COMP 3351 Programming Languages[1]

Christian Grothoff
christian@grothoff.org

http://grothoff.org/christian/

[1] Based on notes by Prof. Jens Palsberg, UCLA

Today

- λ-calculus exercises
- Type soundness: definition
- Type system for the λ-calculus
- Type soundness: proof structure
- Type soundness proof for λ-calculus
- Featherweight Java

Calculate (execute)

$$(\lambda a.\lambda b.\lambda c.b) \quad 5 \quad \lambda a.4$$

Calculate (execute)

$$(\lambda a.\lambda b.\lambda c.a \quad b \quad c)\lambda x.x \quad \lambda a.4 \quad 5$$

Calculate (execute)

$$((\lambda f.\lambda x.fx)(\lambda x.x \quad x))(\lambda x.x)$$

Calculate (execute)

$$(\lambda z.\lambda t.z \quad t)4 \quad \lambda z.z$$

Type Soundness

- A program is a closed expression. $(a \quad b)$ is not a program (because it contains free variables).
- A value is either a λ-abstraction $(\lambda x.e)$ or a constant (c).
- A type system for a programming language is **sound** if *well-typed* programs cannot cause *type errors*.
- A "type error" generally corresponds to a program that has not been reduced to a value but still can not continue to execute.
- The type system must define "well-typed".

A Type Error!

$$(\lambda z.\lambda t.z \quad t)4 \quad \lambda z.z \rightarrow_V^* 4 \quad \lambda z.z$$

The program is "stuck".

Types for the λ-Calculus!

For the λ-calculus, we need two kinds of types: function types and an integer type.

Types are generated from the grammar:

$$t ::= t_1 \rightarrow t_2 \mid \texttt{Int}$$

Note that there are infinitely many types. Notice also that each type can be viewed as a tree. The size of the tree can be used to define a partial order over types.

Type Environments

A type environment Γ is a partial function with finite domain which maps elements of Var to types:

$$\Gamma = [x \mapsto \texttt{Int}, y \mapsto \texttt{Int} \rightarrow \texttt{Int}]$$

Examples: Types for Values

We write $\Gamma \vdash e : t$ to denote that expression e has type t in the type environment Γ:

$$\emptyset \vdash 4 : \texttt{Int}$$

$$\emptyset \vdash \lambda x.(\texttt{succ}\, x) : \texttt{Int} \rightarrow \texttt{Int}$$

UNIVERSITY OF DENVER

Type Rules for the λ-calculus

$$\Gamma \vdash x : \; t \qquad \text{if } \Gamma(x) = t \tag{1}$$

Type Rules for the λ-calculus

$$\frac{\Gamma[x : s] \vdash e : \ t}{\Gamma \vdash \lambda x.e : \ s \rightarrow t} \tag{2}$$

Type Rules for the λ-calculus

$$\frac{\Gamma \vdash e_1 : \; s \rightarrow t \qquad \Gamma \vdash e_2 : s}{\Gamma \vdash e_1 e_2 : t} \tag{3}$$

Type Rules for the λ-calculus

$$\Gamma \vdash c : \texttt{Int} \tag{4}$$

Type Rules for the λ-calculus

$$\frac{\Gamma \vdash e : \texttt{Int}}{\Gamma \vdash \texttt{succ}\, e : \texttt{Int}} \tag{5}$$

UNIVERSITY OF DENVER

Well-typed expressions

- An expression e is well-typed if there exist Γ and t so that $\Gamma \vdash e : t$ is derivable.

Example: Type Derivation

$$\dfrac{\dfrac{\dfrac{\emptyset[f:s\rightarrow t][x:s]\vdash f:s\rightarrow t \qquad \emptyset[f:s\rightarrow t][x:s]\vdash x:s}{\emptyset[f:s\rightarrow t][x:s]\vdash fx:t}}{\emptyset[f:s\rightarrow t]\vdash \lambda x.fx:s\rightarrow t}}{\emptyset\vdash \lambda f.\lambda x.fx:(s\rightarrow t)\rightarrow(s\rightarrow t)}$$

Example: Failing Type Derivation

$$\frac{\emptyset \vdash \lambda x.e : \texttt{Int}}{\emptyset \vdash \texttt{succ}\,(\lambda x.e) : \texttt{Int}}$$

Type Soundness: Proof Structure

- Preservation
  - Substitution (with equal type) preserves type
  - Execution preserves type
- Progress
  - Certain types correspond to values (base case)
  - Closed expressions of other types can make progress
  - Progress does not change closedness

$\Rightarrow$ Well-typed programs cannot "go wrong".

Substitution

If $\Gamma[x : s] \vdash e : t$ and $\Gamma \vdash M : s$ then $\Gamma \vdash e[x := M] : t$.

Proof by Induction

- Each term e in the λ-calculus can be associated with a (finite) "size" based on the syntax tree for the calculus
- We will assume that the substitution lemma holds for a "smaller" term while we try to show that it holds for a "larger" term

Larger?

- $\lambda x.e$ is larger than e
- $e_1 e_2$ is larger than e_1 and/or e_2 (individually)
- $\texttt{succ}\, e$ is larger than e

Substitution

To show:

If $\Gamma[x : s] \vdash e : t$ and $\Gamma \vdash M : s$ then $\Gamma \vdash e[x := M] : t$.

We have: Since $\Gamma[x : s] \vdash e : t$, one of our five type-rules must have been used in the last step of the type derivation.

Case 1: $e \equiv y$

- Case 1a: $y \equiv x$. Then $y[x := M] = M$. Since $\Gamma[x : s] \vdash e : t$ we conclude $s = t$. From $\Gamma \vdash M : s$ and $s = t$ we conclude $\Gamma \vdash M : t$.

- Case 1b: $y \not\equiv x$. Then $y[x := M] = y$; from $\Gamma[x : s] \vdash y : t$ we conclude $\Gamma(y) = t$ and thus $\Gamma \vdash y : t$.

Case 2: $e \equiv \lambda y.e_1$

- Case 2a: $y \equiv x$. Then $(\lambda y.e_1)[x := M] \equiv \lambda y.e_1$. Since x does not occur free in $\lambda y.e_1$ we can use the derivation from $\Gamma[x : s] \vdash \lambda y.e_1 : t$ to produce a derivation of $\Gamma \vdash \lambda y.e_1 : t$.

- Case 2b: $y \not\equiv x$. Then $(\lambda y.e_1)[x := M] \equiv \lambda z.e_1[y := z][x := M]$ with z fresh. (continued)

Case 2b: $e \equiv \lambda y.e_1$, $y \not\equiv x$

The last step in the derivation of $\Gamma[x : s] \vdash e : t$ is of the form:

$$\frac{\Gamma[x : s][y : t_2] \vdash e_1 : t_1}{\Gamma[x : s] \vdash \lambda y.e_1 : t_2 \rightarrow t_1}$$

Hence $\Gamma[x : s][z : t_2] \vdash e_1[y := z] : t_1$. Note that e_1 and consequently $e_1[y := z]$ are "smaller" than $\lambda y.e_1$ and hence by induction hypothesis $\Gamma[z : t_2] \vdash e_1[y := z][x := M] : t_1$. With type rule (2) we can derive $\Gamma \vdash \lambda z.e_1[y := z][x := M] : t_2 \rightarrow t_1$.

Case 3: $e \equiv e_1 e_2$

The last step in the derivation of $\Gamma[x : s] \vdash e : t$ is of the form:

$$\frac{\Gamma[x : s] \vdash e_1 : t_2 \longrightarrow t \qquad \Gamma[x : s] \vdash e_2 : t_2}{\Gamma[x : s] \vdash e_1\ e_2 : t}$$

Using the induction hypothesis we get $\Gamma \vdash e_1[x := M] : t_2 \longrightarrow t$ and $\Gamma \vdash e_2[x := M] : t_2$; with rule (3) $\Gamma \vdash e_1[x := M]\, e_2[x := M] : t$ follows.

Case 4: $e \equiv c$

Obviously $c[x := M] \equiv c$. The entire derivation of $\Gamma[x : s] \vdash e : t$ is of the form $\Gamma[x : s] \vdash c : \texttt{Int}$. From rule (4) we have $\Gamma \vdash c : \texttt{Int}$.

Case 5: $e \equiv \texttt{succ}\, e_1$

Proof is similar to case 3.

Type Preservation

If $\Gamma \vdash e : t$ and $e \rightarrow_V e'$, then $\Gamma \vdash e' : t$.

Proof by Induction

- We use induction over the derivation of $\Gamma \vdash e : t$.
- In the proof, we assume that the theorem holds for a derivation of depth $n-1$ and show it for a derivation of depth n.
- The theorem is obvious for derivations of depth 0 since $e \rightarrow_V e'$ is impossible for those.

Case 1: $e \equiv x$

$e \rightarrow_V e'$ is not possible.

Case 2: $e \equiv \lambda x.e_1$

$e \rightarrow_V e'$ is not possible.

Case 3: $e \equiv e_1 e_2$

There are three subcases depending on which of the possible ways $e \rightarrow_V e'$ was used to make progress.

If either $e_1 e_2 \rightarrow_V e_1' e_2$ or $e_1 e_2 \rightarrow_V e_1 e_2'$ were used, $\Gamma \vdash e' : t$ follows from the induction hypothesis and rule (3).

Case 3c: $e \equiv (\lambda x.e_1)v$

Suppose

$$(\lambda x.e_1)v \rightarrow_V e_1[x := v]$$

was used. Then the last part of the derivation of $\Gamma \vdash e : t$ is of the form:

$$\frac{\frac{\Gamma[x:s] \vdash e_1 : t}{\Gamma \vdash \lambda x.e_1 : s \rightarrow t} \qquad \Gamma \vdash v : s}{\Gamma \vdash (\lambda x.e_1)v : t}$$

Using the substitution lemma, $\Gamma[x : s] \vdash e_1 : t$ and $\Gamma \vdash v : s$ we get $\Gamma \vdash e_1[x := v] : t$.

Case 4: $e \equiv c$

$e \rightarrow_V e'$ is not possible.

Case 5: $e \equiv \texttt{succ}\, e_1$

Again we look at two subcases depending on how $e \rightarrow_V e'$ happened.

If $e \equiv \texttt{succ}\, c_1$ and $e' \equiv c_2$ (where $\langle c_2 \rangle = \langle c_1 \rangle + 1$) then the type derivation of $\Gamma \vdash e : t$ was of the form $\Gamma \vdash \texttt{succ}\, c_1 : \texttt{Int}$ and from rule (4) we have $\Gamma \vdash c_2 : \texttt{Int}$.

UNIVERSITY OF DENVER

Case 5b: $e \equiv \texttt{succ}\, e_1$ and $e_1 \rightarrow_V e_2$

The last part of the derivation of $\Gamma \vdash e : t$ is then of the form:

$$\frac{\Gamma \vdash e_1 : \texttt{Int}}{\Gamma \vdash \texttt{succ}\, e_1 : \texttt{Int}}$$

From the induction hypothesis we have $\Gamma \vdash e_2 : \texttt{Int}$, so using rule (5) we derive $\Gamma \vdash \texttt{succ}\, e_2 : \texttt{Int}$.

Typable Value

If $\Gamma \vdash v : Int$, then v is of the form c.

If $\Gamma \vdash v : s \rightarrow t$ then v is of the form $\lambda x.e$.

Proof: Obvious from type rules 2 and 4.

Progress

If e is a closed expression, and $\Gamma \vdash e : t$ then either e is a value, or there exists e' such that $e \rightarrow_V e'$.

Proof by Induction

- Since $\Gamma \vdash e : t$ there must exist a type derivation for the term e
- We will assume that the progress lemma holds for a type derivation of size $n-1$ while we try to show that it holds for a type derivation of size n
- There are now five subcases depending on which of the type rules was the last one used in the derivation

Case 1: $e \equiv x$

The term is not closed.

Case 2: $e \equiv \lambda x.e$

The term is a value.

Case 3: $e \equiv e_1 e_2$

Since e is closed, e_1 and e_2 must be closed. The last step in the derivation of $\Gamma \vdash e_1 e_2 : t$ must be of the form

$$\frac{\Gamma \vdash e_1 : s \rightarrow t \qquad \Gamma \vdash e_2 : s}{\Gamma \vdash e_1 e_2 : t}$$

From the induction hypothesis we have that either e_1 is a value or there exists e_1' such that $e_1 \rightarrow_V e_1'$ (in which case we can make progress to $e_1' e_2$). Also, either e_2 is a value, or there exists e_2' such that $e_2 \rightarrow_V e_2'$ (in which case we can make progress to $e_1 e_2'$.

University of Denver

Case 3c: $e \equiv (\lambda x.e_3)e_2$

If both e_1 and e_2 are values, then according to the typeable value theorem e_1 must be of the form $\lambda x.e_3$ and hence

$$e_1 e_2 \rightarrow_V e_3[x := e_2]$$

Case 4: $e \equiv c$

The term is a value.

Case 5: $e \equiv \texttt{succ}\, e_1$

Since e is closed, e_1 is also closed. The last step in the derivation of $\Gamma \vdash e : t$ must be of the form

$$\frac{\Gamma \vdash e_1 : \texttt{Int}}{\Gamma \vdash \texttt{succ}\, e_1 : \texttt{Int}}$$

From the induction hypothesis we have that either e_1 is a value or there exists e_1' such that $e_1 \rightarrow_V e_1'$.

Case 5: $e \equiv \texttt{succ}\, e_1$ (continued)

If e_1 is a value, then from $\Gamma \vdash e_1 : \texttt{Int}$ and the typeable value lemma we have that e_1 is of the form c_1 and hence $\texttt{succ} c_1 \rightarrow c_2$ (where $\langle c_2 \rangle = \langle c_1 \rangle + 1$).

Otherwise, if there exists e_1' such that $e_1 \rightarrow_V e_1'$, then we can make progress using $\texttt{succ}\, e_1 \rightarrow_V \texttt{succ}\, e_1'$.

UNIVERSITY OF DENVER

Closedness Preservation

If e is closed, and $e \rightarrow_V e'$, then e' is closed.

Proof: Obvious.

Conclusion

Well-typed programs cannot go wrong.

Proof: Suppose we have a well-typed program e that is stuck at an expression e' with $e \rightarrow_V^* e'$. We know that e' is closed (closendess preservation) and well-typed (type preservation). But then there exists e'' so that $e' \rightarrow_V e''$ (progress), a contradiction (e' can not be stuck).

Questions

?

Question!

Can Java programs go wrong?

UNIVERSITY OF DENVER