#21: Type-directed Synthesis

Sankha Narayan Guria

EECS 700: Introduction to Program Synthesis

Last week

Behavioral constraints

assertions
types
pre/post-conditions

+ bounded guarantees
+ unbounded guarantees

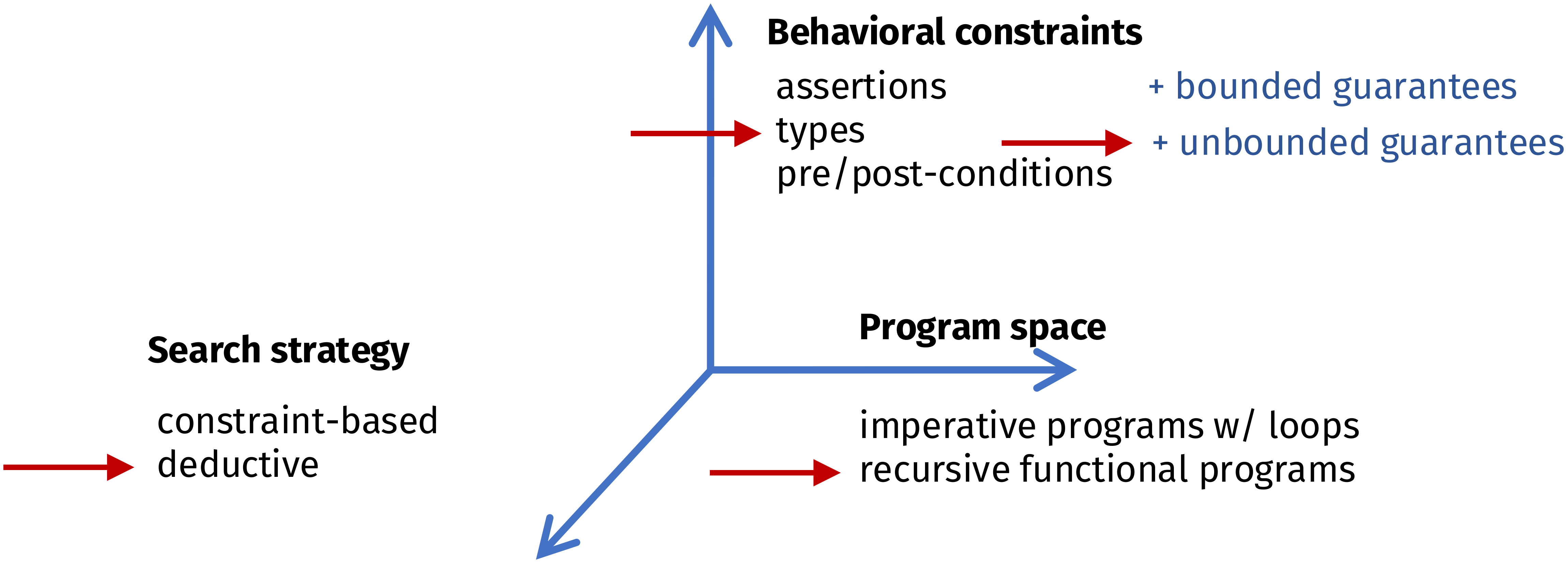
Program space

imperative programs w/ loops
recursive functional programs

Search strategy

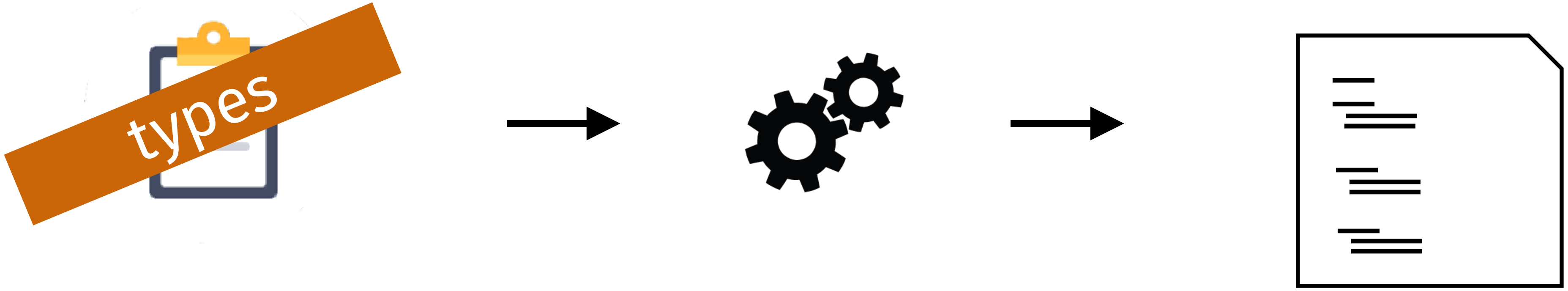
constraint-based
deductive

This week


Behavioral constraints
assertions
types
pre/post-conditions
+ bounded guarantees
+ unbounded guarantees
Program space
imperative programs w/ loops
recursive functional programs
Search strategy
constraint-based
deductive

Type-driven program synthesis

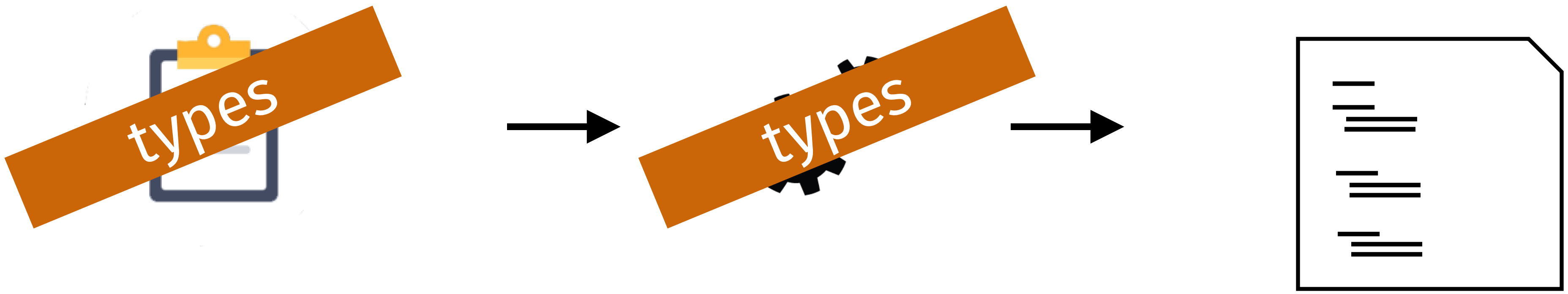
specification search program



programmer-friendly
informative

Type-driven program synthesis

specification — search — program



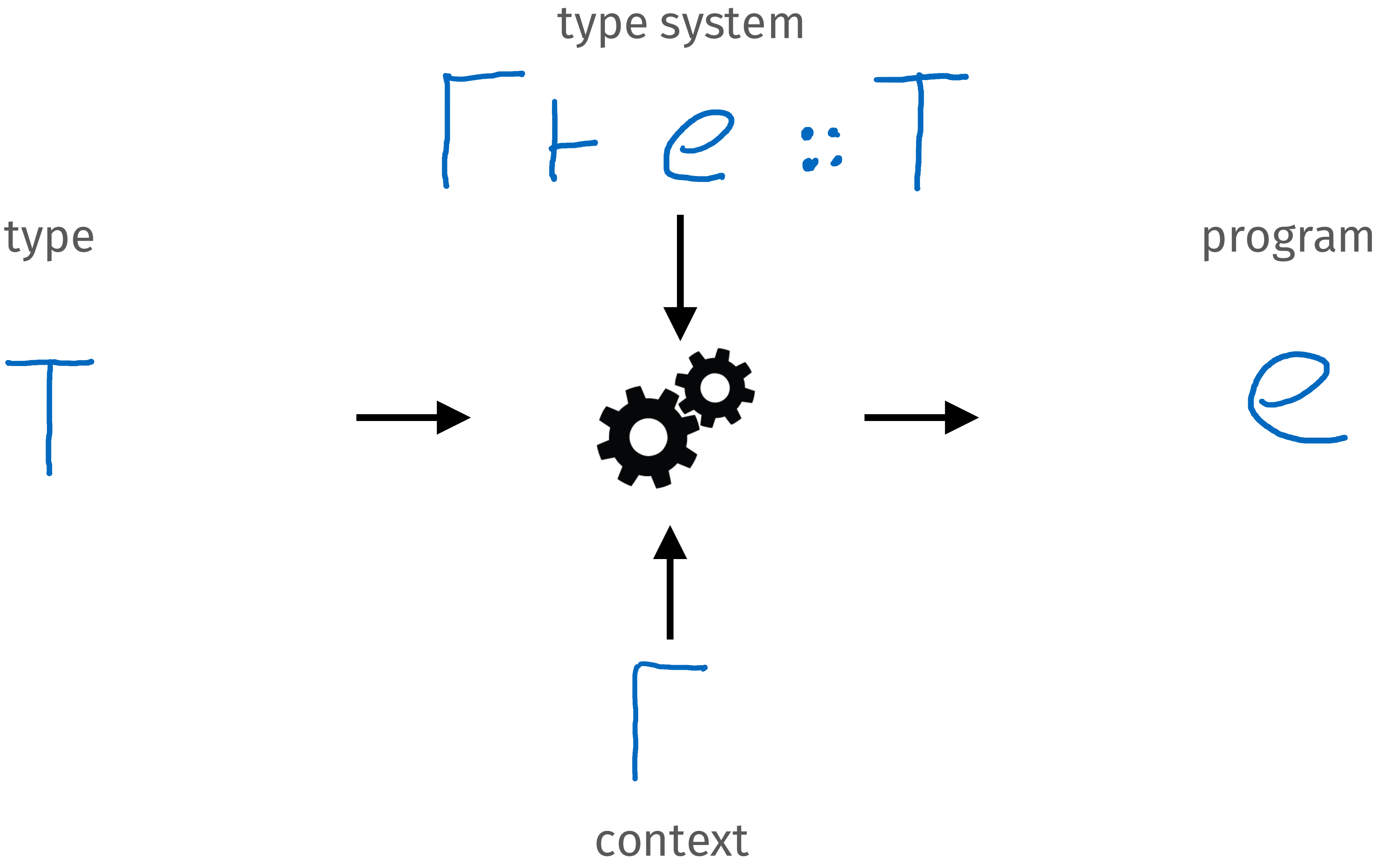
programmer-friendly
informative

help prune
the search space

Which program do I have in mind?

`Char -> String -> [String]`	split string at custom separator
`a -> Int -> [a]`	list with n copies of input value

Type-driven program synthesis



This week

intro to type systems

enumerating well-typed terms

bidirectional type systems

synthesis with types and examples

polymorphic types

refinement types

synthesis with refinement types

This week

intro to type systems

enumerating well-typed terms

bidirectional type systems

synthesis with types and examples

polymorphic types

refinement types

synthesis with refinement types

What is a type system?

Deductive system for proving facts about programs and types

Defined using *inference rules* over *judgments*

typing judgement

program / term

context → $\Gamma \vdash e :: T$ ← type

"under context Gamma, term e has type T"

A simple type system: syntax

$e ::= 0 \mid e+1 \mid x \mid e\ e \mid \lambda x.e$ -- expressions

example program: increment by two

$$\lambda x.\,(x+1)+1$$

A simple type system: syntax

$e ::= 0 \mid e+1 \mid x \mid e\ e \mid \lambda x.e$ -- expressions

$T ::= \mathrm{Int} \mid T \to T$ -- types

$\Gamma ::= \cdot \mid x:T, \Gamma$ -- contexts

Inference rules = typing rules

$$\text{t-zero}\ \frac{}{\Gamma \vdash 0 :: \mathrm{Int}} \qquad \text{t-suc}\ \frac{\Gamma \vdash e :: \mathrm{Int}}{\Gamma \vdash e+1 :: \mathrm{Int}} \qquad \text{t-var}\ \frac{x : T \in \Gamma}{\Gamma \vdash x :: T}$$

$$\text{t-abs}\ \frac{\Gamma, x : T_1 \vdash e :: T_2}{\Gamma \vdash \lambda x.e :: T_1 \to T_2} \qquad \text{t-app}\ \frac{\Gamma \vdash e_1 :: T' \to T \qquad \Gamma \vdash e_2 :: T'}{\Gamma \vdash e_1\, e_2 :: T}$$

Typing derivations

A derivation of $\Gamma \vdash e :: T$ is a tree where

1. the root is $\Gamma \vdash e :: T$
2. children are related to parents via inference rules
3. all leaves are axioms

Typing derivations

let's build a derivation of

$$\cdot \vdash \lambda x.\, x + 1 :: \mathsf{Int} \rightarrow \mathsf{Int}$$

we say that $\lambda x.\, x + 1$ is well-typed in the empty context
and has type Int $\rightarrow$ Int

Typing derivations

$$\text{t-zero}\ \frac{}{\Gamma \vdash 0 :: \text{Int}} \qquad \text{t-suc}\ \frac{\Gamma \vdash e :: \text{Int}}{\Gamma \vdash e+1 :: \text{Int}}$$

$$\text{t-var}\ \frac{x : T \in \Gamma}{\Gamma \vdash x :: T} \qquad \text{t-abs}\ \frac{\Gamma, x : T_1 \vdash e :: T_2}{\Gamma \vdash \lambda x . e :: T_1 \to T_2}$$

$$\text{t-app}\ \frac{\Gamma \vdash e_1 :: T' \to T \qquad \Gamma \vdash e_2 :: T'}{\Gamma \vdash e_1\, e_2 :: T}$$

$$\cdot \vdash \lambda x. x + 1 :: \text{Int} \to \text{Int}$$

Typing derivations

is $(\lambda x.\, x) + 1$ well-typed (in the empty context)?

no! no way to build a derivation of $\cdot \vdash (\lambda x.\, x) + 1 :: \_$

we say that $(\lambda x.\, x) + 1$ is ill-typed

Let’s add lists!

$$e ::= \ldots \mid [\,] \mid e : e \mid \text{match } e \text{ with } [\,] \to e \mid x : x \to e$$

$$T ::= \text{Int} \mid \text{List} \mid T \to T$$

Example program: head with default

$$\lambda x.\,\mathrm{match}\ x\ \mathrm{with}\ nil \to 0 \mid y : ys \to y$$

Typing rules

$$\text{t-nil}\ \frac{}{\Gamma \vdash [\,] :: \mathrm{List}}$$

$$\text{t-cons}\ \frac{\Gamma \vdash e_1 :: \mathrm{Int} \qquad \Gamma \vdash e_2 :: \mathrm{List}}{\Gamma \vdash e_1 : e_2 :: \mathrm{List}}$$

what should the t-match tule be?

$$\text{t-match}\ \frac{\Gamma \vdash e_0 :: \boxed{\quad\quad}^{1} \qquad \Gamma \vdash e_1 :: \boxed{\quad\quad}^{2} \qquad \Gamma\, \boxed{\quad\quad\quad\quad\quad\quad}^{4} \vdash e_2 :: \boxed{\quad\quad}^{3}}{\Gamma \vdash \mathtt{match}\ e_0\ \mathtt{with}\ [\,] \to e_1 \mid x : xs \to e_2 :: T}$$