CIS 7000-1 Homework 1

September 13, 2025

1 Analyzing type systems

Each of the following subsections of this problem describes a variant of STLC, including a (potentially) modified grammar, small-step operational semantics, and type system. Each of these variants is independent and you should consider them separately from all others.

For each variant, determine whether type safety holds, where type safety is defined to be the following property.

Definition 1.1 (Stuck)**.** A term e is *stuck* if it is not a value and there does not exists any e' such that $e \rightsquigarrow e'$.

Theorem 1.1 (Type safety)**.** If $\emptyset \vdash e \in \tau$ then for all e', such that $e \rightsquigarrow^* e'$, e' is not stuck.

If the type safety property fails, in a sentence or two, explain in English the source of the error and intuitively how a well-typed program can get stuck.

Regardless of whether type safety holds, state whether the properties of *substitution*, *preservation* and *progress* are true for that system, as stated in the lecture notes. For each false property, give a concrete counter-example and clearly explain why it is a counter-example.

For example, consider the preservation property: If $\emptyset \vdash e \in \tau$ and $e \rightsquigarrow e'$ then $\emptyset \vdash e' \in \tau$.

To give a counter-example, supply a specific e, a specific e' and a specific τ. Then explain why the preservation statement is false for the specific terms you have supplied. To do that, you will show a derivation of $\emptyset \vdash e \in \tau$ to demonstrate e is well-typed and a derivation to show that $e \rightsquigarrow e'$. Then explain why no derivation of $\emptyset \vdash e' \in \tau$ exists (eg: show a partial derivation and explain why you get stuck finishing it off with the rules supplied.)

1.1 Null

Suppose we add a new value called **null**. As in most programming languages, we also add the following typing rule so that **null** has any type:

$$\frac{}{\Gamma \vdash \textbf{null} \in \tau}\ \text{T\_NULL}$$

1. Is STLC with this modification type safe?

   No. This modification is NOT type safe. The error occurs from the fact that **null** can be any type. Thus function applications can get stuck if the function itself is null.

   Consider if we do **null x**. In this case, $\emptyset \vdash \textbf{null} \in \tau_1 \to \tau_2$ and $\emptyset \vdash \mathbf{x} \in \tau_1$. This function application is now stuck and cannot be further simplified.

2. Does *substitution* hold?

   Substition holds.

3. Does *preservation* hold?

   Preservation holds.

4. Does *progress* hold?

   Progress does not hold. The problem occurs from the fact that **null** can be any type. Thus function applications can get stuck if the function itself is null.

   Consider if we do **null x**. In this case, $\emptyset \vdash \mathbf{null} \in \tau_1 \to \tau_2$ and $\emptyset \vdash \mathbf{x} \in \tau_1$. **null x** is not a value (it is a function application) but it does not step either.

1.2 Void

Suppose we add a new type to STLC called **Void**. But that is it. We don't add any new terms, typing rules or small-step reduction rules. This type is called **Void** because it is empty; there are no closed values with this type.

1. Is STLC with this modification type safe?

   Yes, it is still type safe. No terms can actually have the void type.

2. Does *substitution* hold?

   Yes. No terms have the void type, so substitution is vacuously true.

3. Does *preservation* hold?

   Yes. No terms have the void type, so presrvation is vacuously true.

4. Does *progress* hold?

   Yes. No terms have the void type, so progress is vacuously true.

1.3 A mystery language

Suppose we add the following new rules to STLC, where Σ is some fixed map from natural numbers to types. This map is defined for all numbers, but can return any type.

$$\frac{\Sigma(k) = \tau_1 \to \tau_2}{\Gamma \vdash k \in \tau_1 \to \tau_2} \quad \text{T\_ARR\_PTR}$$

$$\frac{}{k\ v \rightsquigarrow (\lambda x.k\ x)\ v} \quad \text{S\_APP\_NAT}$$

1. Is STLC with this modification type safe?
2. Does *substitution* hold?
3. Does *preservation* hold?
4. Does *progress* hold?

1.4 STLC–

Suppose we remove the typing rule for natural numbers, rule T-LIT, from STLC.

1. Is STLC with this modification type safe?
2. Does *substitution* hold?
3. Does *preservation* hold?
4. Does *progress* hold?

1.5 STLC with lists

Suppose we add lists to STLC by adding two new expression forms, $\mathbf{cons}\, e_1\, e_2$ and $\mathbf{nil}$. These new forms are both values.

$$\begin{aligned} \tau &::= \ldots \mid \mathbf{List}\, \tau \\ v &::= \ldots \mid \mathbf{cons}\, e_1\, e_2 \mid \mathbf{nil} \\ e &::= \ldots \mid \mathbf{cons}\, e_1\, e_2 \mid \mathbf{nil} \end{aligned}$$

The typing rules for lists allows us to construct any sort of list out of **nil** and **cons**.

$$\frac{\begin{array}{c}\Gamma \vdash e_1 \in \tau \\ \Gamma \vdash e_2 \in \mathbf{List}\, \tau\end{array}}{\Gamma \vdash \mathbf{cons}\, e_1\, e_2 \in \mathbf{List}\, \tau} \quad \text{T\_CONS}$$

$$\frac{}{\Gamma \vdash \mathbf{nil} \in \mathbf{List}\, \tau} \quad \text{T\_NIL}$$

We also will allow programmers to access the elements of a list through projection. We will reuse the syntax of function application for list projection: if the first argument is some list l and the second argument is some number k, then the application looks up the kth element of the list l:

$$\frac{\begin{array}{c}\Gamma \vdash e_1 \in \mathbf{List}\, \tau \\ \Gamma \vdash e_2 \in \mathbf{Nat}\end{array}}{\Gamma \vdash e_1\ e_2 \in \tau} \quad \text{T\_NTH}$$

$$\frac{}{(\mathbf{cons}\, v_1\, v_2)\ 0 \rightsquigarrow v_1} \quad \text{S\_APP\_ZERO}$$

$$\frac{}{(\mathbf{cons}\, v_1\, v_2)\ (\mathbf{S}\, k) \rightsquigarrow v_2\ k} \quad \text{S\_APP\_SUCC}$$

1. Is STLC with this modification type safe?
2. Does *substitution* hold?
3. Does *preservation* hold?
4. Does *progress* hold?

1.6 Simply-typed function pointers

Suppose we modify STLC to use *function pointers* instead of anonymous functions. To do so, we assume the existence of μ, a fixed map from natural numbers to abstractions and Σ, a map from natural numbers to types.

We also remove the rules that type check and step anonymous functions (as they can no longer appear directly in programs), rule T-ABS and rule S-BETA, and replace them with the following two rules that allow natural numbers to used as function pointers.

$$\frac{\Sigma(k) = \tau_1 \to \tau_2}{\Gamma \vdash k \in \tau_1 \to \tau_2} \quad \text{T\_ARR\_PTR}$$

$$\frac{\mu(k) = \lambda x.e}{k\ v \rightsquigarrow e[v/x]} \quad \text{S\_APP\_PTR}$$

We also assume that all functions stored in the table typecheck according to this type system:

Assumption 1.1 (Table typing)**.** For all k, if $\mu(k) = \lambda x.e$ and $\Sigma(k) = \tau_1 \to \tau_2$ then $x : \tau_1 \vdash e \in \tau_2$.

1. Is STLC with this modification type safe?
2. Does *substitution* hold?
3. Does *preservation* hold?
4. Does *progress* hold?

2 Preservation and Progress proofs

The next part of the homework assignment involves completing the proofs of preservation and progress for two extensions of STLC. If you would like to use Rocq to mechanize these proofs, you can find initial code in the 'homework' directory of the course repository.

2.1 Let binding

Consider adding let expressions to STLC. To do so we extend the grammar, type system, and operational semantics as follows. We add a new expression form that binds the variable x in the body of the let expression e_2.

$$e ::= \ldots \mid \textbf{let}\, x = e_1 \,\textbf{in}\, e_2$$

We add a single new typing rule:

$$\frac{\begin{array}{l}\Gamma \vdash e_1 \in \tau_1 \\ \Gamma, x : \tau_1 \vdash e_2 \in \tau_2\end{array}}{\Gamma \vdash \textbf{let}\, x = e_1 \,\textbf{in}\, e_2 \in \tau_2} \quad \text{T\_LET}$$

and two new evaluation rules:

$$\frac{}{\textbf{let}\, x = v \,\textbf{in}\, e \rightsquigarrow e[v/x]} \quad \text{S\_LETV}$$

$$\frac{e_1 \rightsquigarrow e_1'}{\textbf{let}\, x = e_1 \,\textbf{in}\, e_2 \rightsquigarrow \textbf{let}\, x = e_1' \,\textbf{in}\, e_2} \quad \text{S\_LET\_CONG}$$

1. Extend the preservation proof. This proof is by induction on evaluation steps. That means there will need to be two new cases for rules S-LETV and S-LET-CONG.
2. Extend the progress proof. This proof is by induction on the typing judgement. That means there will be one new case for rule T-LET.

Lemma 2.1 (Preservation)**.** If $\emptyset \vdash e \in \tau$ and $e \rightsquigarrow e'$ then $\emptyset \vdash e' \in \tau$.

Proof. The proof is by induction on the derivation of the reduction. There are cases for each of the rules that could have been used to conclude $e \rightsquigarrow e'$.

- In the case of rule S-LETV, . . .
- In the case of rule S-LET-CONG, . . .

□

Lemma 2.2 (Progress)**.** If $\emptyset \vdash e \in \tau$ then either e is a value or there exists an e' such that $e \rightsquigarrow e'$.

Proof. We prove this lemma by induction in the typing derivation. In the rules where e is already a value, then the proof is trivial. Otherwise, . . . □

2.2 Natural number recursion

Proof preservation and progress for the extension of STLC with a successor and primitive recursion operation as described in the lecture notes.

$$e ::= \ldots \mid \mathbf{succ}\, e \mid \mathbf{nrec}\, e\, \mathbf{of}\, \{0 \Rightarrow e_1;\, \mathbf{S}\, x \Rightarrow e_2\}$$