CS 162 Programming languages

Final Review

Yu Feng

Winter 2020

Final exam

- 3 hours
- 120 points in total
- One A4 cheat sheet
- Search engine is NOT allowed
- Execute programs locally is NOT allowed
- ProctorU: https://www.proctoru.com/portal/ucsb
- Questions
    - Multiple choices
    - Short answers
    - Program output
    - Programming in OCaml, Racket, and Datalog (difficult than midterm)

Final review

- Lambda-calculus (α-renaming, β-reduction, evaluation)
- OCaml basics (let-binding, list, tuple, datatype, recursion)
- Datatypes (How to access? How to construct?)
- Recursion (tail recursion)
- Higher-order functions (map, fold, filter, etc.)
- Closure

Final review

- Type inference (Given a new type system)
  - Understand typing rules
  - Constraint generation
  - Unification
- Racket programming (let, let*, etc.)
- Solver-aided programming
  - Symbolic variables
  - Solver-aided queries (solve, synthesis, verify, debug)

Final review

- Basic concepts in Datalog
- Program analysis in Datalog
- Pointer analysis (core ideas, complexity, trade-off)
- Information flow (taint) analysis
- Design & implement your own analysis

Simple-typed lambda calculus

$$e ::= x \mid \lambda x : \tau \,.\, e \mid e_1\ e_2 \mid n$$

$$\tau ::= \textbf{int} \mid X \mid \tau_1 \to \tau_2$$

To formally define type inference, we introduce a new typing relation:

$$\Gamma \vdash e : \tau \triangleright C$$

Type environment

Meaning: Expression e has type τ provided that every constraint in the set C is satisfied.

Typing rules for STLC

$$\textsc{CT-Var}\ \frac{x:\tau \in \Gamma}{\Gamma \vdash x:\tau \triangleright \emptyset}$$

If x is of type τ in Env
then x has type τ

$$\textsc{CT-Int}\ \frac{}{\Gamma \vdash n:\textbf{int} \triangleright \emptyset}$$

All natural numbers have type int/number

$$\textsc{CT-Abs}\ \frac{\Gamma, x:\tau_1 \vdash e:\tau_2 \triangleright C}{\Gamma \vdash \lambda x:\tau_1.\, e:\tau_1 \to \tau_2 \triangleright C}$$

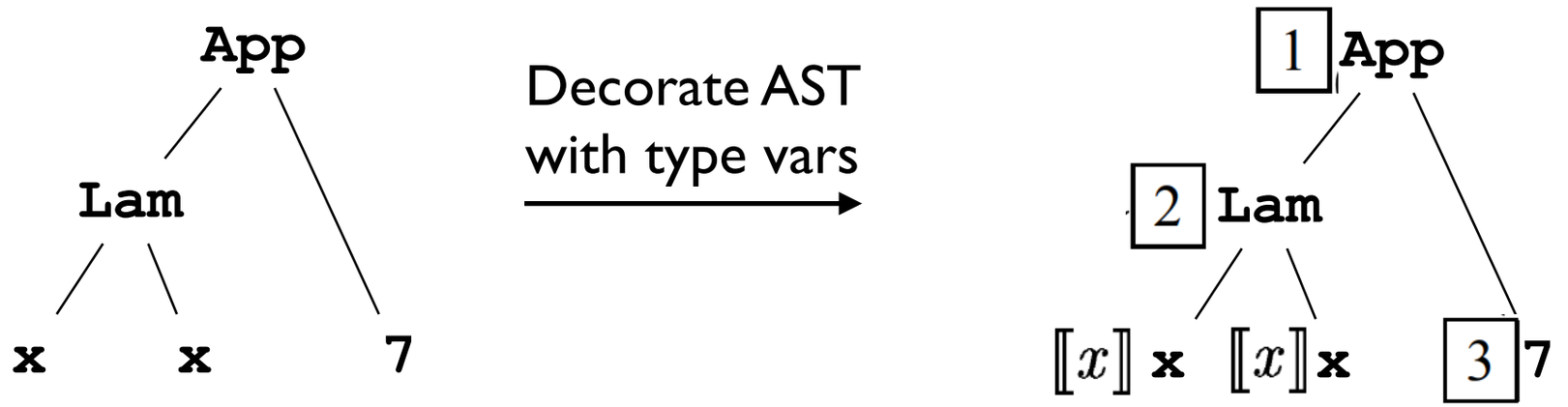
If formal parameter x is type τ1, and body e is type τ2, then the function is type τ1->τ2

$$\textsc{CT-App}\ \frac{\Gamma \vdash e_1:\tau_1 \triangleright C_1 \quad \Gamma \vdash e_2:\tau_2 \triangleright C_2 \quad C' = C_1 \cup C_2 \cup \{\tau_1 = \tau_2 \to X\}}{\Gamma \vdash e_1\ e_2:X \triangleright C'}\ X \text{ is fresh}$$

If function e1 is type τ1 which is in the form of τ2-> X,
and the argument e2 is type τ2, then the function call is type X

Example

What is the type of App(Lam(x, x), 7)?



Generate constraints
using typing rules

$$\boxed{2} = \boxed{3} \to \boxed{1}$$

$$\boxed{2} = [\![x]\!] \to [\![x]\!]$$

$$\boxed{3} = \text{number}$$

How to solve those constraints?

Unification algorithm

$$
\begin{aligned}
\mathit{unify}(\mathrm{C}) \quad = \quad & \textbf{if } \mathrm{C} = \emptyset, \textbf{ then } \{\,\} \\
& \textbf{else let } \{\mathtt{S} = \mathtt{T}\} \cup \mathrm{C}' = \mathrm{C} \textbf{ in} \\
& \quad \textbf{if } \mathtt{S} = \mathtt{T} \\
& \quad\quad \textbf{then } \mathit{unify}(\mathrm{C}') \\
& \quad \textbf{else if } \mathtt{S} = \mathtt{X} \textbf{ and } \mathtt{X} \notin \mathrm{FV}(\mathtt{T}) \\
& \quad\quad \textbf{then } \mathit{unify}(\{\mathtt{X} \mapsto \mathtt{T}\}\mathrm{C}') \circ \{\mathtt{X} \mapsto \mathtt{T}\} \\
& \textbf{else if } \mathtt{T} = \mathtt{X} \textbf{ and } \mathtt{X} \notin \mathrm{FV}(\mathtt{S}) \\
& \quad \textbf{then } \mathit{unify}(\{\mathtt{X} \mapsto \mathtt{S}\}\mathrm{C}') \circ \{\mathtt{X} \mapsto \mathtt{S}\} \\
& \textbf{else if } \mathtt{S} = \mathtt{S}_1 \to \mathtt{S}_2 \textbf{ and } \mathtt{T} = \mathtt{T}_1 \to \mathtt{T}_2 \\
& \quad \textbf{then } \mathit{unify}(\mathrm{C}' \cup \{\mathtt{S}_1 = \mathtt{T}_1, \mathtt{S}_2 = \mathtt{T}_2\}) \\
& \textbf{else} \\
& \quad \textbf{fail}
\end{aligned}
$$

Apply substitution to both the remaining constraint C' and the current mapping

Occurrence check: make sure T does not contain type variable X

Types and Programming Languages p326

Example

How to solve those constraints?

$$[\![2]\!] = [\![3]\!] \to [\![1]\!]$$

$$[\![2]\!] = [\![x]\!] \to [\![x]\!]$$

$$[\![3]\!] = \text{number}$$

Apply substitution to both the remaining constraint C' and the current mapping

Action	Stack	Substitution
Initialize	$[\![2]\!] = [\![3]\!] \to [\![1]\!]$ $[\![2]\!] = [\![x]\!] \to [\![x]\!]$ $[\![3]\!] = \text{number}$	empty
Step 3	$[\![3]\!] \to [\![1]\!] = [\![x]\!] \to [\![x]\!]$ $[\![3]\!] = \text{number}$	$[\![2]\!] \mapsto [\![3]\!] \to [\![1]\!]$
Step 5	$[\![3]\!] = [\![x]\!]$ $[\![1]\!] = [\![x]\!]$ $[\![3]\!] = \text{number}$	$[\![2]\!] \mapsto [\![3]\!] \to [\![1]\!]$
Step 3	$[\![1]\!] = [\![x]\!]$ $[\![x]\!] = \text{number}$	$[\![2]\!] \mapsto [\![x]\!] \to [\![1]\!]$ $[\![3]\!] \mapsto [\![x]\!]$
Step 3	$[\![x]\!] = \text{number}$	$[\![2]\!] \mapsto [\![x]\!] \to [\![x]\!]$ $[\![3]\!] \mapsto [\![x]\!]$ $[\![1]\!] \mapsto [\![x]\!]$
Step 3	empty	$[\![2]\!] \mapsto \text{number} \to \text{number}$ $[\![3]\!] \mapsto \text{number}$ $[\![1]\!] \mapsto \text{number}$ $[\![x]\!] \mapsto \text{number}$

https://cs.brown.edu/courses/cs173/2002/Lectures/2002-11-13.pdf

Questions?