```
+---------------------------+
| CS 4110 - 11/21/25        |
| Lab 33 - Logical Relations |
+---------------------------+
```

We've seen how logical relations can be used to prove termination for the simply-typed lambda calculus. Today we'll explore how it can be used to prove type safety, and to get the "free theorems" that result from parametricity.

These notes are based on Amal Ahmed's notes from OPLSS (https://www.cs.uoregon.edu/research/summerschool/summer23/_lectures/Logical_Relations_Notes.pdf)

I. Termination for STLC

As a review, here are the definitions for proving termination using a logical relation. We'll use slightly different notation, writing $e \in E[\![\tau]\!]$ instead of $R\tau(e)$ like we did last time. This will set up the notation we use for other logical relations in the rest of this lab.

We define a family of sets, indexed by types.

$$E[\![\text{unit}]\!] := \{ e \mid \vdash e : \text{unit} \wedge e \text{ halts} \}$$

$$E[\![\tau_1 \to \tau_2]\!] := \{ e \mid e \vdash e : \tau_1 \to \tau_2 \wedge e \text{ halts} \wedge \forall e' \in E[\![\tau_1]\!].\ e\ e' \in E[\![\tau_2]\!] \}$$

Then we prove,
- If $e \in E[\![\tau]\!]$ then e halts.
- If $\vdash e : \tau$ then $e \in E[\![\tau]\!]$.

which yields termination for STLC. Of course, these lemmas need to be generalized with a context, like we saw last time. But we won't re-hash that here.

II. Semantic Type Soundness

We can also use logical relations to prove type safety. Unlike the standard recipe using Progress and Preservation lemmas, this is a more semantic argument.

To illustrate, we'll again use STLC with unit:

$$\begin{aligned} e &::= x \mid () \mid \lambda x{:}\tau.e \mid e_1\ e_2 \\ v &::= () \mid \lambda x{:}\tau.e \\ \tau &::= \text{unit} \mid \tau \to \tau \end{aligned}$$

We will call an expression "safe" if all of its normal forms (i.e., expressions that cannot take any more steps) are values:

    safe(e) ≜ { e | ∀ e'. (e –>* e') => ∃e''. e' → e'' ∨ e' ∈ Value }

Note that safety does not imply termination, even though it does hold for this language of course.

We will then define two sets indexed by types.

    V⟦τ⟧ ⊆ Value
    V⟦unit⟧ = { () }
    V⟦τ₁→τ₂⟧ = { λx.e | ∀ v ∈ V⟦τ₁⟧. e[v/x] ∈ E⟦τ₂⟧ }

    E⟦τ⟧ ⊆ Expr
    E⟦τ⟧ = { e | ∀ e'. (e –> e' ∧ e' –/–>) => e' ∈ V⟦τ⟧ }

Next we prove the following lemmas:

    If ⊢ e : τ then e ∈ E⟦τ⟧.
    If e ∈ E⟦τ⟧ then safe(e).

III. Parametricity

Now let's extend our langauge to get System F

    e ::= ... | Λα.e | e [τ]
    τ ::= unit | τ→τ | α | ∀α.τ

Logical relations now depend on a **relation environment**:

    ρ : TypeVar → Type × Type × Rel

where Rel is a relation on values, i.e., a subset of Value×Value.

    V⟦unit⟧_ρ = { ((),()) }
    V⟦α⟧_ρ = ρ(α)
    V⟦τ₁→τ₂⟧_ρ = { (λx.e₁ , λx.e₂) | ∀ (v₁,v₂) ∈ V⟦τ₁⟧_ρ . (e₁[v₁/x], e₂[v₂/x]) ∈ E⟦τ₂⟧_ρ
    V⟦∀α.τ⟧_ρ = { (Λα.e₁ , Λα.e₂) | ∀ τ₁, τ₂, R. (e₁[τ₁/α], e₂[τ₂/α]) ∈ E⟦τ⟧_{ρ[α↦(τ₁, τ₂, R)]}

Note that parametricity ensures that at polymorphic types, the behavior is uniform across *all* relations R.

For brevity, we will elide the definition of the logical relation for E⟦τ⟧. See Ahmed's notes for details. The fundamental lemma says that expressions are related to themselves.

As an example, suppose

```
f : ∀α. α → α
f = Λα. e
```

We know that for every relation R, $(e[\tau_1/\alpha], e[\tau_2/\alpha]) \in E[\![\alpha \to \alpha]\!]_{\rho[\alpha \mapsto R]}$. Unfolding the definition of the function type, we have:

> For every $(v_1, v_2) \in R$. $(e[\tau_1/\alpha][v_1/x], e[\tau_2/\alpha][v_2/x]) \in E[\![\alpha]\!]_{\rho[\alpha \mapsto (\tau_1, \tau_2, R)]}$.

Thus, f must map related inputs to related outputs for every relation R!

Let A be a type and let g be an A -> A function. Let R be the relation corresponding to g:

$$R = \{ (x, g\ x) \mid x \in A \}$$

Using the definitions above, for every (x, g x) in R, it follows that

$$(e[\tau/\alpha]\ x\ ,\ e[\tau/\alpha]\ (g\ x)) \in R$$

By the defintion of R this means

$$e[A/\alpha]\ (g\ x) = g\ (e[A/\alpha]\ x)$$

The only function that commutes with every function g in this way is the identity! So e must be equivalent to the identity function on A. This is one of the "free theorems" from Wadler's paper. And, remarkably, there are free theorems for *every* type in System F!