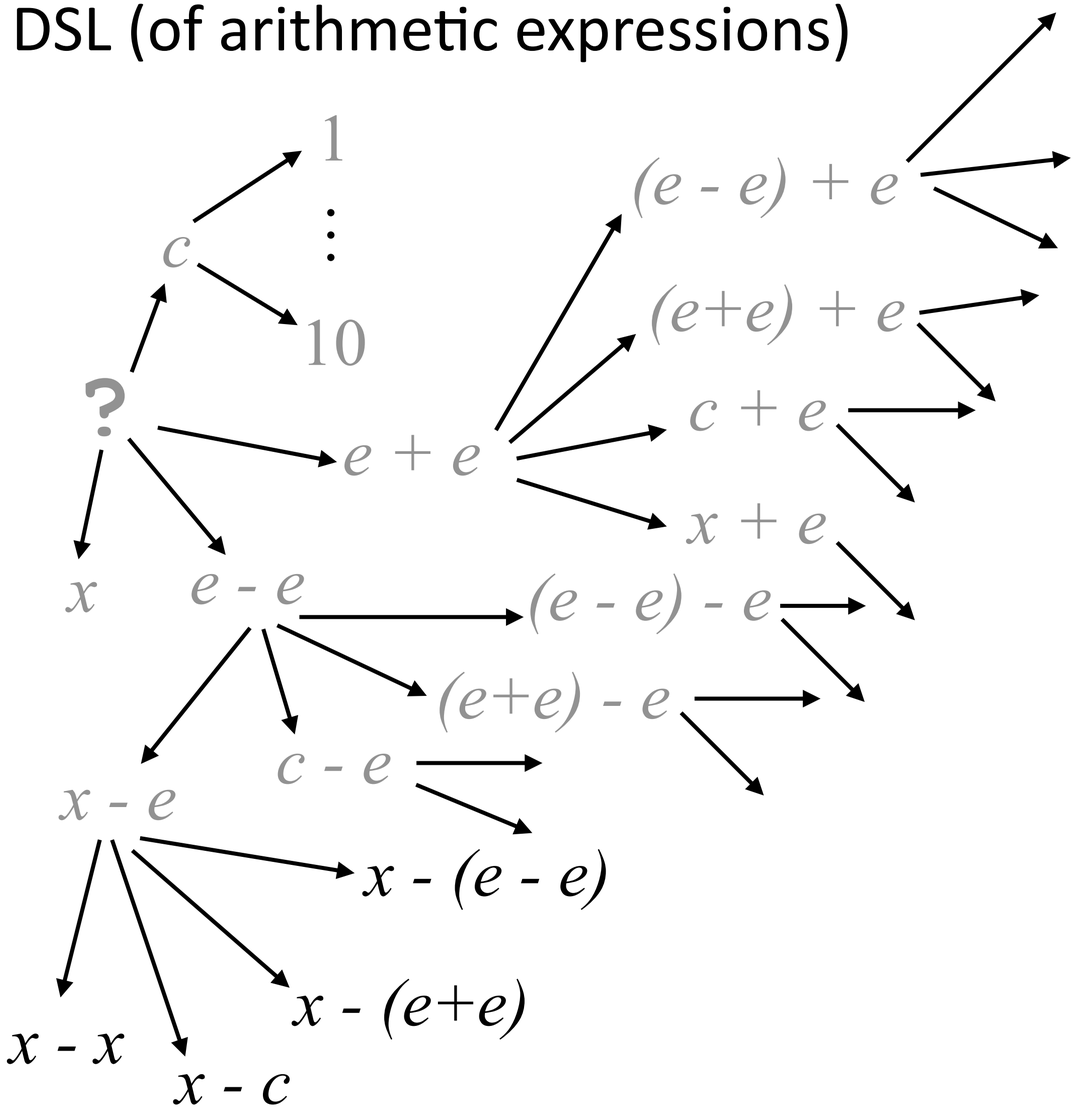
$c ::= 1 \mid 2 \mid \cdots \mid 10$

Example: 5 —> 20

It is necessary to expand all partial ASTs?

Simple Example

- DSL (of arithmetic expressions)



$e ::= x \mid c \mid e + e \mid e - e$

$c ::= 1 \mid 2 \mid \cdots \mid 10$

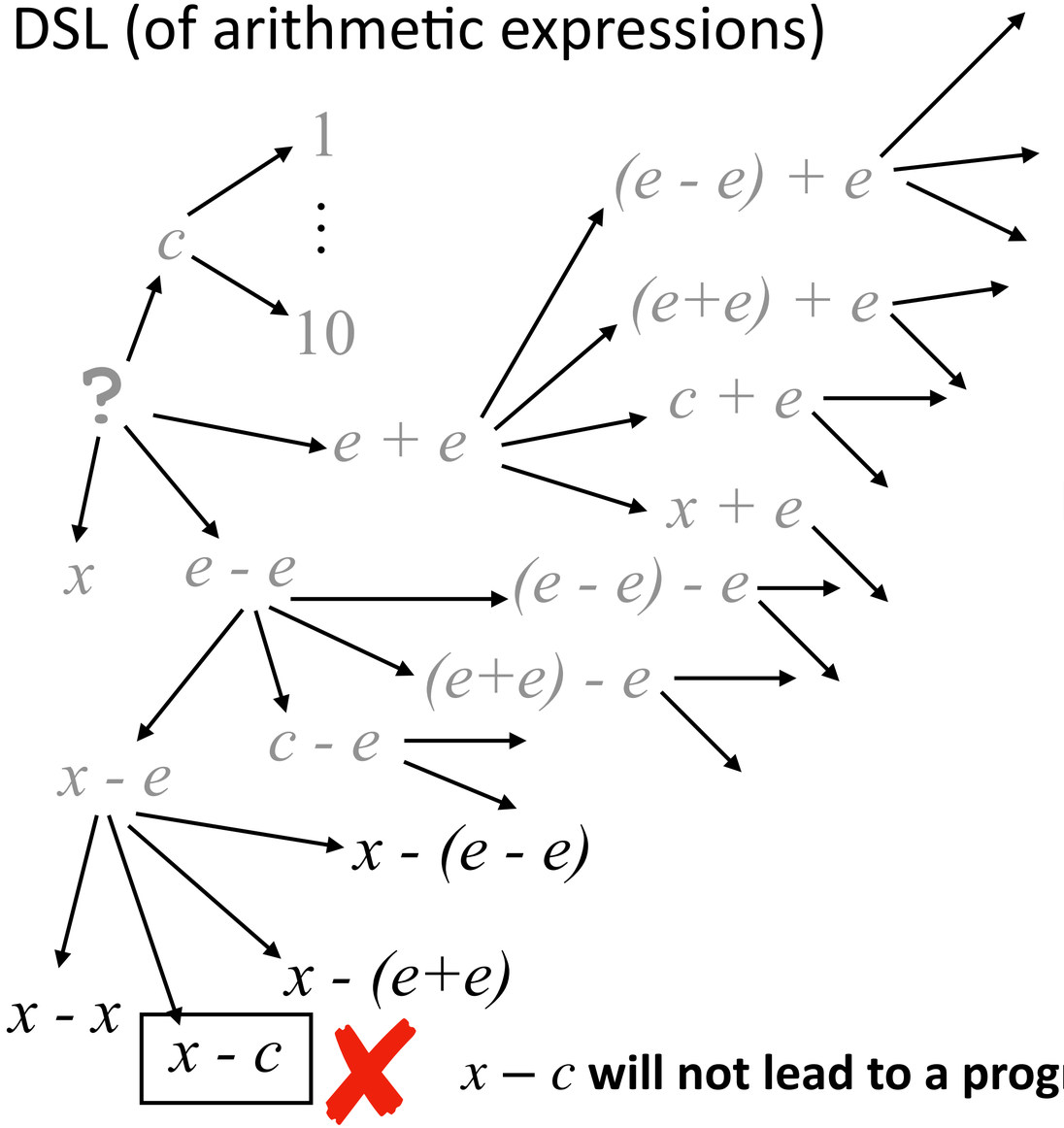
Example: 5 —> 20

It is necessary to expand all partial ASTs?

No!

Simple Example

- DSL (of arithmetic expressions)



$e ::= x \mid c \mid e + e \mid e - e$

$c ::= 1 \mid 2 \mid \cdots \mid 10$

Example: 5 —> 20

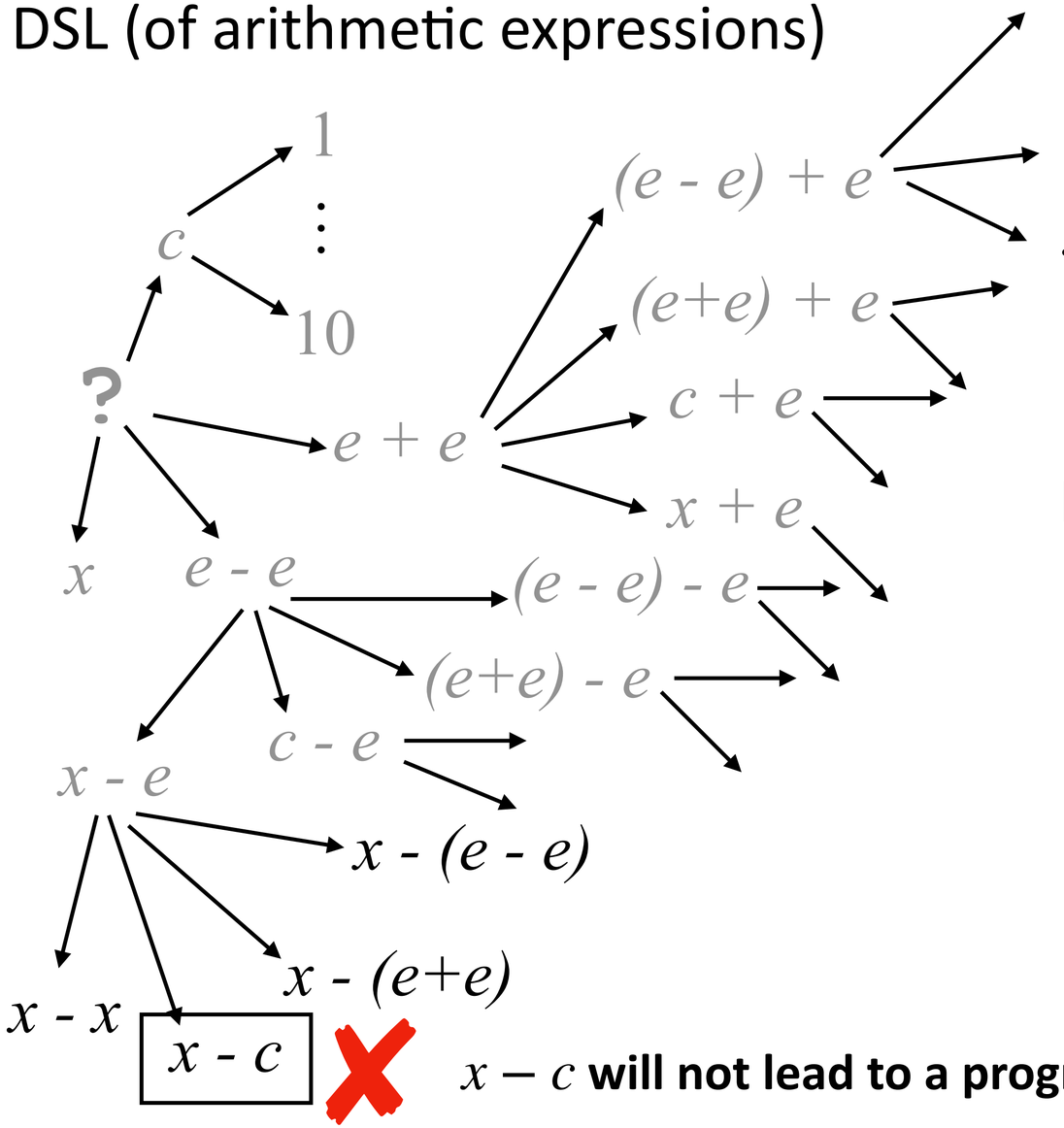
It is necessary to expand all partial ASTs?

No!

$x - c$ **will not lead to a program that satisfies the example**

Simple Example

- DSL (of arithmetic expressions)



$e ::= x \mid c \mid e + e \mid e - e$

$c ::= 1 \mid 2 \mid \cdots \mid 10$

Example: 5 —> 20

… → $(x - c) + c$ ✗

It is necessary to expand all partial ASTs?

No!

$x - c$ **will not lead to a program that satisfies the example**

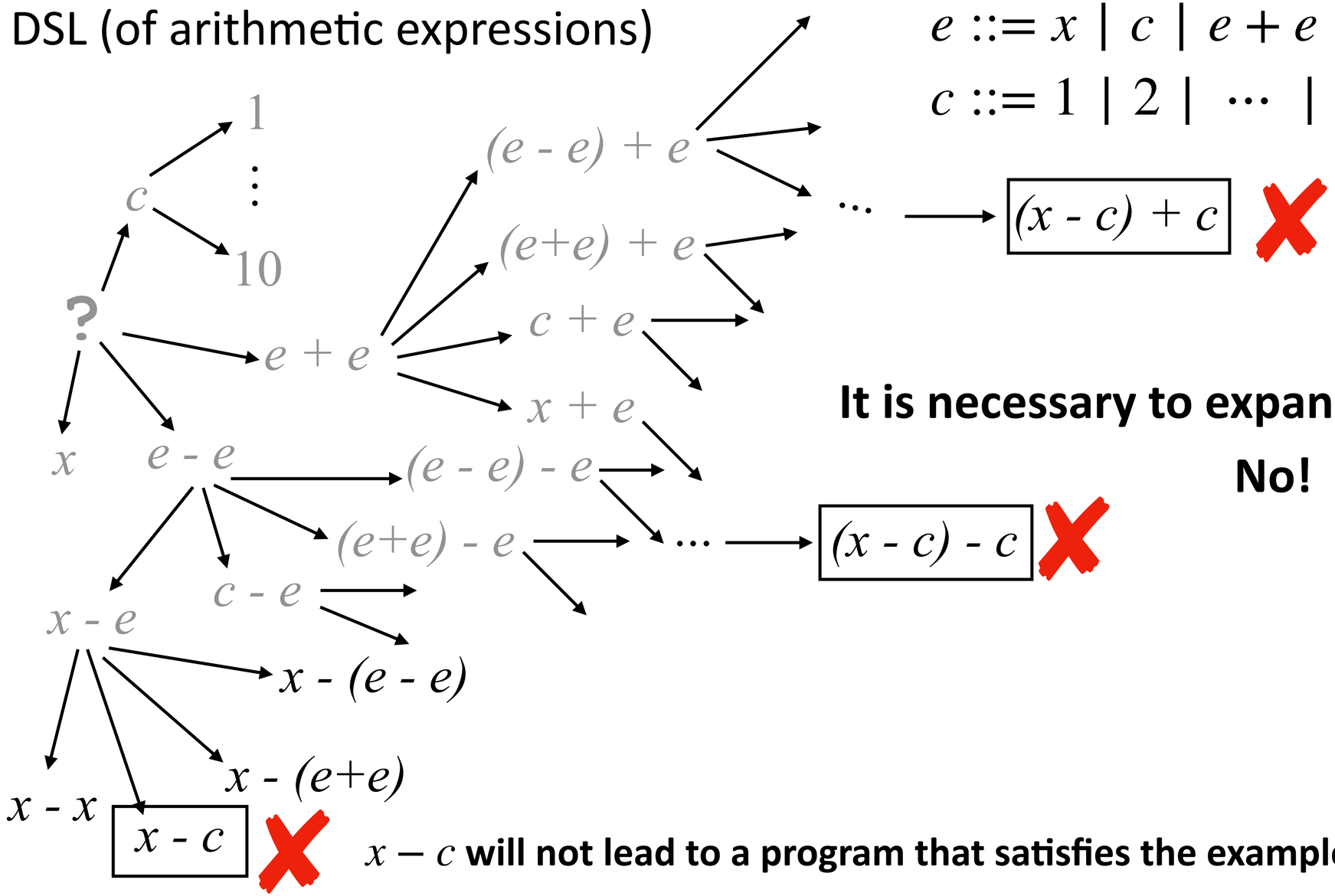
Simple Example

- DSL (of arithmetic expressions)

$$e ::= x \mid c \mid e + e \mid e - e$$

$$c ::= 1 \mid 2 \mid \cdots \mid 10$$

Example: 5 —> 20



It is necessary to expand all partial ASTs?

No!

$x - c$ **will not lead to a program that satisfies the example**

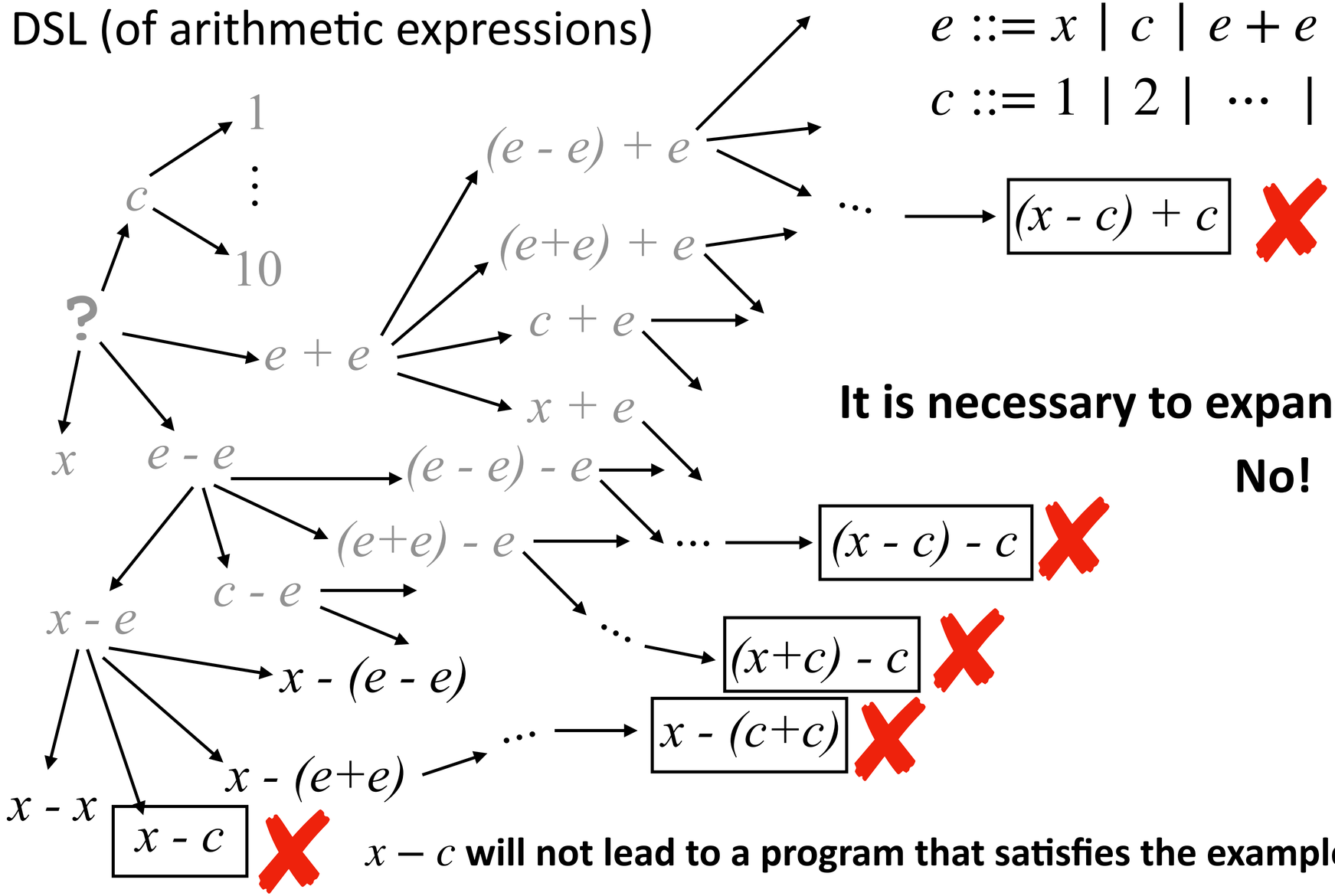
Simple Example

- DSL (of arithmetic expressions)

$$e ::= x \mid c \mid e + e \mid e - e$$

$$c ::= 1 \mid 2 \mid \cdots \mid 10$$

Example: 5 —> 20



It is necessary to expand all partial ASTs?

No!

$x - c$ **will not lead to a program that satisfies the example**

Prune Partial Programs

- Key idea: If we know a partial program won’t lead to a correct program, prune it

Prune Partial Programs

- Key idea: If we know a partial program won't lead to a correct program, prune it
- Key question: How to perform reasoning **automatically**?
  - Perform deduction using SMT solvers

Automatically Reason about Partial Programs

- Partial programs are **abstract**
  - We cannot run a partial program, because it's not complete
  - We need to characterize "abstract behavior" of partial programs

Automatically Reason about Partial Programs

- Partial programs are **abstract**
  - We cannot run a partial program, because it's not complete
  - We need to characterize "abstract behavior" of partial programs
- We use the following idea to "run" partial programs:
  - First, define **"abstract semantics"** of DSL operators which describe **"abstract behavior"** of operators

Automatically Reason about Partial Programs

- Partial programs are **abstract**
  - We cannot run a partial program, because it's not complete
  - We need to characterize "abstract behavior" of partial programs
- We use the following idea to "run" partial programs:
  - First, define **"abstract semantics"** of DSL operators which describe **"abstract behavior"** of operators
  - Then, describe "abstract behavior" of a partial program by composing "abstract behaviors" of its components

Automatically Reason about Partial Programs

- Partial programs are **abstract**
  - We cannot run a partial program, because it's not complete
  - We need to characterize "abstract behavior" of partial programs
- We use the following idea to "run" partial programs:
  - First, define **"abstract semantics"** of DSL operators which describe **"abstract behavior"** of operators
  - Then, describe "abstract behavior" of a partial program by composing "abstract behaviors" of its components
  - Finally, check against the example

Automatically Reason about Partial Programs

- First, define **“abstract behavior”** of each DSL operator

Automatically Reason about Partial Programs

- First, define **“abstract behavior”** of each DSL operator

Example: 5 —> 20

$$e ::= x \mid c \mid e + e \mid e - e \qquad c ::= 1 \mid 2 \mid \cdots \mid 10$$

Automatically Reason about Partial Programs

- First, define **“abstract behavior”** of each DSL operator

Example: 5 —> 20

$$e ::= x \mid c \mid e + e \mid e - e \qquad c ::= 1 \mid 2 \mid \cdots \mid 10$$

$$y = x_1$$

Automatically Reason about Partial Programs

- First, define **“abstract behavior”** of each DSL operator

Example: 5 —> 20

$$e ::= x \mid c \mid e + e \mid e - e \qquad c ::= 1 \mid 2 \mid \cdots \mid 10$$

$y = x_1$ ←

↘

$$1 \leq y \leq 10$$

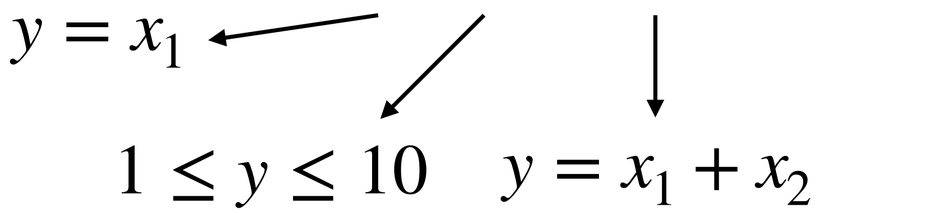
Abstract

Automatically Reason about Partial Programs

- First, define **“abstract behavior”** of each DSL operator

Example: 5 —> 20

$$e ::= x \mid c \mid e + e \mid e - e \qquad c ::= 1 \mid 2 \mid \cdots \mid 10$$

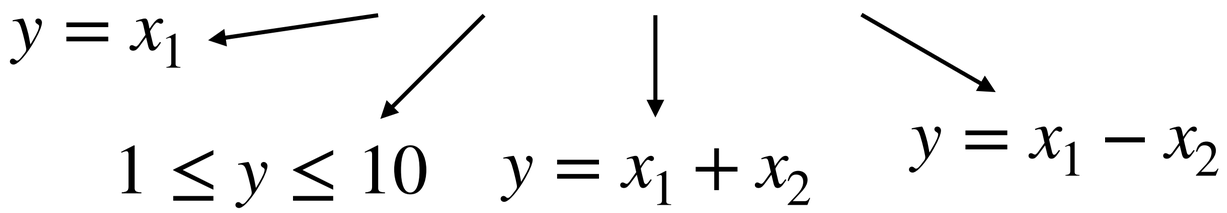


Automatically Reason about Partial Programs

- First, define **“abstract behavior”** of each DSL operator

Example: 5 —> 20

$$e ::= x \mid c \mid e + e \mid e - e \qquad c ::= 1 \mid 2 \mid \cdots \mid 10$$

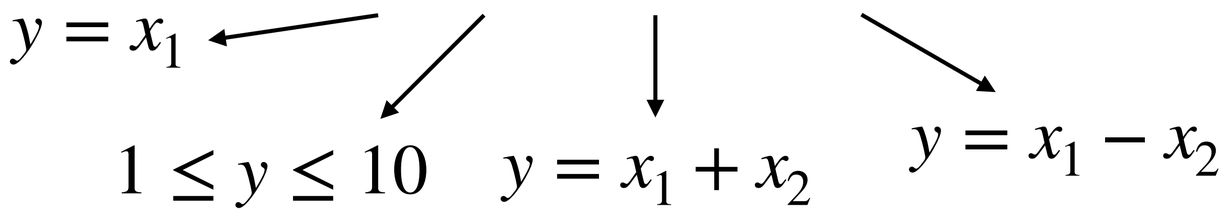


Automatically Reason about Partial Programs

- First, define **“abstract behavior”** of each DSL operator

Example: 5 —> 20

$$e ::= x \mid c \mid e + e \mid e - e \qquad c ::= 1 \mid 2 \mid \cdots \mid 10$$



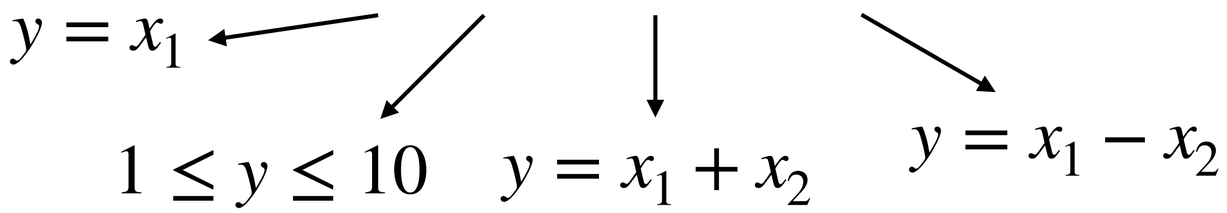
- Second, “abstract behavior” of any partial program is composed using “abstract behaviors” of its components

Automatically Reason about Partial Programs

- First, define **“abstract behavior”** of each DSL operator

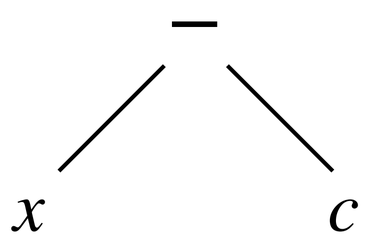
Example: 5 —> 20

$$e ::= x \mid c \mid e + e \mid e - e \qquad c ::= 1 \mid 2 \mid \cdots \mid 10$$



- Second, “abstract behavior” of any partial program is composed using “abstract behaviors” of its components

$x - c$

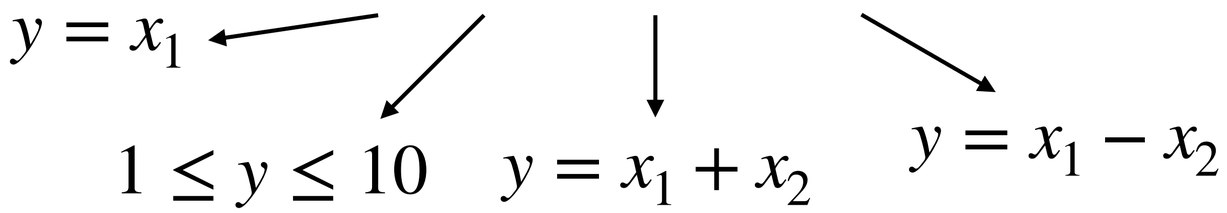


Automatically Reason about Partial Programs

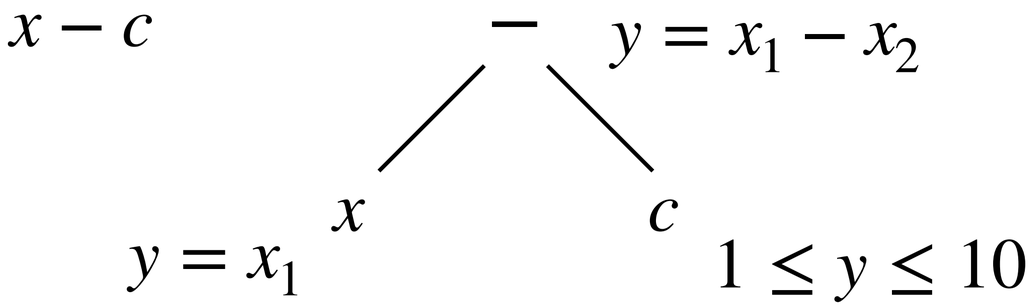
- First, define **"abstract behavior"** of each DSL operator

Example: 5 —> 20

$$e ::= x \mid c \mid e + e \mid e - e \qquad c ::= 1 \mid 2 \mid \cdots \mid 10$$



- Second, "abstract behavior" of any partial program is composed using "abstract behaviors" of its components

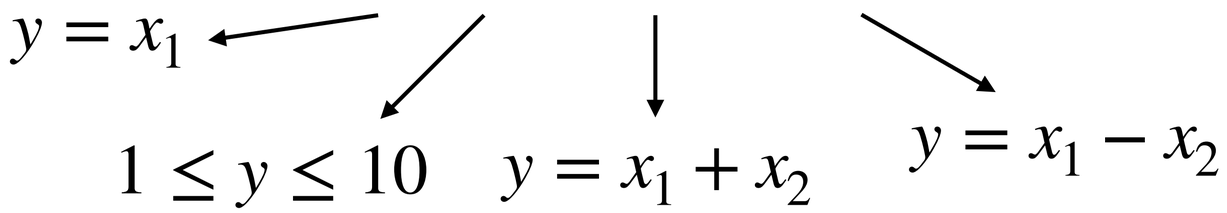


Automatically Reason about Partial Programs

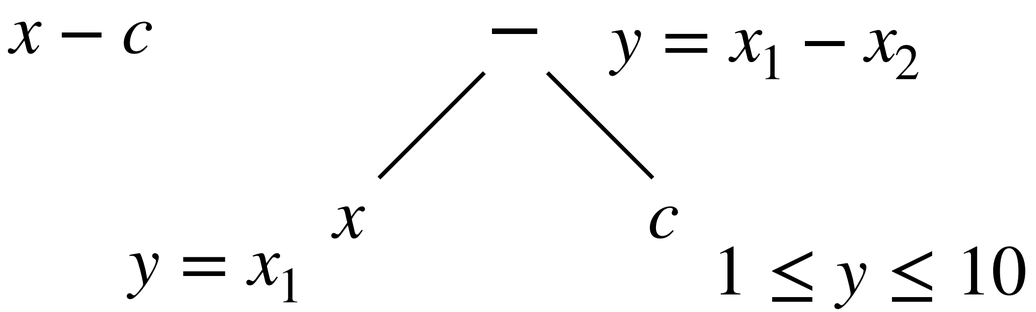
- First, define **"abstract behavior"** of each DSL operator

Example: 5 —> 20

$$e ::= x \mid c \mid e + e \mid e - e \qquad c ::= 1 \mid 2 \mid \cdots \mid 10$$



- Second, "abstract behavior" of any partial program is composed using "abstract behaviors" of its components

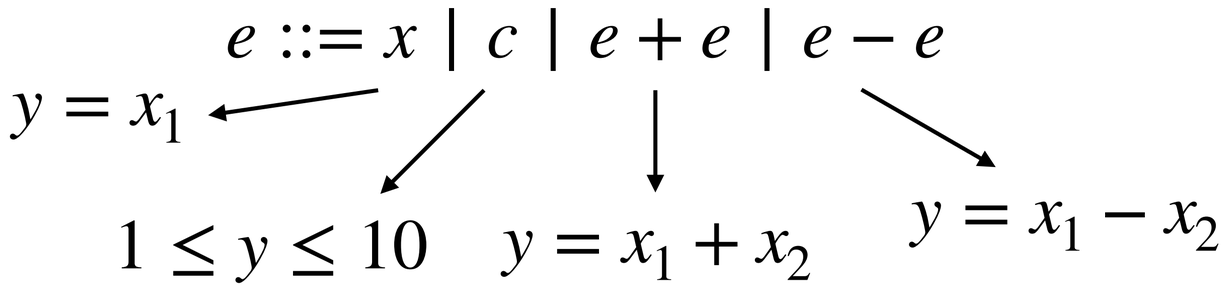


How to **compose**?

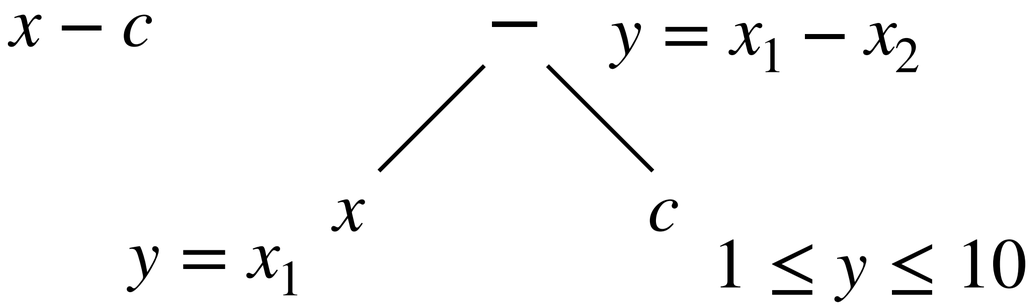
Automatically Reason about Partial Programs

- First, define **"abstract behavior"** of each DSL operator

Example: 5 —> 20



- Second, "abstract behavior" of any partial program is composed using "abstract behaviors" of its components



How to **compose**?

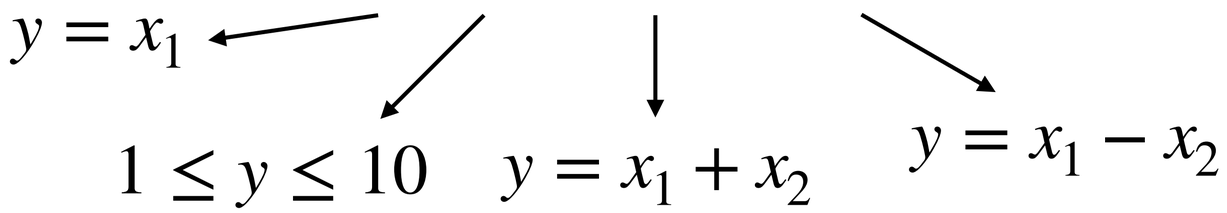
Idea: conjoin all formulas

Automatically Reason about Partial Programs

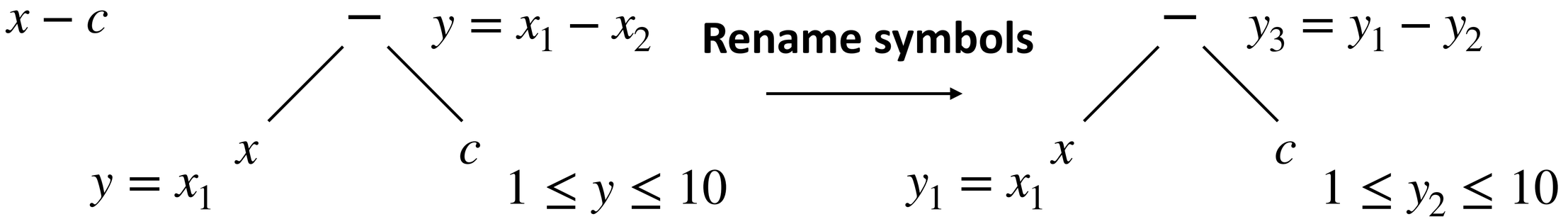
- First, define **"abstract behavior"** of each DSL operator

Example: 5 —> 20

$$e ::= x \mid c \mid e + e \mid e - e \qquad c ::= 1 \mid 2 \mid \cdots \mid 10$$



- Second, "abstract behavior" of any partial program is composed using "abstract behaviors" of its components

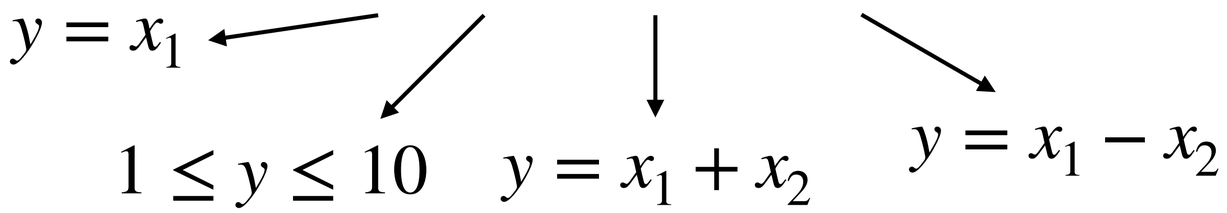


Automatically Reason about Partial Programs

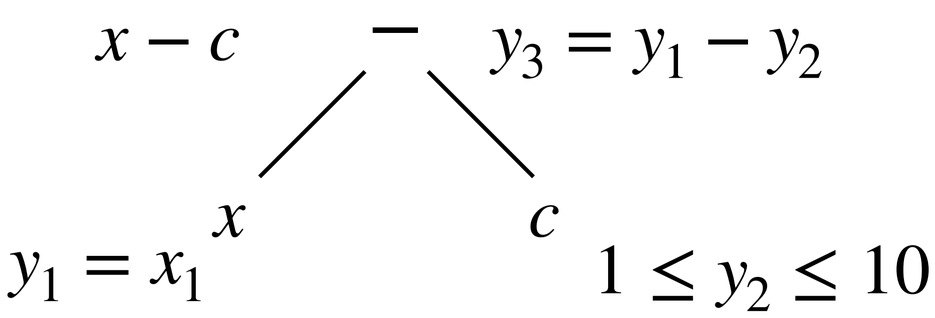
- First, define **"abstract behavior"** of each DSL operator

Example: 5 —> 20

$$e ::= x \mid c \mid e + e \mid e - e \qquad c ::= 1 \mid 2 \mid \cdots \mid 10$$



- Second, "abstract behavior" of any partial program is composed using "abstract behaviors" of its components



Abstract behavior of the entire program:

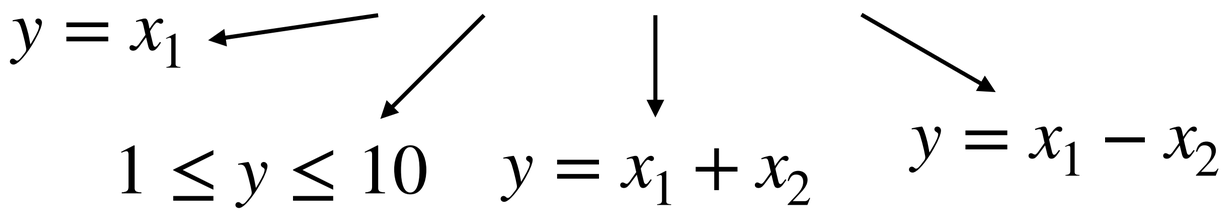
$$(y_1 = x_1) \wedge (1 \le y_2 \le 10) \wedge (y_3 = y_1 - y_2)$$

Automatically Reason about Partial Programs

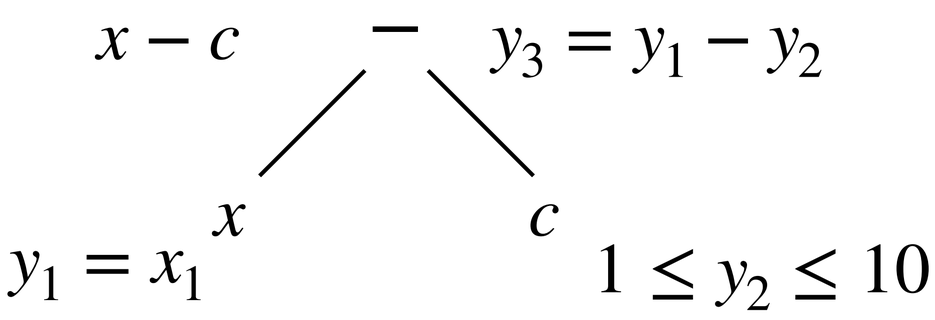
- First, define **"abstract behavior"** of each DSL operator

Example: 5 —> 20

$$e ::= x \mid c \mid e + e \mid e - e \qquad c ::= 1 \mid 2 \mid \cdots \mid 10$$



- Second, "abstract behavior" of any partial program is composed using "abstract behaviors" of its components



Abstract behavior of the entire program:

$$(y_1 = x_1) \wedge (1 \le y_2 \le 10) \wedge (y_3 = y_1 - y_2)$$

Or:

$$(y_1 = x_1) \wedge (1 \le y_2 \le 10) \wedge (y_3 = y_1 - y_2) \wedge (x_1 = x) \wedge (y = y_3)$$

which simplifies to:

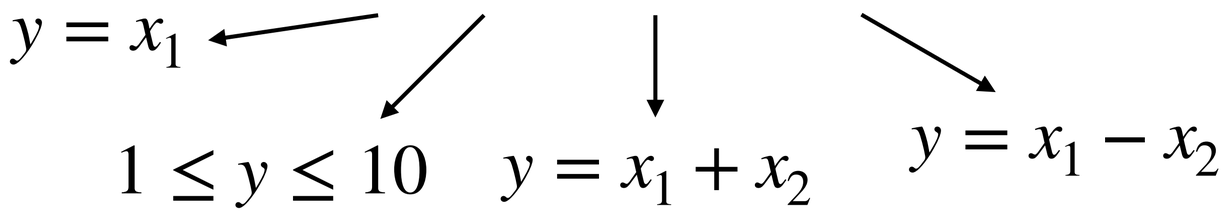
$$(y_1 = x) \wedge (1 \le y_2 \le 10) \wedge (y = y_1 - y_2)$$

Automatically Reason about Partial Programs

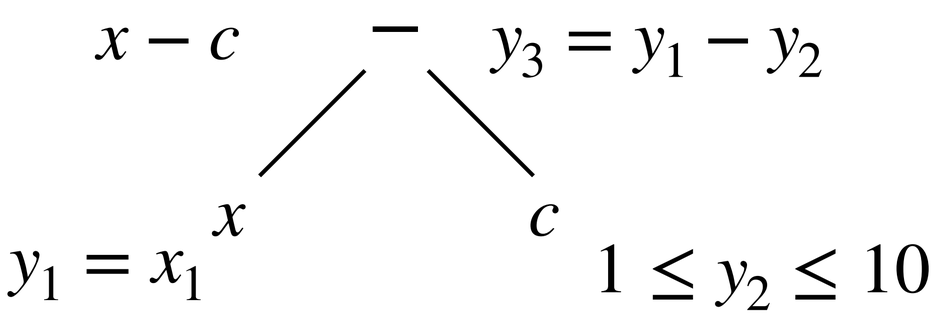
- First, define **"abstract behavior"** of each DSL operator

Example: 5 —> 20

$$e ::= x \mid c \mid e + e \mid e - e \qquad c ::= 1 \mid 2 \mid \cdots \mid 10$$



- Second, "abstract behavior" of any partial program is composed using "abstract behaviors" of its components



Abstract behavior of the entire program:

$$(y_1 = x) \wedge (1 \le y_2 \le 10) \wedge (y = y_1 - y_2)$$

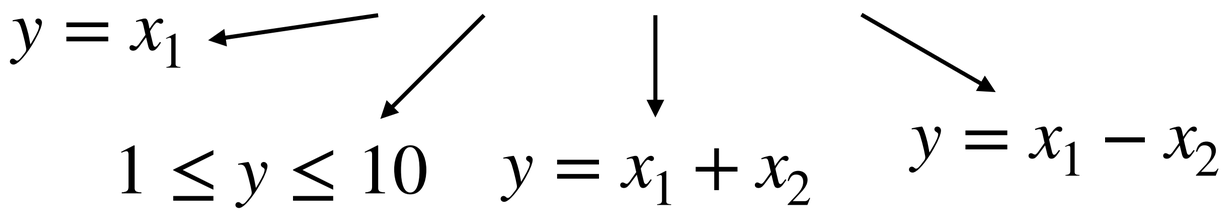
- Finally, check against example

Automatically Reason about Partial Programs

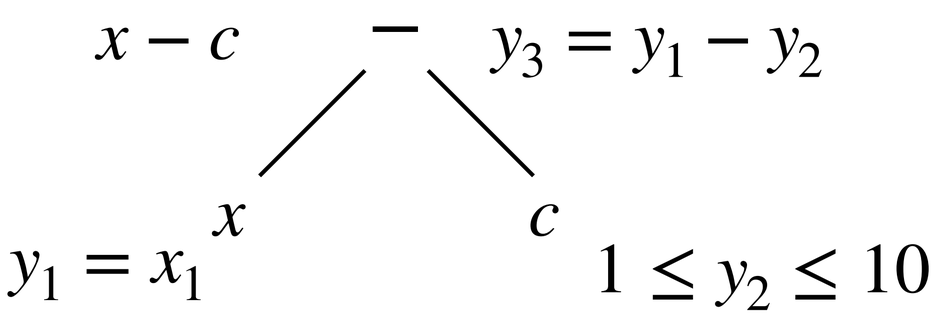
- First, define **"abstract behavior"** of each DSL operator

Example: 5 —> 20

$$e ::= x \mid c \mid e + e \mid e - e \qquad c ::= 1 \mid 2 \mid \cdots \mid 10$$



- Second, "abstract behavior" of any partial program is composed using "abstract behaviors" of its components



Abstract behavior of the entire program:

$$(y_1 = x) \wedge (1 \le y_2 \le 10) \wedge (y = y_1 - y_2)$$

- Finally, check against example

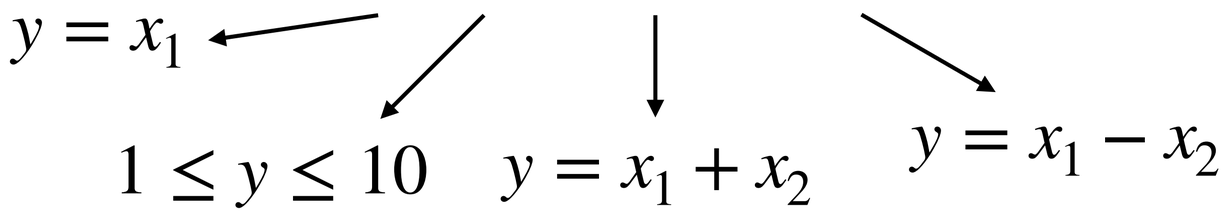
$$\boxed{(x = 5) \wedge (y = 20)} \wedge (y_1 = x) \wedge (1 \le y_2 \le 10) \wedge (y = y_1 - y_2)$$

Automatically Reason about Partial Programs

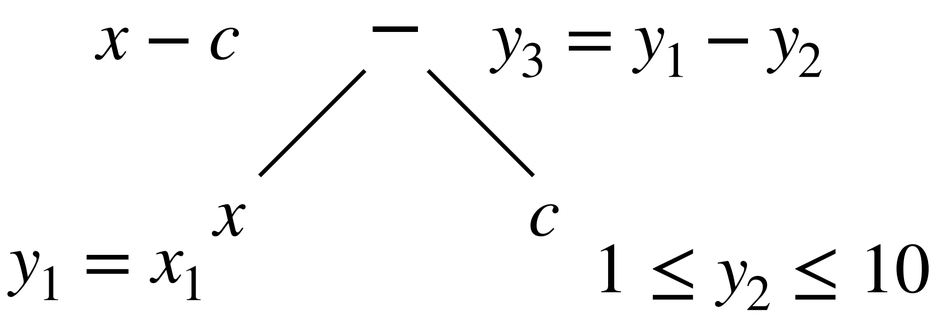
- First, define **"abstract behavior"** of each DSL operator

Example: 5 —> 20

$$e ::= x \mid c \mid e + e \mid e - e \qquad c ::= 1 \mid 2 \mid \cdots \mid 10$$



- Second, "abstract behavior" of any partial program is composed using "abstract behaviors" of its components



Abstract behavior of the entire program:

$$(y_1 = x) \wedge (1 \le y_2 \le 10) \wedge (y = y_1 - y_2)$$

- Finally, check against example

UNSAT

$$(x = 5) \wedge (y = 20) \wedge (y_1 = x) \wedge (1 \le y_2 \le 10) \wedge (y = y_1 - y_2)$$

Top-Down Search Algorithm with Deduction-based Pruning

Top-Down-Search ($(T, N, P, S), E$):

worklist := { S };

while (*worklist* **is not empty**):

AST := *worklist*.remove();

if (AST **is complete &** AST **satisfies** E): **return** AST;

if (**prune**(AST)): **continue**; // **Pruning**

worklist.addAll(**expand**(AST));

Today's Agenda

- Wrap up Logics
- Deduction-based Pruning Techniques in Top-Down Search
- **Observational Equivalence Reduction in Bottom-Up Search**

Bottom-Up Search Algorithm

- DSL (of arithmetic expressions)

$$e ::= x \mid c \mid e + e \mid e - e$$
$$c ::= 1 \mid 2 \mid \cdots \mid 10$$

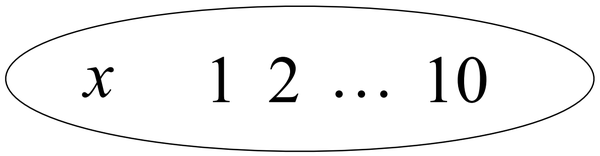
Example: *10 —> 50*

Bottom-Up Search Algorithm

- DSL (of arithmetic expressions)

$$e ::= x \mid c \mid e + e \mid e - e$$
$$c ::= 1 \mid 2 \mid \cdots \mid 10$$

Example: *10 —> 50*

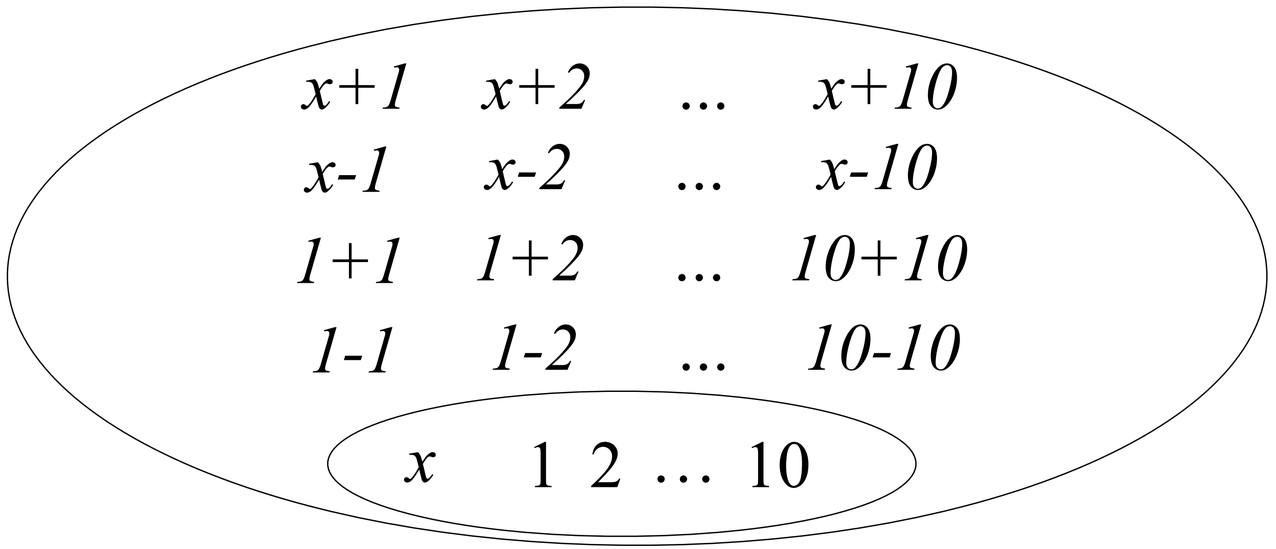


Bottom-Up Search Algorithm

- DSL (of arithmetic expressions)

$$e ::= x \mid c \mid e + e \mid e - e$$
$$c ::= 1 \mid 2 \mid \cdots \mid 10$$

Example: *10 —> 50*

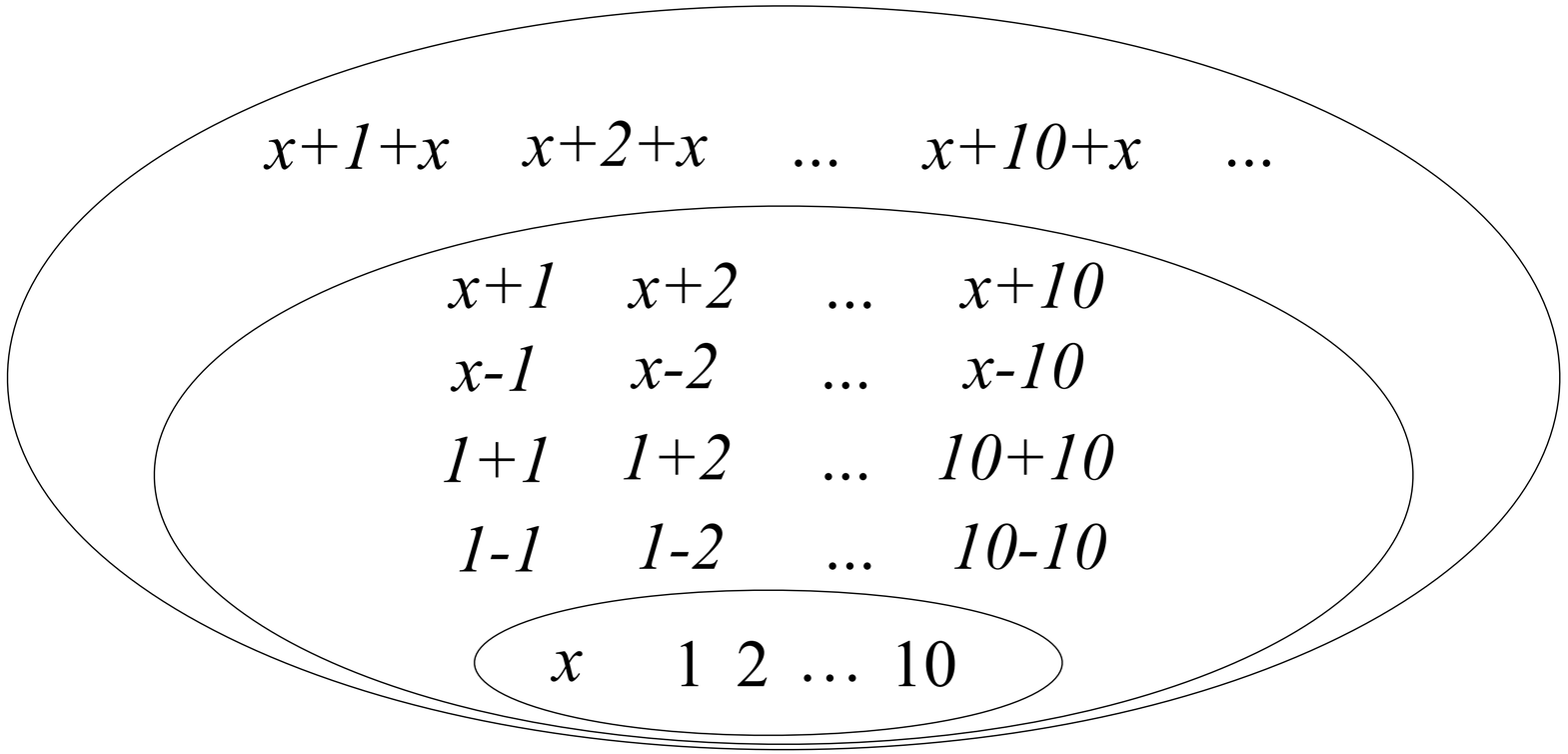


Bottom-Up Search Algorithm

- DSL (of arithmetic expressions)

$$e ::= x \mid c \mid e + e \mid e - e$$
$$c ::= 1 \mid 2 \mid \cdots \mid 10$$

Example: *10 —> 50*



Observationally Equivalent Programs

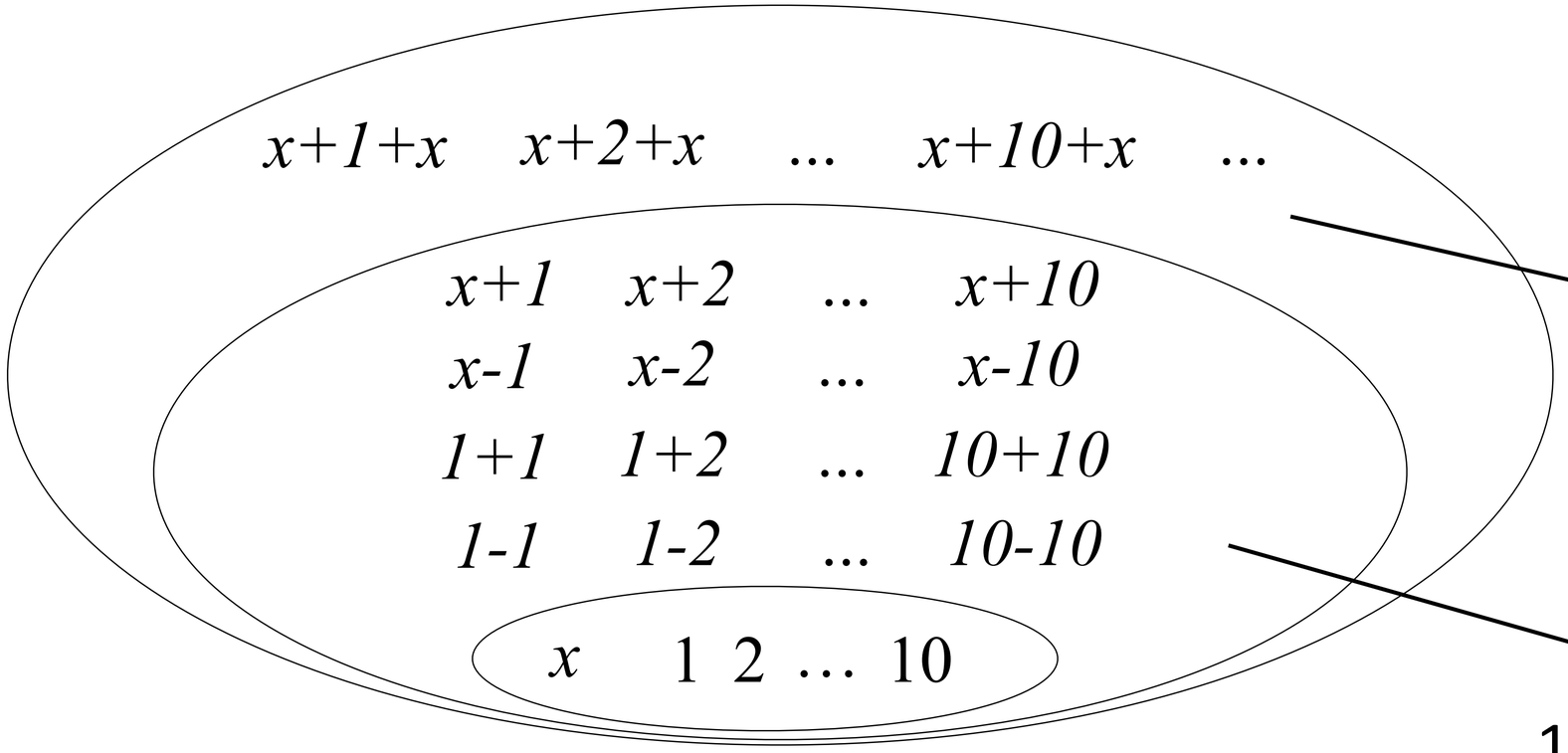
- Key idea: Many programs, but (potentially much) fewer values produced by them

Observationally Equivalent Programs

- Key idea: Many programs, but (potentially much) fewer values produced by them

$$e ::= x \mid c \mid e + e \mid e - e$$
$$c ::= 1 \mid 2 \mid \cdots \mid 10$$

Example: *10 —> 50*



11^3 programs, 50 different values

11^2 programs, 30 different values

Observational Equivalence Reduction (OER)

- Key idea: Many programs, but (potentially much) fewer values produced by them
- If we aim to satisfy examples, no need to keep track of all programs
  - Just need to keep programs that produce **distinct** values!

Observational Equivalence Reduction (OER)

- Key idea: Many programs, but (potentially much) fewer values produced by them
- If we aim to satisfy examples, no need to keep track of all programs
  - Just need to keep programs that produce **distinct** values!
  - E.g., if example is *10 —> 50*, then only need to keep one program for set of programs $\{ x+10,\ 10+10,\ 10+x,\ x+1+9,\ x+2+8,\ \ldots \}$

Summary

- Deduction-based Pruning Techniques in Top-Down Search
- Observational Equivalence Reduction in Bottom-Up Search