OPLSS, Notes for Amal Ahmed's lectures.

Lau

June 26, 2015

Lecture 1, Logical Relations

Simply Typed Lambda Calculus (STLC)

Types:

$$\tau ::= bool \mid \tau \to \tau$$

Terms:

$$e ::= x \mid \text{true} \mid \text{false} \mid \text{if } e \text{ then } e \text{ else } e \mid \lambda x : \tau.\ e \mid e\ e$$

Values:

$$v ::= \text{true} \mid \text{false} \mid \lambda x : \tau.\ e$$

Evaluation contexts:

$$E ::= [] \mid \text{if } E \text{ then } e \text{ else } e \mid E\ e_2 \mid v\ E$$

Evaluations:

$$\text{if true then } e_1 \text{ else } e_1 \mapsto e_1$$
$$\text{if false then } e_1 \text{ else } e_1 \mapsto e_2$$
$$\lambda x : \tau.\ e\ v \mapsto e[v/x]$$

$$\frac{e \mapsto e'}{E[e] \mapsto E[e']}$$

Typing Contexts:

$$\Gamma ::= \dot{\Gamma}, x : \tau$$

Typing rules:

$$\frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau}\ \text{T-Var} \qquad \frac{}{\Gamma \vdash \text{true} : \mathit{bool}}\ \text{T-True}$$

$$\frac{}{\Gamma \vdash \text{false} : \mathit{bool}}\ \text{T-False} \qquad \frac{\Gamma \vdash e : \mathit{bool} \qquad \Gamma \vdash e_1 : \tau \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \text{if } e \text{ then } e_1 \text{ else } e_2}\ \text{T-If}$$

$$\frac{\Gamma, x : \tau \vdash e : \tau_2}{\Gamma \vdash \lambda x : \tau_1.\ e : \tau_1 \to \tau_2}\ \text{T-Abs}$$

$$\frac{\Gamma \vdash e_1 : \tau_2 \to \tau \qquad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash e_1\ e_2 : \tau}\ \text{T-App}$$

Logical Relations

Proof method to establish properties.

- Termination
- Type safety
- Equivalence of programs
  - Correctness of programs
  - Representation independence
  - Parametricity and free theorems, e.g.,

    $$f : \forall\alpha.\ \alpha \to \alpha$$

    Cannot inspect α as it has no idea which type it will be $\rightsquigarrow$ this function must be *id*.

    $$\forall.\ \mathit{int} \to \alpha$$

    A function with this type does not exist (the function would need to return something of type α, but it only has something of type *int* to work with, so it cannot return a value of the proper type).
  - Security-Typed Languages (for Information Flow Control (IFC))
    Example: All types in the code snippet below are labeled. A type can be labeled with either L for *low* or H for *high*. We do not want any flow from variables with a *high* labeled type to a *low* labeled type. The following is an example of *explicit flow* of information:

    ```
    x : int^L
    y : int^H
    x = y     //This assignment is illegal.
    ```

    Further, information may leak through a *side channel*. That is the value denotated by a variable with a *low* labeled type depends on the value of a variable with a *high* labeled type. If this is the case we may not have learned the secret value, but we may have learned some information about it. An example of a side channel:

$$
\begin{array}{l}
\mathtt{x} \ : \ \mathtt{int}^L \\
\mathtt{y} \ : \ \mathtt{int}^H \\
\mathtt{if} \ \ \mathtt{y} > 0 \ \ \mathtt{then} \ \ \mathtt{x} = 0 \ \ \mathtt{else} \ \ \mathtt{x} = 1
\end{array}
$$

How do we then argue a program is secure? Non-interference:

$$
\begin{array}{l}
\vdash P : int^L \times int^H \to int^L \\
P(v_L, v_{H1}) \approx_L P(v_L, v_{H2})
\end{array}
$$

If we run P with the same *low* value and with two different *high* values, then the result of the two runs of the program are equal. That is the *low* result does not depend on *high* values.

Categories of Logical Relations

Logical Predicates	Logical Relations
(Unary)	(Binary)
$P_\tau(e)$	$R\tau(e_1, e_2)$
- One property	- Program Equivalence[1]
- Strong normalization	
- Type safety	

Strong Normalization of STLC

A first try on normalization of STLC

We start with a couple of abbreviations:

$$
\begin{array}{rl}
e \Downarrow v & \stackrel{\text{def}}{=} e \mapsto^* v \\
e \Downarrow & \stackrel{\text{def}}{=} \exists v.\ e \Downarrow v
\end{array}
$$

What we want to prove is:

Theorem (Strong Normalization)**.** *If* $\bullet \vdash e : \tau$ *then* $e \Downarrow$

Proof. ¡ This proof gets stuck and is not complete. It is included to motivate the logical relation. !

By induction on the structure of the typing derivation.
Case $\bullet \vdash \text{true} : bool$, this term has already terminated.
Case $\bullet \vdash \text{false} : bool$, same as for true.
Case $\bullet \vdash \text{if } e \text{ then } e_1 \text{ else } e_2 : \tau$, simple, but requires the use of canonical forms of bool.
Case $\bullet \vdash \lambda x : \tau.\ e : \tau_1 \to \tau_2$, it is a value, it has terminated.

Case $\dfrac{\Gamma \vdash e_1 : \tau_2 \to \tau \qquad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash e_1\ e_2 : \tau}\ \text{T-App}$,

By the induction hypothesis we get $e_1 \Downarrow v_1$ and $e_2 \Downarrow v_2$. By the type of e_1 we conclude $e_1 \Downarrow \lambda x : \tau_2.\ e'$. What we need to show is $e_1\ e_2 \Downarrow$.

$$
\begin{array}{rl}
e_1\ e_2 & \mapsto^* (\lambda x : \tau_2.\ e')\ e_2 \\
& \mapsto^* (\lambda x : \tau_2.\ e')\ v_2 \\
& \mapsto^* e'[v_2/x]
\end{array}
$$

Here we run into an issue as we do not know anything about e'. Our induction hypothesis is not strong enough.[2] □

A logical predicate for strongly normalizing expressions

We want to define a logical predicate, $\mathrm{SN}_\tau(e)$. In English $\mathrm{SN}_\tau(e)$ is the set of expressions of type τ that are strongly normalizing.

In general for a logical predicate, $P_\tau(e)$, we want an expression, e, in accepted by this predicate to satisfy the following properties:

1. $\bullet \vdash e : \tau$[3]
2. the property we wish e to have. In this case it would be: e is strongly normalizing.
3. The condition is preserved by eliminating forms.

With the above in mind we define the strongly normalizing predicate as follows:

$$\mathrm{SN}_{bool}(e) \Leftrightarrow \bullet \vdash e : \mathit{bool} \ \wedge \ e \Downarrow$$
$$\mathrm{SN}_{\tau_1 \to \tau_2}(e) \Leftrightarrow \bullet \vdash e : \tau_1 \to \tau_2 \ \wedge \ e \Downarrow \ \wedge \ (\forall e'.\ \mathrm{SN}_{\tau_1}(e') \Rightarrow \mathrm{SN}_{\tau_2}(e\ e'))$$

It is here important to consider whether the logical predicate is well-founded. $\mathrm{SN}_\tau(e)$ is defined by structural induction over τ, so it is well-founded.

Strongly normalizing using a logical relation

We are now ready to show strong normalization using $\mathrm{SN}_\tau(e)$. The proof is in two steps:

(a) $\bullet \vdash e : \tau \Rightarrow \mathrm{SN}_\tau(e)$

(b) $\mathrm{SN}_\tau(e) \Rightarrow e \Downarrow$

The structure of this proof is common to proofs that use logical relations. We first prove that well-typed terms are in the relation. Then we prove that terms in the relation actually have the property we want to show (in this case strong normalization).

The proof of (b) is by induction on τ.[4]

We could try to prove (a) by induction over $\bullet \vdash e : \tau$, but the case

$$\frac{\Gamma, x : \tau \vdash e : \tau_2}{\Gamma \vdash \lambda x : \tau_1.\ e : \tau_1 \to \tau_2}\ \text{T-Abs}$$

gives issues. What we instead do is to prove a generalization of (a)

Theorem ((b) Generalized)**.** *If* $\Gamma \vdash e : \tau$ *and* $\gamma \models \Gamma$ *then* $\mathrm{SN}_\tau(\gamma(e))$

[2] :(

[3] Note: when we later want to prove type safety we do not want well-typedness to be a property of the predicate.

[4] This should not be difficult, as we baked the property we want into the relation. That was the second property we in general wanted a logical relation to satisfy.

γ is a substitution, $\gamma = \{x_1 \mapsto v_1, \ldots, x_n \mapsto v_n\}$.

In English the theorem reads: If e is well-typed wrt some type τ and we have some substitution that satisfy the typing environment, then is we close of e, then it is in SN.

$\gamma \models \Gamma$ is read "the substitution, γ, satisfies the type environment, *Gamma*," and it is defined as follows:

$$\gamma \models \Gamma \stackrel{\text{def}}{=} \text{dom}(\gamma) = \text{dom}(\Gamma) \ \wedge \ \forall x \in \text{dom}(\Gamma).\ \text{SN}_{\Gamma(x)}(\gamma(x))$$

To prove the generalized theorem we further need two lemmas

Lemma (Substitution Lemma)**.** *If* $\Gamma \vdash e : \tau$ *and* $\gamma \models \Gamma$ *then* $\bullet \vdash \gamma(e) : \tau$

Lemma (SN preserved by forward/backward reduction)**.** *Suppose* $\bullet \vdash e : \tau$ *and* $e \mapsto e'$

1. $\text{SN}_\tau(e')$ *then* $\text{SN}_\tau(e)$
2. $\text{SN}_\tau(e)$ *then* $\text{SN}_\tau(e')$

Proof. Left as an exercise. □

Proof. (Substitution Lemma). Probably also left as an exercise (not proved during the lecture). □

Proof. ($\textcircled{b}$ *Generalized).* Proof by induction on $\Gamma \vdash e : \tau$.
Case $\Gamma \vdash \text{true} : \textit{bool}$,
We have:

$\gamma \models \Gamma$

We need to show:

$\text{SN}_{\textit{bool}}(\gamma(\text{true}))$

If we do the substitution we just need to show $\text{SN}_{\textit{bool}}(\text{true})$ which is true by definition of $\text{SN}_{\textit{bool}}(\text{true})$.
Case $\Gamma \vdash \text{true} : \textit{bool}$, similar to the true case.

Case $\dfrac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau}\ \text{T-Var}$,
We have:

$\gamma \models \Gamma$

We need to show:

$\text{SN}_\tau(\gamma(x))$

This case follows from the definition of $\Gamma \models \gamma$. We know that x is well-typed, so it is in the domain of Γ. From the definition of $\Gamma \models \gamma$ we then get $\text{SN}_{\Gamma(x)}(\gamma(x))$. From well-typedness of x we have $\Gamma(x) = \tau$ which then gives us what we needed to show.
Case if, left as an exercise.

Case $\dfrac{\Gamma \vdash e_1 : \tau_2 \to \tau \qquad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash e_1\ e_2 : \tau}\ \text{T-App}$,
We have:

$\gamma \models \Gamma$

We need to show:

$$\mathrm{SN}_\tau(\gamma(e_1\ e_2)) \equiv \mathrm{SN}_\tau(\gamma(e_1)\ \gamma(e_2))$$

By induction hypothesis we get

$$\mathrm{SN}_{\tau_2 \to \tau}(\gamma(e_1)) \tag{1}$$
$$\mathrm{SN}_{\tau_2}(\gamma(e_2)) \tag{2}$$

If we use the 3rd property of (1), $\forall e'.\ \mathrm{SN}_{\tau_1}(e') \Rightarrow \mathrm{SN}_{\tau_2}(e\ e')$, instantiated with (2) we get $\mathrm{SN}_{\tau_2}(\gamma(e_2))$ which is what we wanted.

Case $\dfrac{\Gamma, x : \tau \vdash e : \tau_2}{\Gamma \vdash \lambda x : \tau_1.\ e : \tau_1 \to \tau_2}\ \textsc{T-Abs}$,
We have:

$$\gamma \models \Gamma$$

We need to show:

$$\mathrm{SN}_{\tau_1 \to \tau_2}(\gamma(\lambda x : \tau_1.\ e)) \equiv \mathrm{SN}_{\tau_1 \to \tau_2}(\lambda x : \tau_1.\ \gamma(e))$$

Our induction hypothesis in this case says:

$$\Gamma, x : \tau_1 \vdash e : \tau_2 \wedge \gamma' \models \Gamma, x : \tau_1 \Rightarrow \mathrm{SN}_{\tau_2}(\gamma'(e))$$

It suffices to show the following three things:

1. $\bullet \vdash \lambda x : \tau_1.\ \gamma(e) : \tau_1 \to \tau_2$
2. $\lambda x : \tau_1.\ \gamma(e) \Downarrow$
3. $\forall e'.\, \mathrm{SN}_{\tau_1}(e') \Rightarrow \mathrm{SN}_{\tau_2}((\lambda x : \tau_1.\ \gamma(e))\ e')$

1 follows from the substitution lemma and then pushing the substitution in under the lambda[5]. 2 is okay as the lambda-abstraction is a value.

It only remains to show 3. To do this we want to somehow apply the induction hypothesis. To do that we need a γ' such that $\gamma' \models \Gamma, x : \tau_1$. As we already have γ and $\gamma \models \Gamma$, so our γ' should probably have have the form $\gamma' = \gamma[x \mapsto v_?]$ for some $v_?$.

Let e' be given and assume $\mathrm{SN}_{\tau_1}(e')$. We then need to show $\mathrm{SN}_{\tau_2}((\lambda x : \tau_1.\ \gamma(e))\ e')$. From $\mathrm{SN}_{\tau_1}(e')$ it follows that $e' \Downarrow v'$. v' is a good candidate for $v_?$ so let $v_? = v'$. From the forward part of the preservation lemma we can further conclude $\mathrm{SN}_{\tau_1}(v')$. We use this to conclude $\gamma[x \mapsto v'] \models \Gamma, x : \tau_1$ which we use with the assumption $\Gamma, x : \tau_1 \vdash e : \tau_2$ to instantiate the induction hypothesis and get $\mathrm{SN}_{\tau_2}(\gamma[x \mapsto v'](e))$.

Now consider the following evaluation:

$$\begin{aligned}(\lambda x : \tau_1.\ \gamma(e))\ e' &\mapsto^* (\lambda x : \tau_1.\ \gamma(e))\ v' \\ &\mapsto \gamma(e)[v'/x] \equiv \gamma[x \mapsto v'](e)\end{aligned}$$

We already concluded that $e' \mapsto^* v'$ which corresponds to the first series of steps. We can then do a β-reduction to take the next step and finally we get something that is equivalent to $\gamma[x \mapsto v'](e)$. That is we have the evaluation

$$(\lambda x : \tau_1.\ \gamma(e))\ e' \mapsto^* \gamma[x \mapsto v'](e)$$

[5]Substitution has not been properly defined here, but one can find a sound definition in Pierce's Types and Programming Languages.

If we use this with $\mathrm{SN}_{\tau_2}(\gamma[x \mapsto v'](e))$ and that is well-typed[6] the backwards part of the SN preservation lemma we get $\mathrm{SN}_{\tau_2}((\lambda x : \tau_1.\ \gamma(e))\ e')$ which is exactly what we needed to show.

□

Exercises

1. Prove SN preserved by forward/backward reduction
2. Prove the substitution lemma
3. Prove the if-case of "(b) Generalized"
4. Extend the language with pairs and do the proves

[6] Follows from $\mathrm{SN}_{\tau_1}(e')$, the assumption $\Gamma \vdash \lambda x : \tau_1.\ e : \tau_1 \to \tau_2$, $\gamma \models \Gamma$, the substitution lemma and the typing rule for lambda-abstractions

Lecture 2, Type Safety

The goal of this lecture was to proof type safety for STLC using logical relations. First we need to consider what type safety is. The classical mantra for type safety is "Well-typed programs do not *go wrong*." It depends on our language and type system what *go wrong* means, but in our case a program has *gone wrong*, if it is stuck[7].

Type Safety for STLC

Theorem (Type Safety for STLC)**.** *If* $\bullet \vdash e : \tau$ *and* $e \mapsto^* e'$ *then* $\mathrm{Val}(e)$ *or* $\exists e'.\ e \mapsto e'$.

Traditionally type safety is proven by two lemmas, progress and preservation.

Lemma (Progress)**.** *If* $\bullet \vdash e : \tau$ *then* $\mathrm{Val}(e)$ *or* $\exists e'.\ e \mapsto e'$.

Progress is normally proved by induction on the typing derivation.

Lemma (Preservation)**.** *If* $\bullet \vdash e : \tau$ *and* $e \mapsto e'$ *then* $\bullet \vdash e' : \tau$.

Preservation is normally proved by induction on the evaluation. Preservation is also known as *subject reduction*. Progress and preservation talks about one step, so to prove type safety we have to do induction on the evaluation. Here we do not want to prove type safety the traditional way, we want to prove it using a logical predicate. We use a logical predicate rather than a logical relation, because type safety is a unary property.

The notation will here further be changed compared to the one from lecture 1. We define the logical predicate in two parts: a value predicate and an expression predicate. We define the value predicate as:

$$\begin{aligned} \mathcal{V}[\![\mathit{bool}]\!] &= \{\mathrm{true}, \mathrm{false}\} \\ \mathcal{V}[\![\tau_1 \to \tau_2]\!] &= \{\lambda x : \tau_1.\ e \mid \forall v \in \mathcal{V}[\![\tau_1]\!].\ e[v/x] \in \mathcal{E}[\![\tau_2]\!]\} \end{aligned}$$

We define the expression predicate as:

$$\mathcal{E}[\![\tau]\!] = \{e \mid \forall e'.\ e \mapsto^* e' \wedge \mathrm{irred}(e') \Rightarrow e' \in \mathcal{V}[\![\tau]\!]$$

[8]The predicate irred is defined as:

$$\mathrm{irred}(e) \stackrel{\mathrm{def}}{=} \nexists e'.\ e \mapsto e'$$

The sets are defined on the structure of the types. $\mathcal{V}[\![\tau_1 \to \tau_2]\!]$ contains $\mathcal{E}[\![\tau_2]\!]$, but $\mathcal{E}[\![\tau_2]\!]$ uses τ_2 directly, so the definition is structurally well-founded.

To prove safety we first define a new predicate, safe:

$$\mathrm{safe}(e) \stackrel{\mathrm{def}}{=} \forall e'.\ e \mapsto^* e' \Rightarrow \mathrm{Val}(e') \vee \exists e''.\ e' \mapsto^* e''$$

We are now ready to prove type safety. Just like we did for strong normalization, we prove type safety in two steps:

ⓐ $\bullet \vdash e : \tau \Rightarrow e \in \mathcal{E}[\![\tau]\!]$

[7]If we consider language-based security for information flow control the notion of *go wrong* would then be that there is an undesired flow of information

[8]Notice that neither $\mathcal{V}[\![\tau]\!]$ nor $\mathcal{E}[\![\tau]\!]$ requires well-typedness. Normally this would be a part of the predicate, but as the goal is to prove type safety we do not want it as a part of the predicate. In fact, if we did include a well-typedness requirement, then we would end up having to prove preservation for some of the proofs to go through.

(b) $e \in \mathcal{E}[\![\tau]\!] \Rightarrow \text{safe}(e)$

Rather than proving (a) directly we prove a more general theorem and get (a) as a corollary. But we are not yet in a position to state the theorem. First we need to define define the interpretation of type environments:

$$\begin{aligned}\mathcal{G}[\![\bullet]\!] &= \{\emptyset\} \\ \mathcal{G}[\![\Gamma, x:\tau]\!] &= \{\gamma[x \mapsto v] \mid \gamma \in \mathcal{G}[\![\Gamma]\!] \wedge v \in \mathcal{V}[\![\tau]\!]\}\end{aligned}$$

Further we need to define semantic type safety:

$$\Gamma \models e : \tau \stackrel{\text{def}}{=} \forall \gamma \in \mathcal{G}[\![\Gamma]\!].\ \gamma(e) \in \mathcal{E}[\![\tau]\!]$$

We can now define our generalized version of (a).

Theorem (Fundemental Property)**.** *If* $\Gamma \vdash e : \tau$ *then* $\Gamma \models e : \tau$

A theorem like this would typically be the first you prove after a logical relation has been defined. The theorem says that every syntactic type safety implies semantic type safety.

We will start by proving the easy part, namely (b).

Proof. *((b)).* Suppose $e \mapsto^* e'$ for some e', then we need to show $\text{Val}(e')$ or $\exists e''.\ e' \mapsto e''$. Now either $\neg\,\text{irred}(e')$ or $\text{irred}(e')$.
Case $\neg\,\text{irred}(e')$, this case follows directly from the definition of irred. $\text{irred}(e')$ is defined as $\nexists e''.\ e' \mapsto e''$ and as the assumption is $\neg\,\text{irred}(e')$ we get $\exists e''.\ e' \mapsto e''$.
Case *irred*(e'), By assumption we have $\bullet \models e : \tau$. As the typing context is empty we choose the empty substitution and get $e \in \mathcal{E}[\![\tau]\!]$. We now use the definition of $e \in \mathcal{E}[\![\tau]\!]$ with what we supposed, $e \mapsto^* e'$, and the case assumption, $\text{irred}(e')$, to conclude $e' \in \mathcal{V}[\![\tau]\!]$. As e' is in the value interpretation of τ we can conclude $\text{Val}(e')$. □

Proof. *(Fundemental Property).* Proof by induction on the typing judgement.
Case $\dfrac{\Gamma, x:\tau \vdash e : \tau_2}{\Gamma \vdash \lambda x:\tau_1.\ e : \tau_1 \to \tau_2}$ T-Abs,
We need to show $\Gamma \models \lambda x:\tau_1.\ e : \tau_1 \to \tau_2$. First suppose $\gamma \in \mathcal{G}[\![\Gamma]\!]$. Thenit suffices to show

$$\gamma(\lambda x:\tau_1.\ e) \in \mathcal{E}[\![\tau_1 \to \tau_2]\!] \equiv (\lambda x:\tau_1.\ \gamma(e)) \in \mathcal{E}[\![\tau_1 \to \tau_2]\!]$$

Now suppose that $\lambda x:\tau_1.\ \gamma(e) \mapsto^* e'$ and $\text{irred}(e')$. We then need to show $e' \in \mathcal{V}[\![\tau_1 \to \tau_2]\!]$. Since $\lambda x:\tau_1.\ \gamma(e)$ is a value it is irreducible, and we can conclude it took no steps. In other words $e' = \lambda x:\tau_1.\ \gamma(e)$. So we need to show $\lambda x:\tau_1.\ \gamma(e) \in \mathcal{V}[\![\tau_1 \to \tau_2]\!]$. Now suppose $v \in \mathcal{V}[\![\tau_1]\!]$ then we need to show $\gamma(e)[v/x] \in \mathcal{E}[\![\tau_2]\!]$.

Now let us keep the above proof goal in mind and consider the induction hypothesis. From the induction hypothesis we have:

$$\Gamma, x:\tau_1 \models e : \tau_2$$

Instantiate this with $\gamma[x \mapsto v]$. We have $\gamma[x \mapsto v] \in \mathcal{G}[\![\Gamma, x:\tau_1]\!]$ because we started by supposing $\gamma \in \mathcal{G}[\![\Gamma]\!]$ and we also had $v \in \mathcal{V}[\![\tau_2]\!]$. The instantiation gives us $\gamma[x \mapsto v]e \in \mathcal{E}[\![\tau_2]\!] \equiv \gamma(e)[v/x] \in \mathcal{E}[\![\tau_2]\!]$. Now recall our proof goal, we now have what we wanted to show.
Case $\dfrac{\Gamma \vdash e_1 : \tau_2 \to \tau \qquad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash e_1\ e_2 : \tau}$ T-App, show this case as an exercise.
The remaining cases were not proved during the lecture. □

Exercises

1. Prove the TApp case of the Fundemental Property