Type-Guided Synthesis

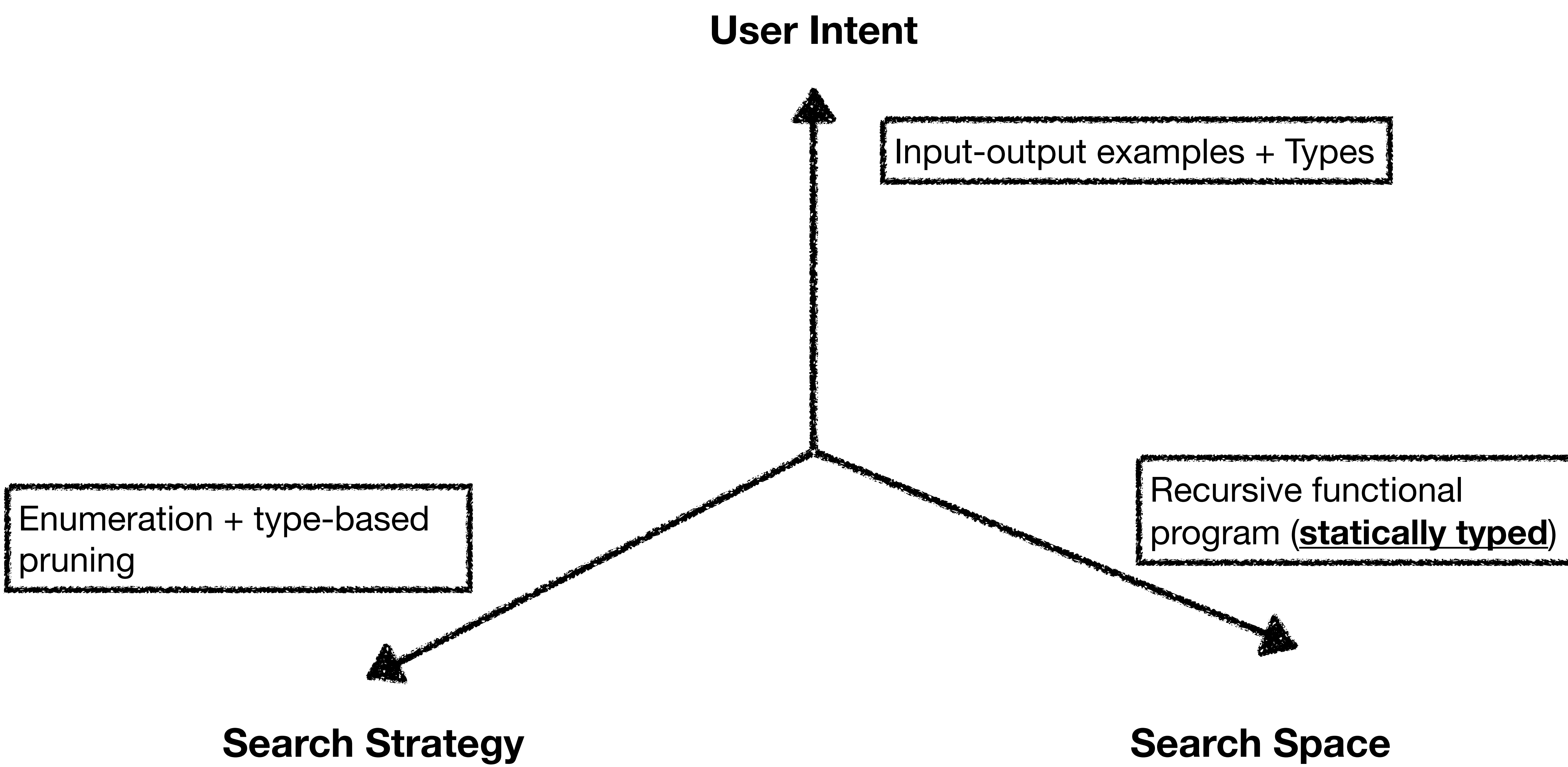
Woosuk Lee

CSE9116 SPRING 2024

Hanyang University

Some part is from https://github.com/nadia-polikarpova/cse291-program-synthesis/blob/master/lectures/Lecture11.pdf

Dimension


User Intent
Input-output examples + Types
Enumeration + type-based pruning
Recursive functional program (statically typed)
Search Strategy
Search Space

Static Types and Dynamic Types

- Statically typed languages: type checking is done at compile-time.
  - All type errors are detected before program executions thanks to a *sound* type system.
  - ML, Haskell, Scala, etc
- Dynamically typed languages: type checking is done at run-time.
  - type errors are detected during program executions
  - Python, JavaScript, Ruby, Lisp, etc

Type for Search Space Pruning

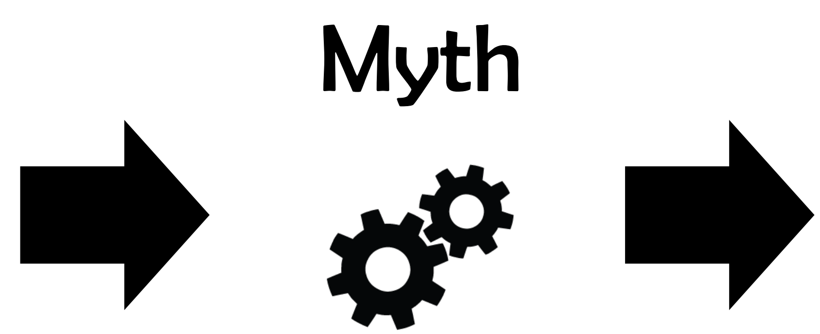
- So far, any legal programs wrt a grammar were considered valid.
- In synthesis for a static type-based language, we can further prune the search space by ruling out programs invalid wrt *types*.
    - By leveraging a sound type system

Example

```
type nat =
  | O
  | S of nat

type list =
  | Nil
  | Cons of nat * list

type cmp =
  | LT
  | EQ
  | GT

let rec compare (n1 : nat) (n2 :nat) : cmp =
  match n1 with
  | O -> (match n2 with
           | O -> EQ
           | S (m) -> LT
         )
  | S (m1) ->
      ( match n2 with
      | O -> GT
      | S (m2) -> (compare m1 m2) )
;;
```

Example

```
let list_compress : list -> list |>
{   [] => []
  | [0] => [0]
  | [1] => [1]
  | [0;0] => [0]
  | [1;1] => [1]
  | [2;0] => [2;0]
  | [1;0;0] => [1;0]
  | [0;1;1] => [0;1]
  | [2;1;0;0] => [2;1;0]
  | [2;2;1;0;0] => [2;1;0]
  | [2;2;0] => [2;0]
  | [2;2;2;0] => [2;0]
  | [1;2;2;2;0] => [1;2;0]
} = ?
```



Example

```
let list_compress : list -> list =
  let rec f1 (l1:list) : list =
    match l1 with
      | Nil -> Nil
      | Cons (n1, l2) -> (match f1 l2 with
                            | Nil -> l1
                            | Cons (n2, l3) -> (match compare n2 n1 with
                                                  | LT -> Cons (n1,
                                                            Cons (n2, l3))
                                                  | EQ -> Cons (n1, l3)
                                                  | GT -> Cons (n1,
                                                            Cons (n2, l3))))
 in
   f1
```

Osera, Zdancewic, Type-and-Example-Directed Program Synthesis. 2015

Contents

- Introduction to type systems
- Enumeration of well-typed programs
- Synthesis with types and examples

What is a Type System?

- System for proving facts about programs' types
- Defined using *inference rules* over judgments

Inference Rules

- An inference rule is of the form: $\dfrac{A}{B}$
  - A : hypothesis (antecedent)
  - B : conclusion (consequent)
  - "If A is true then B is also true"
  - $\dfrac{}{B}$: axiom (inference rule without hypothesis)
- The hypothesis may contain multiple statements: $\dfrac{A \quad B}{C}$
  - "If both A and B are true then so is C"

A Simple Language

- Grammar

$$E \rightarrow n \mid x \mid E \oplus E \mid \lambda x \,.\, E \mid E\ E \qquad (\oplus \in \{\ +\ ,\ -\ \})$$

- Examples
  - $\lambda x \,.\, x - 11$
  - $(\lambda x \,.\, x - 11)\ 12$
  - $\lambda x \,.\, \lambda y \,.\, x + y + 1$
  - $\lambda f \,.\, (f\ 3) + (f\ 4)$

Types

- Types are defined inductively:

$$T \to \text{int} \mid T \to T$$

- Examples:
  - int
  - int → int
  - int → int → int
  - (int → int) → (int → int)

Types of Expressions

- In order to compute the type of an expression, we need type environment:

$$\Gamma : Var \to T$$

- Judgements:

  $\Gamma \vdash e : t \iff$ Under type environment Γ, expression e has type t

Examples

- $[] \vdash 3 : \text{int}$

- $[x \mapsto \text{int}] \vdash x : \text{int}$

- $[] \vdash 3 + 4 :$

- $[] \vdash \lambda x \,.\, x - 11 :$

- $[] \vdash \lambda x \,.\, \lambda y \,.\, x + y + 1 :$

- $[f \mapsto \text{int} \to \text{int}] \vdash (f\ (f\ 1)) :$

Typing Rules

Inductive rules for assigning types to expressions:

$$\frac{}{\Gamma \vdash n : \text{int}}$$

$$\frac{}{\Gamma \vdash x : \Gamma(x)}$$

$$\frac{\Gamma \vdash E_1 : \text{int} \qquad \Gamma \vdash E_2 : \text{int}}{\Gamma \vdash E_1 \oplus E_2 : \text{int}}$$

$$\frac{\Gamma \vdash E_1 : t_1 \to t_2 \qquad \Gamma \vdash E_2 : t_1}{\Gamma \vdash E_1\ E_2 : t_2}$$

Let x bound to t_1 in Γ (any previous binding of x is forgotten)

$$\frac{[x \mapsto t_1]\Gamma \vdash E : t_2}{\Gamma \vdash \lambda x\,.\,E : t_1 \to t_2}$$

We say that a closed expression E has type t iff we can derive $[] \vdash E : t$

Example

$$\frac{}{[\,] \vdash \lambda x\,.\,x - 11 : \text{int} \to \text{int}}$$

Example

$$\frac{}{[\,] \vdash (\lambda x\,.\,x)\ 1 : \mathrm{int}}$$

Example

$$\frac{}{[] \vdash \lambda x\,.\,\lambda y\,.\,x + (y + 1) : \text{int} \to (\text{int} \to \text{int})}$$

Example

$$\overline{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}$$
$$[] \vdash (\lambda x \,.\, x) + 1 \,:\, ?$$

Soundness of Type System

- If E is a closed expression such that $[] \vdash E : t$, E evaluates to a value v such that $v : t$

Contents

- Introduction to type systems
- Enumeration of well-typed programs
- Synthesis with types and examples

Add Lists

- Grammar

$$E \rightarrow n \mid x \mid E \oplus E \mid \lambda x \,.\, E \mid E\ E$$
$$\mid [] \mid E :: E \mid \text{match } E \text{ with } [] \rightarrow E \mid x :: x \rightarrow E$$

- Types

$$T \rightarrow \text{int} \mid \text{list} \mid T \rightarrow T$$

- Examples
  - 1 :: 2 :: []
  - match x with [] → 0 | hd :: tl → hd + 1

Typing Rules

$$\frac{}{\Gamma \vdash [] : \text{list}}$$

$$\frac{\Gamma \vdash E_1 : \text{int} \qquad \Gamma \vdash E_2 : \text{list}}{\Gamma \vdash E_1 :: E_2 : \text{list}}$$

$$\frac{\Gamma \vdash E_0 : \text{list} \qquad \Gamma \vdash E_1 : t \qquad [x \mapsto \text{int}, x' \mapsto \text{list}]\Gamma \vdash E_2 : t}{\Gamma \vdash \text{match } E_0 \text{ with } [] \to E_1 \mid x :: x' \to E_2 : t}$$

For brevity, we assume all lists in a program are integer lists

Example

$$\overline{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}$$

$$[] \vdash \lambda x\,.\,\mathrm{match}\ \ x\ \ \mathrm{with}\ [] - > 0 \mid x :: x' \to x + 1 : \mathrm{list} \to \mathrm{int}$$

Synthesis of Well-Typed Programs

- Program synthesis = proof search
- Given a type *t*, find a program *p* such that $[] \vdash p : t$
  - cf) in type inference: find a type for a given program
- Top-down enumeration and *reversely* applying typing rules
  - Can avoid enumeration of ill-typed programs

Peter-Michael Osera, Steve Zdancewic, Type-and-example-directed Program Synthesis, 2015

Review: Typing Rules

[T-INT]
$$\frac{}{\Gamma \vdash n : \text{int}}$$

[T-VAR]
$$\frac{}{\Gamma \vdash x : \Gamma(x)}$$

[T-BOP]
$$\frac{\Gamma \vdash E_1 : \text{int} \qquad \Gamma \vdash E_2 : \text{int}}{\Gamma \vdash E_1 \oplus E_2 : \text{int}}$$

[T-CALL]
$$\frac{\Gamma \vdash E_1 : t_1 \rightarrow t_2 \qquad \Gamma \vdash E_2 : t_1}{\Gamma \vdash E_1\ E_2 : t_2}$$

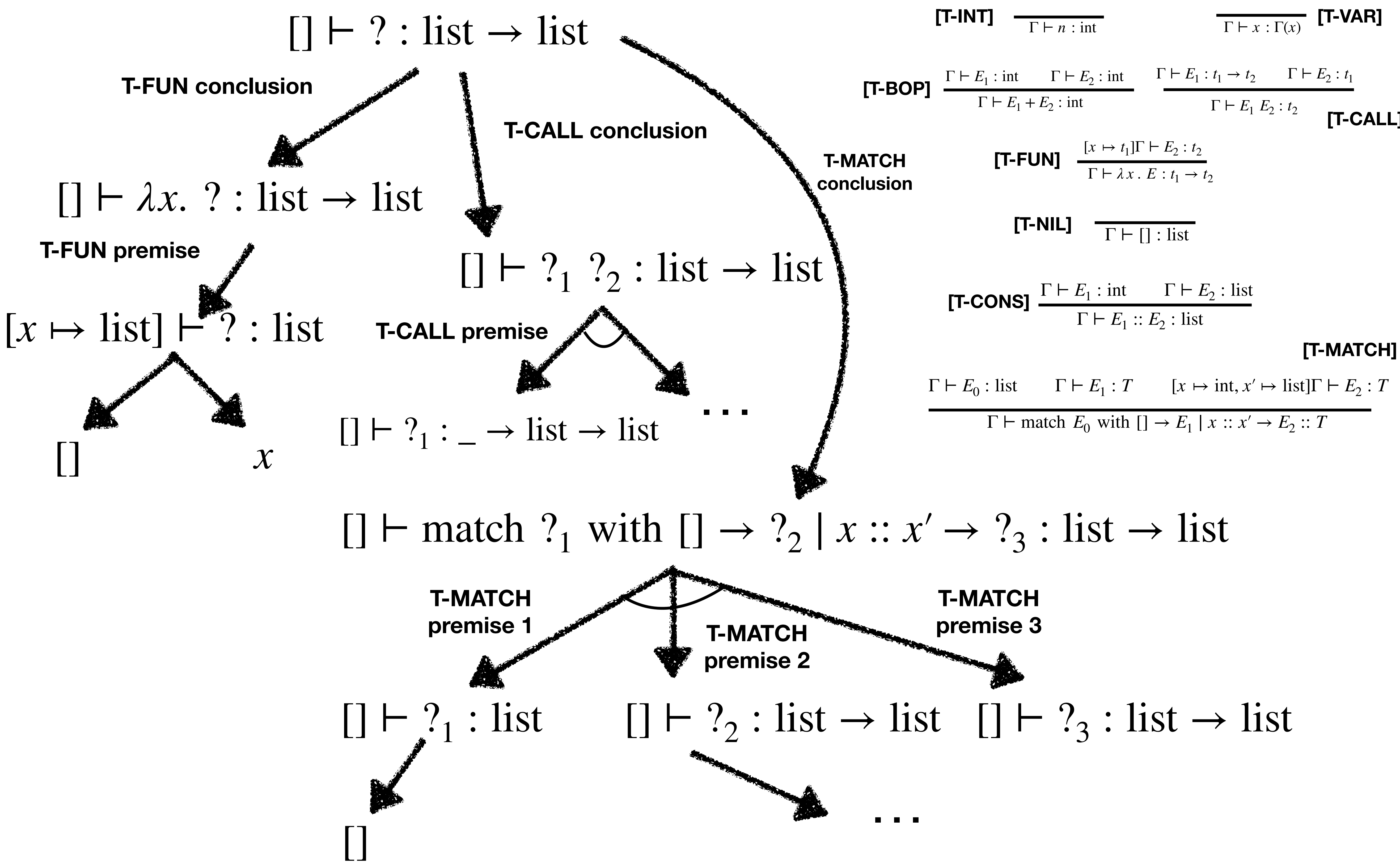
[T-FUN]
$$\frac{[x \mapsto t_1]\Gamma \vdash E_2 : t_2}{\Gamma \vdash \lambda x\,.\,E : t_1 \rightarrow t_2}$$

[T-NIL]
$$\frac{}{\Gamma \vdash [\,] : \text{list}}$$

[T-CONS]
$$\frac{\Gamma \vdash E_1 : \text{int} \qquad \Gamma \vdash E_2 : \text{list}}{\Gamma \vdash E_1 :: E_2 : \text{list}}$$

[T-MATCH]
$$\frac{\Gamma \vdash E_0 : \text{list} \qquad \Gamma \vdash E_1 : t \qquad [x \mapsto \text{int}, x' \mapsto \text{list}]\Gamma \vdash E_2 : t}{\Gamma \vdash \text{match } E_0 \text{ with } [\,] \rightarrow E_1 \mid x :: x' \rightarrow E_2 : t}$$

Type-Guided Top-Down Enumeration



Redundant Programs

- Equivalent programs are enumerated.
  - $(\lambda x \, . \, x)$ [] $\qquad$ []
  - $\lambda x \, . \, ((\lambda y \, . \, y + 1) \; x) \qquad \lambda y \, . \, y + 1$
  - $\text{match [] with []} \rightarrow E_1 \mid x :: x' \rightarrow E_2 \qquad E_1$
- Need to restrict syntax to avoid enumerating redundant programs

New Syntax

- Two reductions:
  - β-reduction: $(\lambda x \,.\, E_1)\ E_2 \Longrightarrow [x \mapsto E_2]E_1$
  - η-reduction: $\lambda x \,.\, (f\ x) \Longrightarrow f$

Replace all occurrences of x in E_1 with E_2

- Grammar

$$E \rightarrow x \mid E\ I$$
$$I \rightarrow n \mid I \oplus I \mid \lambda x \,.\, I \mid [] \mid I :: I \mid \text{match } E \text{ with } [] \rightarrow I \mid x :: x \rightarrow I$$

- Can generate only β,η-normal forms (can't apply β,η-reductions any more)
  - All function calls are of form $(x\ I_1\ \ \ldots\ \ I_k)$
  - No function calls inside function bodies

Contents

- Introduction to type systems
- Enumeration of well-typed programs
- Synthesis with types and examples

Adding Examples (Spec = Type + I/O Examples)

- Set of values $Val = \mathbb{Z} + \mathbb{Z}^*$
- Set of input-output examples $ex \in X = Val \to Val$
- Types refined with examples $R = T \triangleright X$
- (Refined) typing environment $\Gamma : Var \to R$
- (Refined) typing judgement $\Gamma \vdash E : r$

Refined Typing Rules

[R-INT]

$$\frac{}{\Gamma \vdash n : \text{int} \rhd ex} \; range(ex) = \{n\}$$

[R-VAR]

$$\frac{}{\Gamma \vdash x : \Gamma(x)}$$

[R-CALL]

$$\frac{\Gamma \vdash E : t_1 \rightarrow t_2 \qquad \Gamma \vdash I : t_1 \rhd [i_j \mapsto m_j \mid 1 \leq j \leq k, [\![E\ m_j]\!] = n_j]}{\Gamma \vdash E\ I : t_2 \rhd [i_1 \mapsto n_1, \cdots, i_k \mapsto n_k]}$$

Some rules are omitted for brevity.

Refined Typing Rules

[R-FUN]

$$\frac{[x \mapsto t_1 \rhd ex_{\mathrm{ID}}]\Gamma \vdash I : t_2 \rhd [i_1 \mapsto n_1, \cdots, i_k \mapsto n_k]}{\Gamma \vdash \lambda x \,.\, I : t_1 \rightarrow t_2 \rhd [i_1 \mapsto n_1, \cdots, i_k \mapsto n_k]} \quad ex_{\mathrm{ID}} = [i_1 \mapsto i_1, \cdots, i_k \mapsto i_k]$$

[R-NIL]

$$\frac{}{\Gamma \vdash [] : \mathrm{list} \rhd [i_1 \mapsto [], \cdots, i_k \mapsto []]}$$

Some rules are omitted for brevity.

Refined Typing Rules

[R-CONS]

$$\frac{\Gamma \vdash I_1 : \text{int} \rhd [i_1 \mapsto n_1, \cdots, i_k \mapsto n_k] \qquad \Gamma \vdash I_2 : \text{list} \rhd [i_1 \mapsto l_1, \cdots, i_k \mapsto l_k]}{\Gamma \vdash I_1 :: I_2 : \text{list} \rhd [i_1 \mapsto n_1 :: l_1, \cdots, i_k \mapsto n_k :: l_k]}$$

Some rules are omitted for brevity.

Example: Singleton List

$ex_{\mathrm{ID}} = [0 \mapsto 0, 1 \mapsto 1]$

$$[] \vdash \, ? : \mathrm{int} \to \mathrm{list} \rhd [0 \mapsto [0], 1 \mapsto [1]] \implies \, ? = \lambda x \, . \, x :: []$$

R-FUN conclusion

$$[] \vdash \lambda x. \, ? : \mathrm{int} \to \mathrm{list} \rhd [0 \mapsto [0], 1 \mapsto [1]]$$

R-FUN premise

$$[x \mapsto \mathrm{int} \rhd ex_{\mathrm{ID}} \vdash \, ? : \mathrm{list} \rhd [0 \mapsto 0 :: [], 1 \mapsto 1 :: []]$$

R-CONS conclusion

$$[x \mapsto \mathrm{int} \rhd ex_{\mathrm{ID}} \vdash \, ?_1 :: ?_2 : \mathrm{list} \rhd [0 \mapsto 0 :: [], 1 \mapsto 1 :: []]$$

R-CONS premise 1

$$[x \mapsto \mathrm{int} \rhd X_{\mathrm{ID}} \vdash \, ?_1 : \mathrm{int} \rhd X_{\mathrm{ID}}$$

R-VAR conclusion

$$?_1 = x$$

R-CONS premise 2

$$[x \mapsto \mathrm{int} \rhd X_{\mathrm{ID}} \vdash \, ?_2 : \mathrm{list} \rhd [0 \mapsto [], 1 \mapsto []]$$

R-NIL conclusion

$$?_2 = []$$