2024 OPLSS: Foundations of Programming Languages - Lecture 2

Lecturer: Paul Downen
Note Takers: Aaryan Patel, Adriano Corbelino II, Anders Thuné, Thomas Porter

June 2024

1 The System

1.1 Syntax

The abstract syntax of the simply typed λ-calculus can be represented with the following grammar in BNF form:

$$\begin{array}{rrl} \textit{Variables} \ni & x, y, z & ::= \textit{foo} \mid \textit{bar} \mid \textit{baz} \mid \ldots \\ \textit{Constants} \ni & c & ::= \textit{true} \mid \textit{false} \\ \textit{Terms} \ni & M, N & ::= x \mid M\ N \mid \lambda x.\ M \mid c \mid \mathsf{if}\ M\ \mathsf{then}\ N_1\ \mathsf{else}\ N_2 \\ \textit{Values} \ni & v, w & ::= c \mid \lambda x.\ M \end{array}$$

The left-hand side of ::= defines the name of a set and the standard meta-variables ($x, c, M, \ldots$) that range over the elements of the set, while the right-hand side lists the syntactic forms of that set. This is an *inductive* definition. This means the not only does each of the forms on the right hand inhabit the set declared on the left-hand side, but these forms are distinct from each other, and exhaustively cover the set.

Note the distinction between these meta-variables and program variables (*foo*, *bar*, ...), which are part of the language syntax.

1.2 Dynamic Semantics

The small-step operational semantics can be defined by declaring the β rules and the evaluation context.

1.2.1 Beta Rules

These rules are axioms that define how to evaluate terms.

$$\begin{array}{ll} (\beta \to) & \lambda x.\ M\ N \mapsto_\beta M[N/x] \\ (\beta\ \mathrm{Bool}_1) & \mathsf{if}\ \textit{true}\ \mathsf{then}\ N_1\ \mathsf{else}\ N_2 \mapsto_\beta N_1 \\ (\beta\ \mathrm{Bool}_2) & \mathsf{if}\ \textit{false}\ \mathsf{then}\ N_1\ \mathsf{else}\ N_2 \mapsto_\beta N_2 \end{array}$$

1.2.2 Substitutions and Alpha Renaming

Substitution is a familiar operation; we apply this operation all the time in mathematical functions without thinking deeply. We just plug numbers in the place of variables. However, this can lead to complications when we translate the operation to our system if we are not careful while defining it. The substitution $M[N/y]$ is read as "in M replace y with N."

This is a valid example of how we can apply a substitution:

$$(x\ (\lambda x.\ x\ y))[z/x] = z\ (\lambda x.\ x\ y)$$

As aforementioned, not every substitution is valid. We must take care that the names of the free variables in N do not coincide with the bound variables in M. To illustrate, the following substitution is invalid because we are capturing the free variable x creating an ambiguity inside the inner lambda:

$$(x\ (\lambda x.\ x\ y))[x/y] \neq x\ (\lambda x.\ x\ x)$$

To avoid this situation, we adopt a convention where the names of bound variables are consistently renamed to be globally unique (the Barendregt convention). This is based on the intuition that the choice of names for bound variables does not matter. Indeed, our arguments will be greatly simplified if we consider lambda-terms as equivalent up to this choice (this is referred to as α-equivalence, denoted with $=_\alpha$).

$$(x\ (\lambda x.\ x\ y))[x/y] =_\alpha (x\ (\lambda z.\ z\ y))[x/y] = (x\ (\lambda z.\ z)\ x)$$

Here, we first rename the inner bound x to z, before proceeding with the correct substitution.

1.2.3 Evaluation Context

We have specified the β-reduction rules, but these are not sufficient by themselves. For example, we can not reduce the following expression:

$$(\mathsf{if}\ (\lambda x.\ x)\ \mathit{true}\ \mathsf{then}\ \mathit{false}\ \mathsf{else}\ \mathit{true})$$

To address this, we introduce the notion of an evaluation context.

$$\mathit{Eval\ Contexts} \ni E ::= \square \mid E\ M \mid \mathsf{if}\ E\ \mathsf{then}\ N_1\ \mathsf{else}\ N_2$$

Informally, a context represents a term with a hole, which we can use to 'focus' a given subterm to be evaluated. This is formalized through the following rule, where $E[M]$ represents the term obtained by filling the hole of E with M.

$$\frac{M \mapsto_\beta M'}{E[M] \mapsto E[M']}$$

Note that the semantics is call-by-name, since ($\beta \to$) allows reducing an application whenever the left-hand-side is a lambda (without first requiring the argument to be reduced).

1.3 Static Semantics

$$\begin{aligned} \mathit{Types} \ni A, B &::= \mathsf{bool} \mid A \to B \\ \mathit{Environment} \ni \Gamma &::= x_1 : A_1, \ldots, x_n : A_n \\ &\quad\ (xs \text{ distinct, unordered}) \\ \mathit{Judgment} &::= \Gamma \vdash M : A \end{aligned}$$

The inference rules can be used to type check terms. They are made by a group of premises (judgments above the inference line) and a conclusion (a judgment below the inference line). Each judgment contains an environment that binds variables to their types. Those variables are unique and the order in the environment does not matter.

$$\frac{}{\Gamma, x : A \vdash x : A}\ \textsc{Var} \qquad \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.\ M : A \to B}\ {\to}\text{I} \qquad \frac{\Gamma \vdash M : A \to B \qquad \Gamma \vdash N : A}{\Gamma \vdash M\ N : B}\ {\to}\text{E}$$

$$\frac{}{\Gamma \vdash \mathit{true} : \mathsf{bool}}\ \textsc{BoolI}_1 \qquad \frac{}{\Gamma \vdash \mathit{false} : \mathsf{bool}}\ \textsc{BoolI}_2$$

$$\frac{\Gamma \vdash M : \mathsf{bool} \qquad \Gamma \vdash N_1 : A \qquad \Gamma \vdash N_2 : ~~A~~B}{\Gamma \vdash \mathsf{if}\ M\ \mathsf{then}\ N_1\ \mathsf{else}\ N_2 : A}\ \textsc{BoolE}$$

2 Type Safety

Our goal is to prove that our language is type safe. We can achieve these results using different methods, such as logical relations as we saw in Amal Ahmed's lectures. One of the most popular techniques is the progress and preservation method. This is a comparatively easy and scalable proof method, which requires us to prove two properties that together entail type soundness. Intuitively, progress states that a well-typed term is not currently stuck, and preservation states that well-typedness is carried forward by reduction. By combining these, we can conclude that well-typed terms never become stuck.

Proving these lemmas is also valuable as it allows us to spot bugs in our language specifications. To illustrate this point, imagine that we have accidentally deleted the red parts of the system definition above (with the exception that A is replaced with B in the premise of BOOLE). As we try to prove our lemmas below, the structure of the logic will reveal these bugs and, to some degree, suggest their solutions.

2.1 Progress

The progress property checks that we didn't forget any reduction rules or value cases. It should catch the most egregious mistakes in our system specification.

Theorem 2.1 (Progress)**.** *If $\cdot \vdash M : A$ is derivable then either:*

1. *M is a value*
2. *$M \to M'$ for some M'*

Proof. By induction on the only object we are given: the derivation of $\cdot \vdash M : A$. There is one case for each inference rule for the typing judgement.

Case $\dfrac{x : A \vdash M : B}{\cdot \vdash \lambda x.\ M : A \to B}\ {\to}\mathrm{I}$

For the arrow introduction case, we already discover one of our bugs. To prove this case of the lemma, we must either prove that $\lambda x.\ M$ is a value, or that it steps to another term. There is no rule for stepping lambdas, and we erroneously removed the lambda case in our definition of values. This case of the proof cannot be completed until we restore lambdas as values. Once we do so, we satisfy the first option of progress and we are done with this case.

Case $\dfrac{\cdot \vdash M : A \to B \qquad \cdot \vdash N : A}{\cdot \vdash M\ N : B}\ {\to}\mathrm{E}$

Suppose we have derivation $\mathcal{D}$ of the first premise and derivation $\mathcal{E}$ of the second. Since our proof is by induction, we have an inductive hypothesis for both $\mathcal{D}$ and $\mathcal{E}$. The inductive hypothesis for $\mathcal{D}$ states that either M is a value or $M \mapsto M'$. Let us proceed by cases on these alternatives:

- M is a value. We know that M is assigned type $A \to B$, so by canonicity, $M = \lambda x.\ M'$ for some x and M'. Now we can prove the second option of progress by showing that $M\ N$, which we know to be $(\lambda x.\ M')\ N$, takes a step. This follows from $(\beta \to)$.
- $M \mapsto M'$. Here we expose another bug. An application cannot be a value, but we also have no rule for stepping applications since we deleted our definition of evaluation contexts. We can recover evaluation contexts one piece at a time, by using the less compact inference rule form. We can add the rule:

$$\frac{M \mapsto M'}{M\ N \mapsto M'\ N}$$

Which is exactly what we need to complete this case by satisfying the second option of progress.

Case $\dfrac{}{\cdot \vdash \mathit{true} : \mathsf{bool}}\ \textsc{BoolI}_1$

true is a value. We are done. The reasoning is the same for $\textsc{BoolI}_2$.

Case $\dfrac{\cdot \vdash M : \mathsf{bool} \qquad \cdot \vdash N_1 : A \qquad \cdot \vdash N_2 : \cancel{A}B}{\cdot \vdash \mathsf{if}\ M\ \mathsf{then}\ N_1\ \mathsf{else}\ N_2 : A}\ \textsc{BoolE}$

Again, we will utilize our inductive hypothesis for the derivation $\mathcal{D}$ of $\Gamma \vdash M : \mathsf{bool}$. Again, we will consider its two cases:

- M is a value. We know that M is assigned type bool, so by canonicity (see below), $M = \mathit{true}$ or $M = \mathit{false}$. If $M = \mathit{true}$, or expression steps by (β Bool_1) and we are done.
  On the other hand, if $M = \mathit{false}$, then we are stuck since a conditional term is not a value, and we have no way to step without our deleted (β Bool_2). Once we restore this rule, we may proceed as above, stepping to N_2 instead of N_1.
- $M \mapsto M'$. Analogously to the case of stepping on the left of an application, we cannot proceed until we restore some of the evaluation context logic. We can add the rule form for stepping in the first position of the conditional:

$$\frac{M \mapsto M'}{\mathsf{if}\ M\ \mathsf{then}\ N_1\ \mathsf{else}\ N_2 \mapsto \mathsf{if}\ M'\ \mathsf{then}\ N_1\ \mathsf{else}\ N_2}$$

  Which allows us to take a step and complete this case. With this second rule for stepping within subexpressions, the entirety of the evaluation context logic has been reproduced. We can think of the BNF form of evaluation contexts that we used originally as a compressed representation of these inference rules which has become standard for its economy.

Case $\dfrac{}{x : A \vdash x : A}\ \textsc{Var}$

A variable cannot step, nor is it a value. With the environment entry deleted in the Var rule, this case cannot be completed. When we restore the requirement that $x : A$, this case does not exist since the theorem supposes a typing derivation in the empty context.

□

Canonicity is a helpful lemma used above in the proof of progress. It states that well-typed, closed values must be of certain forms, called *canonical* forms, depending on their type.

Lemma 2.2 (Canonicity)**.**

If $\cdot \vdash v : \mathsf{bool}$ *then* $v = \mathit{true}$ *or* $v = \mathit{false}$

If $\cdot \vdash v : A \to B$ *then* $v = \lambda x.\ M$ *where* $x : A \vdash M : B$

Proof. By induction on the type assignment derivation. The only forms ever assigned bool are *true*, *false*, applications, and conditionals. Among these, only *true* and *false* are values.

Similarly, the only forms ever assigned an arrow type are abstractions, applications, and conditionals. Among these, only abstractions are values. The rest of the statement is exactly the premise of the $\to$I rule. □

2.2 Preservation

Theorem 2.3 (Preservation)**.** *If* $\Gamma \vdash M : A$ *and* $M \to M'$ *then* $\Gamma \vdash M' : A$

Proof. By Induction on the derivation of $M \mapsto M'$, and of $\Gamma \vdash M : A$:

Case $\mathsf{if}\ \mathit{true}\ \mathsf{then}\ N_1\ \mathsf{else}\ N_2 \mapsto_\beta N_1$

$$\frac{\Gamma \vdash \mathit{true} : \mathsf{bool} \qquad \Gamma \vdash N_1 : A \qquad \Gamma \vdash N_2 : \cancel{A}B}{\Gamma \vdash \mathsf{if}\ \mathit{true}\ \mathsf{then}\ N_1\ \mathsf{else}\ N_2 : A}\ \textsc{BoolE}$$

To prove the conclusion, we have to show that N_1 can be assigned the same type as our outer expression. Fortunately, this is exactly the second premise of the rule, so we are done.

Case $\mathsf{if}\ \mathit{false}\ \mathsf{then}\ N_1\ \mathsf{else}\ N_2 \mapsto_\beta N_2$

$$\frac{\Gamma \vdash \mathit{false} : \mathsf{bool} \qquad \Gamma \vdash N_1 : A \qquad \Gamma \vdash N_2 : \cancel{A}B}{\Gamma \vdash \mathsf{if}\ \mathit{false}\ \mathsf{then}\ N_1\ \mathsf{else}\ N_2 : A}\ \textsc{BoolE}$$

When we try the same thing here, we locate our final bug. We need to prove that $N_2 : A$, which is impossible unless we revert our error. When we do so, the proof is the same as the previous case.

Case $\lambda x.\ M\ N \mapsto_\beta M[N/x]$

$$\frac{\dfrac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.\ M : A \to B}\to\text{I} \qquad \Gamma \vdash N : A}{\Gamma \vdash M\ N : B}\to\text{E}$$

To complete this case, we must show that $M[N/x] : B$ using just the available information form the premises: that $\Gamma,\ x : A \vdash M : B$ and $\Gamma \vdash N : A$. This gives us the exact form of a required substitution lemma, stated below. Such lemmas are typically required for proving preservation for any language.

Case $\dfrac{M \mapsto M'}{M\ N \mapsto M'\ N}$

$$\frac{\Gamma \vdash M : A \to B \qquad \Gamma \vdash N : A}{\Gamma \vdash M\ N : B}\to\text{E}$$

We need to show that $\Gamma \vdash M'\ N : B$. We can do so by applying the same $\to E$ rule, but with the premise $\Gamma \vdash M' : A \to B$ this follows by inductive hypothesis for the step $M \mapsto M'$ and the type assignment $\Gamma \vdash M : A \to B$.

Case $\dfrac{M \mapsto M'}{\mathsf{if}\ M\ \mathsf{then}\ N_1\ \mathsf{else}\ N_2 \mapsto M'\ N_1 N_2}$

$$\frac{\Gamma \vdash M : \mathsf{bool} \qquad \Gamma \vdash N_1 : A \qquad \Gamma \vdash N_2 : A}{\Gamma \vdash \mathsf{if}\ M\ \mathsf{then}\ N_1\ \mathsf{else}\ N_2 : A}\ \textsc{BoolE}$$

We use the same strategy as above. $\Gamma \vdash \mathsf{if}\ M'\ \mathsf{then}\ N_1\ \mathsf{else}\ N_2 : A$ holds by BoolE with the premise $\Gamma \vdash M' : \mathsf{bool}$, which we obtain by induction on $M \mapsto M'$ and $\Gamma \vdash M : \mathsf{bool}$.

□

Lemma 2.4 (Substitution)**.** *If* $\Gamma, x : A \vdash M : B$ *and* $\Gamma \vdash N : A$ *then* $\Gamma \vdash M[N/x] : B$.

The proof is left as an exercise for the reader. Use induction on the derivation of $\Gamma, x : A \vdash M : B$.

Practice Problem: Extend the system and proofs of the lemmas to include recursive types.