Lecture Notes on Termination

15-814: Types and Programming Languages
Frank Pfenning

Lecture 13
Tue Oct 21, 2025

1 Introduction

In this lecture we prove termination for a fragment of our call-by-value language using the technique of *logical relations* [Statman, 1985], in this context often called *Tait's method* [1967]. In a way, this lecture represents a preliminary study towards the more important theorems regarding parametricity and representation independence in the following lectures.

The technique we introduce here has proved quite robust with respect to extensions of the language with features that, at first, seem outside its scope. For example, it has been used to prove the soundness of extensions of the Rust language that fall outside its type system. We will mention some of these generalization and applications as we go along. As usual, we concentrate on the essence of the concepts.

2 The Terminating Fragment

There are two fundamental ingredients in our language that allow us to write nonterminating programs. The first is *recursive types*. For example, we can embed the untyped λ-calculus using the type $U = \mu\alpha.\, \alpha \to \alpha$ satisfying the isomorphism $U \cong U \to U$. The other is *fixed point expressions* $\mathsf{fix}\ f.\, e$ which reduce to $[\mathsf{fix}\ f.\, e/f]e$. This immediately leads to nontermination with $\mathsf{fix}\ f.\, f$, which is well-typed (and, in fact, has arbitrary type τ).

Perhaps surprisingly, all other types together, including universal and existential types, constitute a terminating language. In order to keep matters as simple as possible, we will start with treating the language with function types $\tau_1 \to \tau_2$ and bool in the form of Lecture 5, Section 5. The extension to pairs, unit, sums, and lazy records follows the same blueprint and mapped out in Section 6. Universals and existential pose a more significant challenge we postpone to the next lecture.

An important aspect of this development like much of what we have been doing is that the proof is modular in the type structure, so adding more types does not fundamentally change the structure of our arguments. This is a benefit we reap from the careful, principled design of the statics and dynamics of our language.

3 A Proof Attempt

The termination property we ultimately want is the following.

Theorem 1 (Termination, v1) *If $\cdot \vdash e : \tau$ then $e \mapsto^* v$ for some value v.*

If we go into this with the proof techniques we have already seen, the natural attempt would be to prove this by rule induction on the typing derivation.

> **Proof attempt of v1 (doomed to failure).** We try a proof by rule induction on $\cdot \vdash e : \tau$.
>
> **Case:**
>
> $$\frac{x : \tau_1 \vdash e_2 : \tau_2}{\cdot \vdash \lambda x.\, e_2 : \tau_1 \to \tau_2}\ \mathsf{tp/lam}$$
>
> Then $e = \lambda x.\, e_2 \mapsto^0 \lambda x.\, e_2$ which is a value.
>
> That went well! In particular, the restriction to empty contexts in the theorem statement didn't bother us. There is no case for variables (since the context is empty), but matters go awry in the case of applications.
>
> **Case:**
>
> $$\frac{\cdot \vdash e_1 : \tau_2 \to \tau \quad \cdot \vdash e_2 : \tau_2}{\cdot \vdash e_1\, e_2 : \tau}\ \mathsf{tp/app}$$
>
> We can apply the induction hypothesis to the derivations of the premises.
>
> $e_1 \mapsto^* v_1$ for some value v_1 — By ind. hyp.
> $e_2 \mapsto^* v_2$ for some value v_2 — By ind. hyp.
>
> What we need to show is that $e_1\, e_2 \mapsto^* v$ for some v. We can get part of the way there, first reducing e_1 to its value and then reducing e_2 to its value.
>
> $e_1\, e_2 \mapsto^* v_1\, e_2 \mapsto^* v_1\, v_2$ — By properties of $\mapsto^*$
>
> By using the canonical forms theorem we can take one more step.
>
> $v_1 = \lambda x.\, e_1'$ — By canonical forms
> $v_1\, v_2 = (\lambda x.\, e_1') \mapsto [v_2/x]e_1'$ — By rule β
>
> It remains to show that $[v_2/x]e_1' \mapsto^* v$ for some value v. But we can't apply the induction hypothesis to $[v_2/x]e_1'$, nor is this in some way small than e_1 or e_2.

The key issue with this proof attempt is that the conclusion, $e \mapsto^* v$ for some v is too weak. As the proof attempt shows, we need some property of v besides it mere existence so that once we have reduced $e_1\, e_2$ to $v_1\, v_2$ we can conclude that this reduces to a value.

4 Two Logical Predicates

In order to capture what is missing from the proof attempt so far, we first define (for closed expressions e only) a predicate (or set) $[\![\tau]\!]$ of expressions of a given type τ.

$e \in [\![\tau]\!]$ iff $e \mapsto^* v$ and $v \in [\tau]$.

The question is how do we specify the property of the value v? The failed proof attempt suggests that we need to specify that somehow $v_1\, v_2$ can be subject to our induction hypothesis, where both v_1 and v_2 satisfy our value condition. That is:

$v \in [\tau_1 \to \tau_2]$ iff for all $v_1 \in [\tau_1]$ we have $v\, v_1 \in [\![\tau_2]\!]$.

The right-hand side here means that $v\, v_1 \mapsto w$ for some value w and, moreover, $w \in [\tau_2]$ so it seems we should be able to complete our proof in this case now.

What about booleans? They are directly observable, so we just prescribe the possible observation.

$v \in [\mathsf{bool}]$ iff $v = \mathsf{true}$ or $v = \mathsf{false}$.

The principle that is emerging here is:

- For observable types τ, the predicate $[\tau]$ captures the typing of values and thereby the contents of canonical form theorem. In other words, it is based on the constructors for values of type τ.
- For nonobservable types τ, the predicate $[\tau]$ is based on the destructors of the type. This is because we characterize functions by their behavior when applied to arguments, and lazy records by their behavior when projections are applied.

An important observation is that this is a good definition because it is inductive on the structure of τ. In defining $[\![\tau]\!]$ we refer to $[\tau]$, and when $[\tau]$ refers back to $[\![\sigma]\!]$ or $[\sigma]$ it is always on σ that are strictly smaller than τ.

We can now generalize the theorem statement to take the logical predicate into account.

Theorem 2 (Termination, v2) *If* $\cdot \vdash e : \tau$ *then* $e \in [\![\tau]\!]$.

As you can easily check the case for application in a rule induction now goes through, but unfortunately not the case for λ-abstraction.

Before we see whether we are now "done" (that is, have found the needed generalization), let's do a couple of examples to get some experience with these definitions.

$\lambda x.\, x \in [\![\mathsf{bool} \to \mathsf{bool}]\!]$**.** Here is the chain of reasoning:

Consider $\lambda x.\, x \in [\![\mathsf{bool} \to \mathsf{bool}]\!]$
$\lambda x.\, x \mapsto^0 \lambda x.\, x$ so, by definition, it suffices to show $\lambda x.\, x \in [\mathsf{bool} \to \mathsf{bool}]$
Assume $w \in [\mathsf{bool}]$
By definition of $[\to]$, it remains to show: $(\lambda x.\, x)\, w \in [\![\mathsf{bool}]\!]$
By rule, $(\lambda x.\, x)\, w \mapsto^1 w$
So it remains to show: $w \in [\mathsf{bool}]$
This matches our assumption.

$\lambda x.\, \mathsf{if}\ x\ \mathsf{false}\ \mathsf{true} \in [\![\mathsf{bool} \to \mathsf{bool}]\!]$**.** Here is the chain of reasoning:

Consider $\lambda x.\, \mathsf{if}\ x\ \mathsf{false}\ \mathsf{true} \in [\![\mathsf{bool} \to \mathsf{bool}]\!]$
This is a value, so as before we need to show $\lambda x.\, \mathsf{if}\ x\ \mathsf{false}\ \mathsf{true} \in [\mathsf{bool} \to \mathsf{bool}]$
Assume $w \in [\mathsf{bool}]$
By definition of $[\to]$, it remains to show: $(\lambda x.\, \mathsf{if}\ x\ \mathsf{false}\ \mathsf{true})\, w \in [\![\mathsf{bool}]\!]$
$(\lambda x.\, \mathsf{if}\ x\ \mathsf{false}\ \mathsf{true})\, w \mapsto \mathsf{if}\ w\ \mathsf{false}\ \mathsf{true}$
We know $w \in [\mathsf{bool}]$, so we can distinguish its two cases.

Case: $w = \mathsf{true}$.

Then if w false true $\mapsto^1$ false
and false $\in [\![\mathsf{bool}]\!]$ because it is a value and false $\in [\mathsf{bool}]$

Case: $w =$ true.
Then if w false true $\mapsto^1$ true
and true $\in [\![\mathsf{bool}]\!]$ because it is a value and true $\in [\mathsf{bool}]$

Regarding our proposed theorem above, a proof by rule induction still does not work.

Proof attempt of v2 (doomed once again to failure). We try to apply rule induction on the typing derivation.

Case:

$$\frac{x : \tau_1 \vdash e_2 : \tau_2}{\cdot \vdash \lambda x.\, e_2 : \tau_1 \to \tau_2}\ \mathsf{tp/lam}$$

We have that $e = \lambda x.\, e_2 \mapsto^0 \lambda x.\, e_2$ which is a value, so we have to show

$\lambda x.\, e_2 \in [\tau_1 \to \tau_2]$

This is the case if

$(\lambda x.\, e_2)\, v_1 \in [\![\tau_2]\!]$ for all $v_1 \in [\tau_1]$

While we can step the expression to $[v_1/x]e_2$ we cannot apply the induction hypothesis because the context of the premise $x : \tau_1 \vdash e_2 : \tau_2$ is not empty and, anyway, the subject of the judgment is e_2 rather than $[v_1/x]e_2$.

5 Simultaneous Substitutions

In order to solve this second impasse, we need to think about the role of contexts Γ as an expression is evaluated. In a call-by-value language we always evaluated closed expressions, and we accomplish that by always substituting closed values for variables. In the failed proof attempt above, we substitute v_1 for x in e_2. We know that v_1 is not only closed, but that $v_1 \in [\tau_1]$.

More generally, when we type-check an expression $\Gamma \vdash e : \tau$, then if reduction every reaches e, we have substituted for all the variables declared in Γ close up the expression. We write

$$\text{Simultaneous Substitution} \quad \eta \quad ::= \quad v_1/x_1, \ldots, v_n/x_n$$

where all the x_i are distinct. We say η matches Γ (written $\eta \in [\Gamma]$) if $\eta(x) \in [\tau]$ for every $x : \tau$ in Γ. We write $\eta(e)$ for applying the substitution to an expression e, under the presupposition that η substitutes for all the free variables in e. More rigorously, on our small fragment we define it as

$$\begin{array}{lcll}
\eta(e_1\, e_2) & = & \eta(e_1)\, \eta(e_2) & \\
\eta(\lambda x.\, e) & = & \lambda x.\, (\eta, x/x)(e) & \\
\eta(x) & = & v & \text{where } v/x \in \eta \\
\eta(\mathsf{true}) & = & \mathsf{true} & \\
\eta(\mathsf{false}) & = & \mathsf{false} & \\
\eta(\mathsf{if}\ e_1\ e_2\ e_3) & = & \mathsf{if}\ \eta(e_1)\ \eta(e_2)\ \eta(e_3) &
\end{array}$$

It should be easy to see how to extend this to a more complete language.

Now, our final generalization:

Theorem 3 (Termination, v3) *If* $\Gamma \vdash e : \tau$ *and* $\eta \in [\Gamma]$ *then* $\eta(e) \in [\![\tau]\!]$.

Proof: By rule induction on $\Gamma \vdash e : \tau$.

Case:

$$\frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau}\ \mathsf{tp/var}$$

This case is new, because the expression in the type derivation are no longer closed, even if the result of applying the substitution is.

$\eta \in [\Gamma]$	Assumption
$\eta(e) = \eta(x) = v$ for $(v/x) \in \eta$	By definition of substitution
$v \in [\tau]$	By definition of $[\Gamma]$ and $x : \tau \in \Gamma$
$v \mapsto^0 v$	By definition of $\mapsto^0$
$\eta(e) = v \in [\![\tau]\!]$	By definition of $[\![-]\!]$

Case:

$$\frac{\Gamma \vdash e_1 : \tau_2 \to \tau \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash e_1\, e_2 : \tau}\ \mathsf{tp/app}$$

We started thinking about this case. Hopefully it should be relatively straightforward now.

$\eta \in [\Gamma]$	Assumption
To show: $\eta(e) = \eta(e_1\, e_2) = \eta(e_1)\, \eta(e_2) \in [\![\tau_2]\!]$	
$\eta(e_1) \in [\![\tau_2 \to \tau]\!]$	By ind. hyp.
$\eta(e_1) \mapsto^* v_1$ and $v_1 \in [\tau_2 \to \tau]$	By definition
$\eta(e_2) \in [\![\tau_2$	By ind. hyp.
$\eta(e_2) \mapsto^* v_2$ and $v_2 \in [\tau_2]$	By definition
$\eta(e_1)\, \eta(e_2) \mapsto^* v_1\, \eta(e_2) \mapsto^* v_1\, v_2$	By properties of $\mapsto^*$
$v_1\, v_2 \in [\![\tau]\!]$	By definition of $[\to]$

The last step was the critical one. So we have

$$\eta(e_1\, e_2) \mapsto^* v_1\, v_2 \in [\![\tau]\!]$$

Now we need a small lemma called *closure under reverse evaluation* (see Lemma 4 below) to conclude that $\eta(e_1\, e_2) \in [\![\tau]\!]$.

Case:

$$\frac{\Gamma, x : \tau_1 \vdash e_2 : \tau_2}{\Gamma \vdash \lambda x.\, e_2 : \tau_1 \to \tau_2}\ \mathsf{tp/lam}$$

$\eta \in [\Gamma]$	Assumption
$\eta(\lambda x.\, e_2) = \lambda x.\, (\eta, x/x)(e_2) \mapsto^0 \lambda x.\, (eta, x/x)(e_2)$ *value*	By definition of substitution
It remains to show: $\lambda x.\, (eta, x/x)(e_2) \in [\tau_1 \to \tau_2]$	
$w \in [\tau_1]$	Assumption
$(\lambda x.\, (eta, x/x)(e_2))\, w \mapsto (eta, w/x)(e_2)$	By property of substitution
$(\eta, w/x) \in [\Gamma, x : \tau_1]$ since $\eta \in \Gamma$ and $w \in [\tau_1]$	
$(\eta, w/x)(e_2) \in [\![\tau_2]\!]$	By ind. hyp
$\lambda x.\, (\eta, x/x)(e_2) \in [\tau_1 \to \tau_2]$	By definition of $[\to]$

Case:

$$\frac{}{\Gamma \vdash \mathsf{true} : \mathsf{bool}}\ \mathsf{tp/true}$$

Then true *value* and $\mathsf{true} \in [\![\mathsf{bool}]\!]$ so $\mathsf{true} \in [\![\mathsf{bool}]\!]$.

Case: $e = \mathsf{false}$ is like the previous case.

Case:

$$\frac{\Gamma \vdash e_1 : \mathsf{bool} \quad \Gamma \vdash e_2 : \tau \quad \Gamma \vdash e_3 : \tau}{\Gamma \vdash \mathsf{if}\ e_1\ e_2\ e_3 : \tau}\ \mathsf{tp/if}$$

$\eta \in [\Gamma]$	Assumption
$\eta(\mathsf{if}\ e_1\ e_2\ e_3) = \mathsf{if}\ \eta(e_1)\ \eta(e_2)\ \eta(e_3)$	By defn.
$\eta(e_1) \in [\![\mathsf{bool}]\!]$	By ind. hyp.
$\eta(e_1) \mapsto^* v_1$ and $v_1 \in [\mathsf{bool}]$	By defn.
$v_1 = \mathsf{true}$ or $v_2 = \mathsf{false}$	By defn of $[\mathsf{bool}]$

$v_1 = \mathsf{true}$ — First subcase

$\mathsf{if}\ \eta(e_1)\ \eta(e_2)\ \eta(e_3) \mapsto^* \mathsf{if}\ v_1\ \eta(e_2)\ \eta(e_3) = \mathsf{if}\ \mathsf{true}\ \eta(e_2)\ \eta(e_3) \mapsto^1 \eta(e_2)$ By property of evaluation and rule

$\eta(e_2) \in [\![\tau]\!]$	By ind. hyp.
$\mathsf{if}\ \eta(e_1)\ \eta(e_2)\ \eta(e_3) \in [\![\tau]\!]$	By closure under reverse evaluation (Lemma 4)

$v_1 = \mathsf{false}$ — Second subcase

Symmetric to the first subcase

□

Lemma 4 (Closure Under Reverse Evaluation) *If $e \mapsto^* e'$ and $e' \in [\![\tau]\!]$ then $e \in [\![\tau]\!]$.*

Proof: Direct.

$e' \mapsto^* v$ and $v \in [\tau]$ since $e' \in [\![\tau]\!]$	By definition
$e \mapsto^* v$	By transitivity of $\mapsto^*$
$e \in [\![\tau]\!]$	By definition

□

The property and its proof is remarkable in more than one way. Like the proofs of preservation and progress it decomposes into subproofs for each type constructor. The proof also incorporates three theorems we proved separately before: *type preservation*, *progress*, and *canonical forms*.

We spent a lot of effort of showing that all well-typed terms satisfy the logical predicate, but some ill-typed terms do as well. For example,

$$\mathsf{if}\ \mathsf{true}\ \mathsf{false}\ (\lambda x.\, x) \in [\![\mathsf{bool}]\!]$$

That's because

$$\mathsf{if}\ \mathsf{true}\ \mathsf{false}\ (\lambda x.\, x) \mapsto \mathsf{true}$$

and $\mathsf{true} \in [\mathsf{bool}]$.

The fact that our definition allows for some ill-typed terms is actually useful: it allows us to prove property of programs we may wish to write but are not well-typed, but nonetheless well-behaved (see Timany et al. [2024]). This can be applied, for example, when we want to show the soundness of extensions of Rust with modules that cannot be statically checked to be type-correct.

6 Logical Predicates for More Types

As already mentioned, our call-by-value language has some constructs that allow nonterminating computations, so we cannot extend the termination theorem to the full language. But we can consider those that are still terminating and write out the definition of the $[\tau]$ predicate on them. Note that $[\![\tau]\!]$ does not change, since it reduces to evaluation and the $[\tau]$ property.

First, the observable types, generalizing the booleans.

$$\begin{array}{lll} v \in [\tau_1 \times \tau_2] & \text{iff} & v = \langle v_1, v_2 \rangle \text{ and } v_1 \in [\tau_1] \text{ and } v_2 \in [\tau_2] \\ v \in [1] & \text{iff} & v = \langle\,\rangle \\ v \in [\sum_{i \in I}(i : \tau_i)] & \text{iff} & v = k \cdot v' \text{ and } v' \in [\tau_k] \end{array}$$

Recall that under this definition, the booleans are now represented as $\mathsf{bool} = (\mathbf{true} : 1) + (\mathbf{false} : 1)$. The new definition of the logical predicate on booleans coincides with the previous one.

For the types that are not directly observable types we apply the destructors to observe their behavior. Since we have already shown the functions, only the lazy records remain.

$$v \in [\&_{i \in I}(i : \tau_i)] \quad \text{iff} \quad v.i \in [\tau_i] \text{ for all } i \in I$$

For all of these extensions of the logical predicate, it remains defined inductively on the structure of the type. So one might ask why we don't use the logical predicate directly as a definition for validity of programs. The problem is that the quantification over all $w \in [\tau_1]$ in the case of functions

$$v \in [\tau_1 \to \tau_2] \quad \text{iff} \quad v\, w \in [\![\tau_2]\!] \text{ for all } w \in [\tau_1]$$

leads to an undecidable criterion. So instead we use syntactic typing, ideally bidirectional, as our criteria for valid program and then show that all syntactically valid programs are also semantically valid.

If we try to extend the logical predicate further to recursive types in a straightforward way, we lose the inductive nature of the definition.

$$v \in [\mu\alpha.\,\tau] \quad \text{iff} \quad v = \mathsf{fold}\, v' \text{ and } v' \in [[\mu\alpha.\,\tau/\alpha]\tau] \quad \text{not a well-founded definition!}$$

The issue here is that $[\mu\alpha.\,\tau/\alpha]\tau$ may be bigger than the original type $\mu\alpha.\,\tau$ so the definition is no longer inductive. And, indeed, it would not be possible to prove termination because it no longer holds with recursive types unless we impose some restrictions.

Similarly, if we leave fixed points in the language we can no longer prove termination because it doesn't hold, even if you don't have recursive types. In Exercise 3 we ask you to show exactly where the proof fails.

But there is also good news: We can add suitable cases for universal and existential quantification to the definition of the logical predicate and prove termination! The first argument in this style for normalization in an expressive polymorphic λ-calculus was given by Girard [1971]. What does not work is the following natural attempt.

$$v \in \forall\alpha.\,\tau \quad \text{iff} \quad v[\sigma] \in [\![[\sigma/\alpha]\tau]\!] \text{ for all } \sigma \quad \text{not a well-founded definition!}$$

The problem here is once again that the type $\forall\alpha.\,\tau$ itself is a possible σ, so the definition is also circular. But in this case there is an elegant way out, which we discuss in the next lecture.

Exercises

Exercise 1 Show the cases for each of the following types in the termination proof using a logical predicate. Note that it will by necessity consider both constructors and destructors for each kind of type.

(i) Products $\tau_1 \times \tau_2$

(ii) Unit 1

(iii) Sums $\sum_{i \in I}(i : \tau_i)$

(iv) Lazy records $\&_{i \in I}(i : \tau_i)$

Exercise 2 Consider the fully observable types

$$\tau^+ ::= \tau_1^+ \times \tau_2^+ \mid 1 \mid \sum_{i \in I}(i : \tau_i^+)$$

Prove that $\cdot \vdash v : \tau^+$ iff $v \in [\tau^+]$, that is, typing coincides with the logical predicate on fully observable values.

Exercise 3 Consider adding fixed points with their usual typing rule and identify the point in the proof of termination where the argument for the language fragment considered in this lecture breaks down.

References

Jean-Yves Girard. Une extension de l'interpretation de Gödel à l'analyse, et son application à l'élimination des coupures dans l'analyse et la théorie des types. In *Proceedings of the Second Scandinavian Logic Symposium*, pages 63–92, Amsterdam, 1971.

Richard Statman. Logical relations and the typed λ-calculus. *Information and Control*, 65:85–97, 1985.

W. W. Tait. Intensional interpretation of functionals of finite type I. *Journal of Symbolic Logic*, 32: 198–212, 1967.

Amin Timany, Robbert Krebbers, Derek Dreyer, and Lars Birkedal. A logical approach to type soundness. *Journal of the ACM*, 71(6):1–75, 2024.