Termination Casts: A Flexible Approach to Termination with General Recursion

Aaron Stump
Computer Science
The University of Iowa
astump@acm.org

Vilhelm Sjöberg
Computer and Information Science
University of Pennsylvania
vilhelm@cis.upenn.edu

Stephanie Weirich
Computer and Information Science
University of Pennsylvania
sweirich@cis.upenn.edu

Abstract

This paper proposes a type-and-effect system called $\mathtt{T}^{\mathtt{eq}\downarrow}$, which distinguishes terminating terms and total functions from possibly diverging terms and partial functions, for a lambda calculus with general recursion and equality types. The central idea is to include a primitive type-form "Terminates t", expressing that term t is terminating; and then allow terms t to be coerced from possibly diverging to total, using a proof of Terminates t. We call such coercions *termination casts*, and show how to implement terminating recursion using them. For the meta-theory of the system, we describe a translation from $\mathtt{T}^{\mathtt{eq}\downarrow}$ to a logical theory of termination for general recursive, simply typed functions. Every typing judgment of $\mathtt{T}^{\mathtt{eq}\downarrow}$ is translated to a theorem expressing the appropriate termination property of the computational part of the $\mathtt{T}^{\mathtt{eq}\downarrow}$ term.

1 Introduction

Soundly combining general recursion and dependent types is a significant current challenge in the design of dependently typed programming languages. The two main difficulties raised by this combination are (1) type-equivalence checking with dependent types usually depends on term reduction, which may fail to terminate in the presence of general recursion; and (2) under the Curry-Howard isomorphism, non-terminating recursions are interpreted as unsound inductive proofs, and hence we lose soundness of the type system as a logic.

Problem (1) can be addressed simply by bounding the number of steps of reduction that can be performed in a single conversion. This solution may seem ad hoc, but it is less problematic if one works, as we do here, with a primitive notion of propositional equality, and no automatic conversion. Explicit casts with equality proofs are used to change the types of terms, and so with a bound on the number of reduction steps allowed, one may simply chain together a sequence of conversions to accommodate long-running terms in types. There are certainly some issues to be addressed in making such a solution workable in practice, but it is not a fundamental problem.

Problem (2), on the other hand, cannot be so easily dealt with, since we must truly know that a recursive function is total if we are to view it soundly as an inductive proof. One well-known approach to this problem was proposed by Capretta [6]: extend a terminating type theory (that is, one for which we have a sound static analysis for totality, which we use to require all functions to be total) with general recursion via coinductive types. Corecursion is used to model general recursive functions, without losing logical soundness: productive corecursive functions correspond to sound coinductive arguments. The type constructor $(\cdot)^{\nu}$ for possibly diverging computations, together with natural operations on it, is shown to form a monad.

A separate problem related to (2) is extending the flexibility of totality checking for total type theories. It is well-known that structural termination can become awkward for some functions like, for

example, natural-number division, where a recursive call must be made on the result of another function call. For this, methods like type-based termination have been proposed: see Barthe et al. [3] and several subsequent works by those authors; also, Abel [1]. The idea in type-based termination is, roughly, to associate sizes with data, and track sizes statically across function calls. Recursive calls must be on data with smaller size. This method certainly increases the range of functions judged total in their natural presentation. No static termination analysis will be complete, so there will always be programs that type-based termination cannot judge terminating. When such analyses fail, programmers must rewrite their code so that its termination behavior is more apparent to the analysis. What is required is a flexible method for such explicit termination arguments.

1.1 This paper's contribution

This paper proposes a system called $\mathtt{T}^{\mathtt{eq}\downarrow}$ that can be seen as building on both these lines of work. We develop a type-and-effect system where the effect distinguishes total from possibly partial terms. The type assignment judgement $\Gamma \vdash t : T\ \theta$ includes a *termination effect* θ, which can be either $\downarrow$ (called "total"), for terms that are known to terminate, or ? (called "general"), for terms whose termination behavior is unknown.

We can view this approach as building, at least in spirit, on Capretta's approach with the partiality monad, thanks to the close connection between monads and effects, as shown by Wadler and Thiemann [16]. Working out this connection in detail in this particular case is an interesting research challenge. Although this is beyond the scope of the current paper, we give some indication of the connections in Section 6. Of course, there are important differences between the monadic and effectful approaches, most notably that effects are hard-wired into the language definition, while monads are usually programmer-defined. We adopt the effectful approach here, since we are particularly focused on these two kinds of computation, terminating and possibly partial, as fundamental. We thus deem them appropriate for hard-wiring into the language itself. Exploring the tradeoffs more deeply between these two approaches must remain to future work.

Importantly, $\mathtt{T}^{\mathtt{eq}\downarrow}$ provides a flexible approach to termination because the judgment of totality, $\Gamma \vdash t : T\ \downarrow$, is internalized into the type system. The type **Terminates** t expresses termination of term t. The effect of a term can thus be changed from possibly partial to total by casting the term t with a proof of **Terminates** t. These *termination casts* change the type checker's view of the termination behavior of a term, much as a (sound) type cast changes its view of the type of the term. Termination casts are used with the terminating recursion operator: the body of the putatively terminating recursive function is type-checked under the additional explicit assumption that calls with a structurally smaller argument are terminating.

By reifying this basic view of structural termination as an explicit typing assumption, we follow the spirit of type-based termination: our method eliminates the need for a separate structural check (proposed as an important motivation for type-based termination [3]), and gives the programmer even more flexibility in the kind of functions s/he can write. This is because instead of relying on a static analysis to track sizes of datatypes, our approach allows the user (or an automated reasoning system) to perform arbitrarily complex reasoning to show termination of the function. This reasoning can be internal, using termination casts, or completely external: one can write a general recursive function that the type checker can only judge to be possibly partial, and later prove a theorem explicitly showing that the function is terminating. Of course, one could also wish to support what we would see as a hybrid approach, in the style of the PROGRAM tactic in Coq [14], but this is outside the scope of the present paper.

$$
\begin{array}{rcl}
T & ::= & \mathbf{nat} \mid \Pi^{\theta} x{:}T.T' \mid t = t' \mid \mathbf{Terminates}\ t \\
t & ::= & x \mid \lambda x.t \mid t\,t' \mid 0 \mid \mathbf{S}\,t \\
 & \mid & \mathbf{rec}\, f(x) = t \mid \mathbf{C}\ t\ t'\ t'' \\
 & \mid & \mathbf{join} \mid \mathbf{terminates} \mid \mathbf{inv} \mid \mathbf{contra} \mid \mathbf{abort}
\end{array}
$$

Figure 1: Syntax of $\mathtt{T}^{\mathtt{eq}\downarrow}$

1.2 Outline of the development

In Section 2, we first present the syntax, reduction rules and type assignment system for $\mathtt{T}^{\mathtt{eq}\downarrow}$. We follow this explanation with a number of examples of the use of termination casts, in Section 3. Next, in Section 4 we develop our central meta-theoretic result, based on a translation of $\mathtt{T}^{\mathtt{eq}\downarrow}$ typing judgments to judgments about termination of the term in question, formulated in a first-order logical theory of general recursive functions (called W'). This system is similar in spirit to Feferman's theory W (see Chapter 13 of [9]), although with significant syntactic differences, and support for hypothetical reasoning about termination. We show that $\mathtt{T}^{\mathtt{eq}\downarrow}$ is sound with respect to this translation. Also, we find that constructive reasoning suffices for soundness of the translation, so we take W' to be intuitionistic (where an important characteristic of W is that its logic is classical). Then, in Section 5 we develop an annotated version of $\mathtt{T}^{\mathtt{eq}\downarrow}$, where terms are annotated to enable algorithmic type checking. This is the version that would be suitable for implementation. Soundness of annotated $\mathtt{T}^{\mathtt{eq}\downarrow}$ is easily shown by a translation to unannotated $\mathtt{T}^{\mathtt{eq}\downarrow}$.

2 Definition of $\mathtt{T}^{\mathtt{eq}\downarrow}$

The language $\mathtt{T}^{\mathtt{eq}\downarrow}$ is a simple language with natural numbers and dependently-typed recursive functions. The syntax of types T and terms t appears in Figure 1. The variable x is bound in t in the terms $\lambda x.t$ and in T in the type $\Pi^{\theta} x{:}T.T'$. The variables f and x are bound in t in the term $\mathbf{rec}\, f(x) = t$. We use the notation $[t'/x]T$ and $[t'/x]t$ to denote the capture-avoiding substitution of t' for x in types and terms respectively.

We deliberately omit from $\mathtt{T}^{\mathtt{eq}\downarrow}$ many important type-theoretic features which we believe to be orthogonal to the central ideas explored here. A full-fledged type theory based on these ideas would include user-defined inductive types, type polymorphism, perhaps a universe hierarchy, large eliminations, implicit products, and so forth. We focus here just on the core ideas needed for distinguishing total and possibly partial computations with our effect system, and using termination casts to internalize termination.

2.1 Operational semantics

Reduction for $\mathtt{T}^{\mathtt{eq}\downarrow}$ is defined as a call-by-value small-step operational semantics. Figure 2 presents the syntax of values and evaluation contexts as well as the two reduction relations that make up this semantics. Values in $\mathtt{T}^{\mathtt{eq}\downarrow}$ include variables, natural numbers, functions and primitive proof terms for the internalized judgements of equality and termination.

We define the reduction rules with two relations: the primitive β rules, written $t \leadsto_{\beta} t'$ describe reduction when a value is in an active position. This relation is used by the main reduction relation $t \leadsto t'$, which lifts beta reduction through evaluation contexts $\mathscr{C}$ and terminates computation for **abort**, representing finite failure. Other proof forms, including **contra**, are considered values. We cannot, in

$$\begin{array}{rcl} v & ::= & x \mid 0 \mid \mathbf{S}\,v \mid \lambda x.t \mid \mathbf{rec}\,f(x) = t \\ & \mid & \mathbf{join} \mid \mathbf{terminates} \mid \mathbf{inv} \\ C & ::= & [] \mid \mathbf{S}\mathscr{C} \mid \mathscr{C}\,t \mid v\mathscr{C} \mid \mathbf{case}\mathscr{C}\,t\,t \end{array}$$

$\boxed{t \rightsquigarrow_\beta t'}$

$$\frac{}{(\lambda x.t)\,v \rightsquigarrow_\beta [v/x]t} \quad \textsc{Beta\_AppAbs}$$

$$\frac{}{(\mathbf{rec}\,f(x) = t)\,v \rightsquigarrow_\beta [v/x][\mathbf{rec}\,f(x) = t/f]t} \quad \textsc{Beta\_AppRec}$$

$$\frac{}{\mathbf{C}\;0\;t\;t' \rightsquigarrow_\beta t} \quad \textsc{Beta\_CaseZero}$$

$$\frac{}{\mathbf{C}\;(\mathbf{S}\,v)\;t\;t' \rightsquigarrow_\beta t'\,v} \quad \textsc{Beta\_CaseSuc}$$

$\boxed{t \rightsquigarrow t'}$

$$\frac{t \rightsquigarrow_\beta t'}{\mathscr{C}[t] \rightsquigarrow \mathscr{C}[t']} \quad \textsc{Red\_Ctxt}$$

$$\frac{}{\mathscr{C}[\mathbf{abort}] \rightsquigarrow \mathbf{abort}} \quad \textsc{Red\_Abort}$$

Figure 2: Call-by-value small-step operational semantics

fact, obtain a contradiction in the empty context (assuming our theory W' is consistent), but at this point in the development that cannot be shown. So we just include **contra** as a value.

2.2 Type assignment

Figure 3 defines the *type-assignment* system. The judgment $\Gamma \vdash t : T\ \theta$ states that the term t can be assigned type T in the context Γ with effect θ. (The other two judgments, $\Gamma \vdash \mathbf{Ok}$ and $\Gamma \vdash T$, are used by this one to check that contexts and types are well formed.) We define the system such that θ is an approximation of the termination behavior of the system. If we can derive a judgment $\Gamma \vdash t : T\ \downarrow$, then this means that for any assignment of values to the variables in Γ, reduction of t must terminate. (If the context is inconsistent, t might not terminate even if the type system judges it to do so, since an inconsistent context can make unsatisfiable assertions about termination, which may pollute the type system's judgments.) In contrast, the judgment $\Gamma \vdash t : T$? places no restrictions on the termination behavior of t. To simplify the rules, we view θ is as a *capability* on termination behavior [8]. A term with capability ? is allowed to diverge, but terms with capability $\downarrow$ cannot. As a result, any term that typechecks with $\downarrow$ will also typecheck with ?. Thus ? is more permissive than $\downarrow$, and we order them as $\downarrow \leq ?$.

Such reasoning is reflected in the type system. $\mathtt{T}^{\mathtt{eq}\downarrow}$ has a call-by-value operation semantics, so variables stand for values. Therefore, a variable is known to terminate, so we can type variables with any effect in rule T_VAR. This pattern occurs often; all terms that are known to terminate have unconstrained effects in the rules. In this way, we build subeffecting into the type system and do not need an additional rule to coerce total terms to general ones.

As is standard in type-and-effect systems, function types are annotated with a *latent effect*. This effect

$\boxed{\Gamma \vdash T}$

$$\frac{\Gamma \vdash \mathbf{Ok}}{\Gamma \vdash \mathbf{nat}} \text{ K\_NAT} \qquad \frac{\Gamma \vdash T \quad \Gamma, x : T \vdash T'}{\Gamma \vdash \Pi^{\theta} x{:}T.T'} \text{ K\_PI}$$

$$\frac{\Gamma \vdash t : T\ \theta \quad \Gamma \vdash t' : T'\ \theta' \quad \Gamma \vdash T \quad \Gamma \vdash T'}{\Gamma \vdash t = t'} \text{ K\_EQ} \qquad \frac{\Gamma \vdash t : T\ ?}{\Gamma \vdash \mathbf{Terminates}\ t} \text{ K\_TERM}$$

$\boxed{\Gamma \vdash \mathbf{Ok}}$

$$\frac{}{\cdot \vdash \mathbf{Ok}} \text{ OK\_EMPTY} \qquad \frac{\Gamma \vdash \mathbf{Ok} \quad \Gamma \vdash T}{\Gamma, x : T \vdash \mathbf{Ok}} \text{ OK\_CONS}$$

$\boxed{\Gamma \vdash t : T\ \theta}$

$$\frac{t \rightsquigarrow^* t_0 \quad t' \rightsquigarrow^* t_0 \quad \Gamma \vdash t : T\ ? \quad \Gamma \vdash t' : T'\ ?}{\Gamma \vdash \mathbf{join} : t = t'\ \theta} \text{ T\_JOIN} \qquad \frac{\Gamma \vdash t : [t_2/x]T\ \theta \quad \Gamma \vdash t' : t_1 = t_2\ \downarrow}{\Gamma \vdash t : [t_1/x]T\ \theta} \text{ T\_CONV}$$

$$\frac{\Gamma \vdash t : T\ \downarrow}{\Gamma \vdash \mathbf{terminates} : \mathbf{Terminates}\ t\ \theta} \text{ T\_REIFY} \qquad \frac{\Gamma \vdash t : T\ \theta \quad \Gamma \vdash t' : \mathbf{Terminates}\ t\ \downarrow}{\Gamma \vdash t : T\ \theta'} \text{ T\_REFLECT}$$

$$\frac{\Gamma \vdash t : \mathbf{Terminates}\ \mathscr{C}[t']\ \theta}{\Gamma \vdash \mathbf{inv} : \mathbf{Terminates}\ t'\ \theta} \text{ T\_CTXTERM} \qquad \frac{\Gamma(x) = T \quad \Gamma \vdash \mathbf{Ok}}{\Gamma \vdash x : T\ \theta} \text{ T\_VAR}$$

$$\frac{\Gamma, x : T' \vdash t : T\ \theta \quad \Gamma \vdash \Pi^{\theta} x{:}T'.T}{\Gamma \vdash \lambda x.t : \Pi^{\theta} x{:}T'.T\ \theta'} \text{ T\_ABS} \qquad \frac{\Gamma \vdash t : \Pi^{\rho} x{:}T'.T\ \theta \quad \Gamma \vdash t' : T'\ \theta \quad \rho \leq \theta}{\Gamma \vdash t t' : [t'/x]T\ \theta} \text{ T\_APP}$$

$$\frac{\Gamma \vdash \mathbf{Ok}}{\Gamma \vdash 0 : \mathbf{nat}\ \theta} \text{ T\_ZERO} \qquad \frac{\Gamma \vdash t : \mathbf{nat}\ \theta}{\Gamma \vdash \mathbf{S}t : \mathbf{nat}\ \theta} \text{ T\_SUC}$$

$$\frac{\Gamma \vdash t : 0 = \mathbf{S}t'\ \downarrow}{\Gamma \vdash \mathbf{contra} : T\ \theta} \text{ T\_CONTRA} \qquad \frac{\Gamma \vdash \mathbf{Ok}}{\Gamma \vdash \mathbf{abort} : T\ ?} \text{ T\_ABORT}$$

$$\frac{\Gamma, f : \Pi^{?} x{:}T'.T, x : T' \vdash t : T\ ?}{\Gamma \vdash \mathbf{rec}\, f(x) = t : \Pi^{?} x{:}T'.T\ \theta} \text{ T\_REC} \qquad \frac{\Gamma \vdash t : \mathbf{nat}\ \theta \quad \Gamma \vdash t' : [0/x]T\ \theta \quad \Gamma \vdash t'' : \Pi^{\rho} x' : \mathbf{nat}.[\mathbf{S}x'/x]T\ \theta \quad \rho \leq \theta}{\Gamma \vdash \mathbf{C}\ t\ t'\ t'' : [t/x]T\ \theta} \text{ T\_CASENAT}$$

$$\frac{p \notin \mathbf{fv}\, t \quad \Gamma, f : \Pi^{?} x{:}\mathbf{nat}.T, x : \mathbf{nat}, p : \Pi^{\downarrow} x_1 : \mathbf{nat}.\Pi^{\downarrow} q : x = \mathbf{S}x_1.\mathbf{Terminates}\ (f\, x_1) \vdash t : T\ \downarrow}{\Gamma \vdash \mathbf{rec}\, f(x) = t : \Pi^{\downarrow} x{:}\mathbf{nat}.T\ \theta} \text{ T\_RECNAT}$$

Figure 3: Type assignment system

records the termination effect for the body of the function, in rule T_ABS. Likewise, in an application (rule T_APP), the latent effect of the function must be equal or less than the the current termination effect.

$\mathtt{T}^{\mathtt{eq}\downarrow}$ types include two propositions. The type $t = t'$ states that two terms are equal and the type **Terminates** t declares that term t is terminating. The introduction form for the equality proposition (rule T_JOIN) requires both terms to be well typed and evaluate to a common reduct. For flexibility, these terms need not be judged terminating nor have the same type. The elimination form (T_CONV) uses a total proof of equality to convert between equivalent types. Likewise, the introduction form for the **Terminates** t proposition (T_REIFY) requires showing that the term terminates. Analogously, the elimination form (T_REFLECT) uses a total proof of termination to change the effect of t. $\mathtt{T}^{\mathtt{eq}\downarrow}$ also internalizes an admissible property of the judgement—if a term terminates, then the subterm in the active position of the term terminates (T_CTXTERM).

Recursive functions can be typed with either general or total latent effects. In the latter case, the T_RECNAT rule introduces a new hypothesis into the context that may be used to show that the body of the function is total. The assumption $p : \Pi^{\downarrow} x_1 : \mathbf{nat}.\Pi^{\downarrow} q : x = \mathbf{S}\, x_1.\mathbf{Terminates}\ (f\, x_1)$ is an assertion that for any number x_1 that is one less than x, the recursive call $(f\, x_1)$ terminates. Even though the type of f has a ? latent effect, recursive calls on the immediate predecessor can be cast to be total using this assumption.

3 Examples

Natural number addition: internal verification Our first example shows how simple structurally recursive functions can be shown terminating at their definition time using the T_RECNAT rule. We define natural number addition with the following term.

$$\mathit{plus} \stackrel{\text{def}}{=} \lambda\, x_2\,.\,\lambda\, x_1\,.\,(\mathbf{rec}\, f(x_1) = \mathbf{C}\, x_1\ (\lambda\, q\,.\,x_2)\ (\lambda\, z\,.\,\lambda\, q\,.\,\mathbf{S}\,(f\, z\,\mathbf{join}))) x_1\,\mathbf{join}$$

Here, we must use explicit abstractions λq to abstract over equality types, which are then applied to **join**. This standard trick introduces different assumptions of equalities into the context, depending on the case branch. As remarked above, we have deliberately omitted from $\mathtt{T}^{\mathtt{eq}\downarrow}$ a number of features that would improve some of these examples, notably implicit products (as proposed by Miquel [10]) for equality proofs in case-terms.

The type assignment rules verify that plus is a total operation.

$$\cdot \vdash \mathit{plus} : \Pi^{\downarrow} x_2 : \mathbf{nat}.\Pi^{\downarrow} x_1 : \mathbf{nat}.\mathbf{nat}\ \downarrow$$

To see why this is so, first note the type that we assign to the recursion term below, where we make the substitution in the case analysis explicit. The abstraction of q must have a $\downarrow$ latent effect to assign *plus* the type above.

$$x_2 : \mathbf{nat} \vdash \mathbf{rec}\, f(x_1) = \mathbf{C}\, x_1\ (\lambda\, q\,.\,x_2)\ (\lambda\, z\,.\,\lambda\, q\,.\,\mathbf{S}\,(f\, z\,\mathbf{join})) : \Pi^{\downarrow} x_1 : \mathbf{nat}.[x_1\,/\,w]\,(\Pi^{\downarrow} q : x_1 = w.\mathbf{nat})\ \downarrow$$

In this typing derivation, after using rule T_RECNAT,we type check both branches of the case term with the following context.

$$\Gamma \stackrel{\text{def}}{=} x_2 : \mathbf{nat}, x_1 : \mathbf{nat}, f : \Pi^{?} z : \mathbf{nat}.\Pi^{\downarrow} q : z = z.\mathbf{nat}, p : \Pi^{\downarrow} z : \mathbf{nat}.\Pi^{\downarrow} q : x_1 = \mathbf{S}\, z.\mathbf{Terminates}\ (f\, z)$$

In the zero case, we use rules T_ABS and T_VAR to show that the abstraction has the desired total function type.

$$\Gamma \vdash \lambda\, q\,.\,x_2 : (\Pi^{\downarrow} q : x_1 = 0.\mathbf{nat})\ \downarrow$$

In the successor case, we use a termination cast to show that the recursive call $f\,z$ is total. Without this cast, we would be unable to use the latent effect $\downarrow$ in the abstraction of q.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma, z : \mathbf{nat}, q : x_1 = \mathbf{S}z \vdash f\,z : \Pi^{\downarrow} q{:}z = z.\mathbf{nat}\ ?}{\Gamma, z : \mathbf{nat}, q : x_1 = \mathbf{S}z \vdash f\,z : \Pi^{\downarrow} q{:}z = z.\mathbf{nat}\ \downarrow}}{\Gamma, z : \mathbf{nat}, q : x_1 = \mathbf{S}z \vdash f\,z\mathbf{join} : \mathbf{nat}\ \downarrow}}{\Gamma, z : \mathbf{nat}, q : x_1 = \mathbf{S}z \vdash \mathbf{S}(f\,z\mathbf{join}) : \mathbf{nat}\ \downarrow}}{\Gamma, z : \mathbf{nat} \vdash \lambda\, q\,.\mathbf{S}(f\,z\mathbf{join}) : \Pi^{\downarrow} q{:}x_1 = \mathbf{S}z.\mathbf{nat}\ \downarrow}}{\Gamma \vdash \lambda\, z\,.\lambda\, q\,.\mathbf{S}(f\,z\mathbf{join}) : \Pi^{\downarrow} z{:}\mathbf{nat}.\Pi^{\downarrow} q{:}x_1 = \mathbf{S}z.\mathbf{nat}\ \downarrow}$$

$$\dfrac{\Gamma, z : \mathbf{nat}, q : x_1 = \mathbf{S}z \vdash p\,z\,q : \mathbf{Terminates}\ (f\,z)\ \downarrow}{\text{(above)}}$$

Natural number addition: external verification An advantage of this system is that we do not need to prove that plus is total when we define it. We could also define plus using general recursion:

$$plus \stackrel{\text{def}}{=} \lambda\, x_2\,.\mathbf{rec}\, f(x_1) = \mathbf{C}\ x_1\ x_2\ (\lambda\, z\,.\mathbf{S}(f\,z))$$

But note, the best typing derivation will assign a ? latent effect to this function.

$$\cdot \vdash plus : \Pi^{\downarrow} x_2 : \mathbf{nat}.\Pi^{?} x_1 : \mathbf{nat}.\mathbf{nat}\ \downarrow$$

However, all is not lost. We can still prove the following theorem and use it in a termination cast to show that a particular application of *plus* terminates.

$$\cdot \vdash plustotal : \Pi^{\downarrow} x_2 : \mathbf{nat}.\Pi^{\downarrow} x_1 : \mathbf{nat}.\mathbf{Terminates}\ (plus\, x_2\, x_1)\ \downarrow$$

The proof term (below) uses recursion to construct a total witness for this theorem.

$$plustotal \stackrel{\text{def}}{=} \lambda\, x_2\,.\lambda\, x_1\,.(\mathbf{rec}\, f(x_1) = \mathbf{C}\ x_1\ (\lambda\, q\,.\mathbf{terminates})\ (\lambda\, z\,.\lambda\, q\,.\mathbf{terminates}))\, x_1\, \mathbf{join}$$

Here the recursive term has type (again making the substitution explicit)

$$\Pi^{\downarrow} x_1 : \mathbf{nat}.[x_1\,/\,w]\,\Pi^{\downarrow} q{:}x_1 = w.\mathbf{Terminates}\ (plus\, x_2\, w)$$

To understand this proof term, we look at the typing derivation in each case of the case term. Again, let Γ be the context that rule T_RECNAT uses to check the body of the recursive definition, shown below.

$$\begin{array}{rcl} \Gamma \stackrel{\text{def}}{=} plus & : & \Pi^{\downarrow} x_2 : \mathbf{nat}.\Pi^{?} x_1 : \mathbf{nat}.\mathbf{nat}, \\ x_2 & : & \mathbf{nat}, \\ x_1 & : & \mathbf{nat}, \\ f & : & \Pi^{?} z{:}\mathbf{nat}.\Pi^{\downarrow} q{:}z = z.\mathbf{Terminates}\ (plus\, x_2\, z), \\ p & : & \Pi^{\downarrow} z{:}\mathbf{nat}.\Pi^{\downarrow} q{:}x_1 = \mathbf{S}z.\mathbf{Terminates}\ (f\,z) \end{array}$$

Then in the zero case, because $plus\, x_2\, 0$ evaluates to x_2 and variables terminate, we can use rule T_CONV to show that case total.

$$\dfrac{\dfrac{\Gamma, q : x_1 = 0 \vdash \mathbf{terminates} : \mathbf{Terminates}\ x_2\ \downarrow \qquad \Gamma \vdash \mathbf{join} : x_2 = (plus\, x_2\, 0)\ \downarrow}{\Gamma, q : x_1 = 0 \vdash \mathbf{terminates} : \mathbf{Terminates}\ (plus\, x_2\, 0)\ \downarrow}}{\Gamma \vdash \lambda\, q\,.\mathbf{terminates} : \Pi^{\downarrow} p{:}x_1 = 0.\mathbf{Terminates}\ (plus\, x_2\, 0)\ \downarrow}$$

For the successor case, we need to make a recursive call to the theorem to show that the recursive call to the function terminates. Below, let Γ' be the extended environment $\Gamma, z : \mathbf{nat}, q : x_1 = \mathbf{S}z$. Then, the derivation looks like:

$$\dfrac{\dfrac{\dfrac{\Gamma' \vdash plus\, x_2\, z : \mathbf{nat}\ ? \quad \Gamma' \vdash f\, z\, \mathbf{join} : \mathbf{Terminates}\ plus\, x_2\, z\ \downarrow}{\Gamma' \vdash plus\, x_2\, z : \mathbf{nat}\ \downarrow}}{\Gamma' \vdash \mathbf{S}\,(plus\, x_2\, z) : \mathbf{nat}\ \downarrow}}{\Gamma' \vdash \mathbf{terminates} : \mathbf{Terminates}\ (\mathbf{S}\,(plus\, x_2\, z))\ \downarrow} \qquad \Gamma' \vdash \mathbf{join} : (\mathbf{S}\,(plus\, x_2\, z)) = (plus\, x_2\, (\mathbf{S}\, z))\ \downarrow$$

$$\Gamma' \vdash \mathbf{terminates} : \mathbf{Terminates}\ (plus\, x_2\, (\mathbf{S}\, z))\ \downarrow$$

Semantic structural recursion Of course not all functions are syntactically structurally recursive. However, those that are "semantically" structurally recursive can also be internally verified. Consider this recursive definition of multiplication, which has an extraneous call to an identity function in it.

$$mult \stackrel{\text{def}}{=} \lambda\, u\, .\, \mathbf{rec}\, f(x) = \mathbf{C}\, x\, (\lambda\, q\, .\, u)\, (\lambda\, z\, .\, \lambda\, q\, .\, plus\, x\, (f\, (id\, z)\, \mathbf{join}))$$

To show that this recursive function is total, we have to show that the recursive call $(f\,(id\, z))$ terminates. However, all we know is that $f\, z$ terminates. But, we can construct a proof that z is equal to $id\, z$ (as these expressions are joinable) and use it to coerce the termination proof to the one that we need.

$$\dfrac{\Gamma, q : T' \vdash p\, z\, q : \mathbf{Terminates}\ (f\, z)\ \downarrow \qquad \Gamma \vdash \mathbf{join} : z = id\, z\ \downarrow}{\Gamma, q : T' \vdash p\, z\, q : \mathbf{Terminates}\ (f\,(id\, z))\ \downarrow}$$

This example is artificial, but it demonstrates that the reasoning done by the system is semantic instead of syntactic.

Natural number division Finally, we demonstrate a function that requires a course-of-values argument to show termination: natural number division. The general problem is that division calls itself recursively on a number that is smaller, but is not the direct predecessor of the argument. To show that this function terminates, we do structural recursion on an upper bound of the dividend instead of the dividend itself. We could define division as a possibly partial function, and prove externally that it terminates. Here, we explore the use of termination casts for course-of-values recursion, so we define division as a total function. The type we use for division is:

$$div : \Pi^{\downarrow} z : \mathbf{nat}.\Pi^{\downarrow} x : \mathbf{nat}.\Pi^{\downarrow} x' : \mathbf{nat}.\Pi^{\downarrow} u : (lte\, x'\, x) = \mathbf{true}.\mathbf{nat}$$

where z is the divisor, x is the dividend, x' is an upper bound of the dividend, and *lte* is a function that determines if the first number is "less-than-or-equal" the second. We have been parsimonious in omitting a boolean type, so we will use 0 and $\mathbf{S}0$ for **false** and **true**, respectively in the result of *lte*. Therefore, we define

$$lte \stackrel{\text{def}}{=} \mathbf{rec}\, f(x) = \lambda\, u\, .\, \mathbf{C}\, x\, (\mathbf{S}0)\, (\lambda\, x'\, .\, \mathbf{C}\, u\, 0\, (f\, x'))$$

and show

$$\cdot \vdash lte : \Pi^{?} x : \mathbf{nat}.\Pi^{?} x' : \mathbf{nat}.\mathbf{nat}\ \downarrow$$

Note that we are considering *lte* as a possibly partial function; nothing is harmed by not requiring it to be total. We also define cut-off subtraction as a total function *minus* of type $\Pi^{\downarrow} x : \mathbf{nat}.\Pi^{\downarrow} x' : \mathbf{nat}.\mathbf{nat}$ (details omitted). The code for division is then:

$$div \stackrel{\text{def}}{=} \lambda\, z\, .\, ((\mathbf{C}\, z\, (\lambda\, q\, .\, 0)\, (\lambda\, z'\, .\, \lambda\, q\, .\, \mathbf{rec}\, f(x) = \lambda\, x'\, .\, \lambda\, u\, .\, ((\mathbf{C}\, (lte\, (\mathbf{S}\, x)\, z)\, t_1\, (\lambda\, z''\, .\, \lambda\, q'\, .\, 0))\, \mathbf{join}))))\, \mathbf{join})$$

We handle the case of division by 0 up front, obtaining an assumption $q : z = \mathbf{S}z'$ for the **rec**-term. Next, we case split on whether or not x is strictly less than z ($lte\,(\mathbf{S}x)\,z$). If not, we use the term t_1, defined below. If so, we use the term $\lambda\, z''.\lambda\, q'.0$ of type

$$\Pi^{\downarrow} z'' : \mathbf{nat}.\Pi^{\downarrow} q' : lte\,(\mathbf{S}x)\,z = (\mathbf{S}z'').\mathbf{nat}$$

Then the quotient is 0. Term t_1, of type $\Pi^{\downarrow} q' : (\,lte\,(\mathbf{S}x)\,z = 0\,).\mathbf{nat}$, is (with t_2 discussed below):

$$t_1 \stackrel{\text{def}}{=} \lambda\, q'.(\mathbf{S}(f\,(pred\,x)\,(minus\,x\,z)\,t_2\,))$$

In this case, we are decreasing our bound on the dividend by one, and then using a termination cast to show that $f\,(pred\,x)$ is terminating. Here, we define *pred* as just $\lambda\, x.\mathbf{C}\ x\ 0\ \lambda\, x'.x'$. Of course, since this is the unannotated language, the termination cast does not appear in the term itself; for that, see the annotated language (Section 5). To apply the termination cast, we must use the implicit assumption p telling us that f terminates on the predecessor of x. We can prove that $\mathbf{C}\ x\ 0\ \lambda\, x'.x'$ is the predecessor of x in this case, because the following assumptions show that x is non-zero:

- $q : z = (\mathbf{S}z')$
- $q' : lte\,(\mathbf{S}x)\,z = \mathbf{false}$

Intuitively, q' implies that x is greater than or equal to z, which we know is non-zero. The term t_2, omitted here, is a proof that $minus\,x\,z$ is less than or equal to $\mathbf{C}\ x\ 0\ \lambda\, x'.x'$. We can show that **join** will serve for t_2, since the desired equation is provable from the assumptions listed just above (since they imply that both x and z are non-zero).

4 A Logical Semantics for $\mathtt{T}^{\mathtt{eq}\downarrow}$

In this section, we give a semantics for $\mathtt{T}^{\mathtt{eq}\downarrow}$ in terms of a simple constructive logic called W'. This semantics informs our design of $\mathtt{T}^{\mathtt{eq}\downarrow}$ and can potentially be used as part of a consistency proof for $\mathtt{T}^{\mathtt{eq}\downarrow}$. The theory W' is reminiscent of Feferman's theory W (see, for example, Chapter 13 of [9]). W is a classical second-order theory of general recursive functions, classified by class terms which correspond to simple types. W supports quantification over class terms, and quantification over defined individual terms. It is defined in Beeson's Logic of Partial Terms, a logic designed for reasoning about definedness in the presence of partial functions [4]. W includes a relatively weak form of natural-number induction. Indeed, W is conservative over Peano Arithmetic.

4.1 The theory W'

Figure 4 gives the syntax for sorts and formulas for the theory W'. Terms t are just as for (unannotated) $\mathtt{T}^{\mathtt{eq}\downarrow}$, except without **contra**, **inv**, **terminates**, and **join**. Figure 5 gives the proof rules for the theory W'. W' is similar in spirit to Feferman's W, but differs in a number of details. First, W is a two-sorted theory: there is a sort for individual terms, and one for class terms. To express that term t is in class C, theory W uses an atomic formula $t \in C$. Our theory W', in contrast, is a multi-sorted first-order logic, with one sort for every simple type. So W' does not make use of a predicate symbol to express that a term has a sort. We only insist that terms are well-sorted when instantiating quantifiers. Well-formedness of equations does not require well-sortedness of the terms in W' (as also in W). Also, we have no reason at the moment to include non-constructive reasoning in W', so we define it using principles of intuitionistic logic only. As a small matter of convenience, we do not require quantifiers to be instantiated by only terminating

$$\begin{array}{lcl} A & ::= & \mathbf{nat} \mid A \to A' \\ F & ::= & \mathbf{True} \mid \forall x : A.F \mid F \Rightarrow F' \mid F \wedge F' \mid \mathbf{Terminates}\ t \mid t = t' \end{array}$$

Figure 4: Sorts and formulas of W'

terms. This means that for induction principles, we must state explicitly that the terms in question are terminating. Finally, we include a principle PV_COMPIND of computational induction (induction on the structure of a terminating computation); and a principle PV_TERMINV of computational inversion, which allows us to conclude **Terminates** t from **Terminates** $\mathscr{C}[t]$. Interestingly, even without the **inv** construct of $\mathtt{T}^{\mathtt{eq}\downarrow}$, the theorem we prove below would make heavy use of computational inversion. In a classical theory like W, this principle may well be derivable from the other axioms. Here, it does not seem to be.

4.2 Computational translation of terms

Figure 6 defines what we will refer to as the computational translation of $\mathtt{T}^{\mathtt{eq}\downarrow}$ terms (the "C" is for computational). This translation, which is almost trivial, just maps logical terms **join**, **terminates**, **contra**, and **inv** to 0.

4.3 Translation of types

Next, given $\mathtt{T}^{\mathtt{eq}\downarrow}$ type T, we define $[\![T]\!]^C$ and $[\![T]\!]^L$. The "L" is for logical translation. The former is a simple type A, and the latter a predicate F on translated terms (see Figure 4 above). The definition of the interpretations is then given in Figure 7. Note that one can confirm the well-foundedness of this definition by expanding the definition of $[\![T]\!]^L_\theta$, a convenient abbreviation, wherever it is used.

4.4 Examples

Example 1. If we consider the type $\Pi^{\downarrow} x_1 : \mathbf{nat}.\Pi^{\downarrow} x_2 : \mathbf{nat}.\mathbf{nat}$, we will get the following:

$$[\![\Pi^{\downarrow} x_1 : \mathbf{nat}.\Pi^{\downarrow} x_2 : \mathbf{nat}.\mathbf{nat}]\!]^C = \mathbf{nat} \to (\mathbf{nat} \to \mathbf{nat})$$

$$\begin{array}{lcl} [\![\Pi^{\downarrow} x_1 : \mathbf{nat}.\Pi^{\downarrow} x_2 : \mathbf{nat}.\mathbf{nat}]\!]^L\ \mathit{plus} & = & \forall x_1 : \mathbf{nat}.\ \mathbf{Terminates}\ x_1 \wedge \mathbf{True} \Rightarrow \mathbf{Terminates}\ (\mathit{plus}\ x_1) \wedge \\ & & \forall x_2 : \mathbf{nat}.\ \mathbf{Terminates}\ x_2 \wedge \mathbf{True} \Rightarrow \mathbf{Terminates}\ (\mathit{plus}\ x_1\ x_2) \wedge \mathbf{True} \end{array}$$

Example 2. For the type $\Pi^{?} x_1 : \mathbf{nat}.\Pi^{\downarrow} x_2 : \mathbf{nat}.\mathbf{nat}$, we will get the following:

$$[\![\Pi^{?} x_1 : \mathbf{nat}.\Pi^{\downarrow} x_2 : \mathbf{nat}.\mathbf{nat}]\!]^C = \mathbf{nat} \to (\mathbf{nat} \to \mathbf{nat})$$

$$\begin{array}{lcl} [\![\Pi^{?} x_1 : \mathbf{nat}.\Pi^{\downarrow} x_2 : \mathbf{nat}.\mathbf{nat}]\!]^L\ f & = & \forall x_1 : \mathbf{nat}.\ \mathbf{Terminates}\ x_1 \wedge \mathbf{True} \Rightarrow \mathbf{Terminates}\ (\mathit{plus}\ x_1) \Rightarrow \\ & & \forall x_2 : \mathbf{nat}.\ \mathbf{Terminates}\ x_2 \wedge \mathbf{True} \Rightarrow \mathbf{Terminates}\ (\mathit{plus}\ x_1\ x_2) \wedge \mathbf{True} \end{array}$$

Example 3 (higher-order). If we wanted to type a function *iter* which iterates a terminating function x_1, starting from x_2, and does this iteration x_3 times, we might use the type: $\Pi^{\downarrow} x_1 : \Pi^{\downarrow} x : \mathbf{nat}.\mathbf{nat}.\Pi^{\downarrow} x_2 :$

$$\frac{F \in H}{\Sigma; H \vdash F}\ \textsc{Pv\_Assume} \qquad \frac{\Sigma, x : A; H \vdash F \quad x \notin \mathbf{fv} H}{\Sigma; H \vdash \forall x : A.F}\ \textsc{Pv\_Alli}$$

$$\frac{\Sigma; H \vdash \forall x : A.F \quad \Sigma \vdash t : A}{\Sigma; H \vdash [t/x]F}\ \textsc{Pv\_Alle} \qquad \frac{\Sigma; H, F \vdash F'}{\Sigma; H \vdash F \Rightarrow F'}\ \textsc{Pv\_Impi}$$

$$\frac{\Sigma; H \vdash F \Rightarrow F' \quad \Sigma; H \vdash F}{\Sigma; H \vdash F'}\ \textsc{Pv\_Impe} \qquad \frac{\Sigma; H \vdash F \quad \Sigma; H \vdash F'}{\Sigma; H \vdash F \wedge F'}\ \textsc{Pv\_Andi}$$

$$\frac{\Sigma; H \vdash F \wedge F'}{\Sigma; H \vdash F}\ \textsc{Pv\_Ande1} \qquad \frac{\Sigma; H \vdash F \wedge F'}{\Sigma; H \vdash F'}\ \textsc{Pv\_Ande2}$$

$$\frac{}{\Sigma; H \vdash \mathbf{True}}\ \textsc{Pv\_Truei} \qquad \frac{\Sigma; H \vdash 0 = \mathbf{S}t}{\Sigma; H \vdash F}\ \textsc{Pv\_Contra}$$

$$\frac{t \rightsquigarrow^* t'}{\Sigma; H \vdash t = t'}\ \textsc{Pv\_OpSem} \qquad \frac{\Sigma; H \vdash t = t' \quad \Sigma; H \vdash [t/x]F}{\Sigma; H \vdash [t'/x]F}\ \textsc{Pv\_Subst}$$

$$\frac{}{\Sigma; H \vdash \mathbf{Terminates}\ 0}\ \textsc{Pv\_Term0} \qquad \frac{\Sigma; H \vdash \mathbf{Terminates}\ t}{\Sigma; H \vdash \mathbf{Terminates}\ \mathbf{S}t}\ \textsc{Pv\_TermS}$$

$$\frac{}{\Sigma; H \vdash \mathbf{Terminates}\ \lambda x.t}\ \textsc{Pv\_TermAbs} \qquad \frac{}{\Sigma; H \vdash \mathbf{Terminates}\ \mathbf{rec}\, f(x) = t}\ \textsc{Pv\_TermRec}$$

$$\frac{\Sigma; H \vdash \mathbf{Terminates}\ \mathscr{C}[t]}{\Sigma; H \vdash \mathbf{Terminates}\ t}\ \textsc{Pv\_TermInv} \qquad \frac{\Sigma; H \vdash \mathbf{Terminates\ abort}}{\Sigma; H \vdash F}\ \textsc{Pv\_NotTermAbort}$$

$$\frac{\Sigma; H \vdash [0/x]F \quad \Sigma, x' : \mathbf{nat}; H, \mathbf{Terminates}\ x', [x'/x]F \vdash [\mathbf{S}x'/x]F}{\Sigma; H \vdash \forall x : \mathbf{nat}.\mathbf{Terminates}\ x \Rightarrow F}\ \textsc{Pv\_Ind}$$

$$\frac{\Sigma, f : A' \to A; H, \forall x : A'.[f\,x/z]F \vdash \forall x : A'.[t/z]F \quad \Sigma \vdash \mathbf{rec}\, f(x) = t : A' \to A}{\Sigma; H \vdash \forall x : A'.\mathbf{Terminates}\ (\mathbf{rec}\, f(x) = t)\, x \Rightarrow [(\mathbf{rec}\, f(x) = t)\, x/z]F}\ \textsc{Pv\_CompInd}$$

Figure 5: Theory W'

$\mathbf{nat}.\Pi^{\downarrow} x_3 : \mathbf{nat}.\mathbf{nat}$. For this type (call it T for brevity), we will get the following translations:

$$[\![T]\!]^C = (\mathbf{nat} \to \mathbf{nat}) \to (\mathbf{nat} \to (\mathbf{nat} \to \mathbf{nat}))$$

$$\begin{aligned}[\![T]\!]^L\ iter = {} & \forall x_1 : \mathbf{nat} \to \mathbf{nat}.\, \mathbf{Terminates}\ x_1 \wedge (\forall x : \mathbf{nat}.\mathbf{Terminates}\ x \wedge \mathbf{True} \Rightarrow \mathbf{Terminates}\ (x_1\ x)) \Rightarrow \\ & \qquad \mathbf{Terminates}\ (iter\ x_1) \wedge \\ & \forall x_2 : \mathbf{nat}.\, \mathbf{Terminates}\ x_2 \wedge \mathbf{True} \Rightarrow \mathbf{Terminates}\ (iter\ x_1\ x_2) \wedge \\ & \forall x_3 : \mathbf{nat}.\, \mathbf{Terminates}\ x_3 \wedge \mathbf{True} \Rightarrow \mathbf{Terminates}\ (iter\ x_1\ x_2\ x_3) \wedge \mathbf{True}\end{aligned}$$

Notice that in this case, the logical interpretation $[\![T]\!]^L$ includes a hypothesis that the function x_1 is

$$
\begin{array}{rcl}
[\![x]\!]^C & = & x \\
[\![t\,t']\!]^C & = & [\![t]\!]^C\ [\![t']\!]^C \\
[\![\lambda\, x\,.\,t]\!]^C & = & \lambda x.\,[\![t]\!]^C \\
[\![0]\!]^C & = & 0 \\
[\![\mathbf{S}\,t]\!]^C & = & \mathbf{S}\ [\![t]\!]^C \\
[\![\mathbf{join}]\!]^C & = & 0 \\
[\![\mathbf{terminates}]\!]^C & = & 0 \\
[\![\mathbf{inv}]\!]^C & = & 0 \\
[\![\mathbf{contra}]\!]^C & = & 0 \\
[\![\mathbf{abort}]\!]^C & = & \mathbf{abort} \\
[\![\mathbf{rec}\,f(x) = t]\!]^C & = & \mathbf{rec}\,f(x).[\![t]\!]^C \\
[\![\mathbf{C}\ t\ t'\ t'']\!]^C & = & \mathbf{C}\ [\![t]\!]^C\ [\![t']\!]^C\ [\![t'']\!]^C
\end{array}
$$

Figure 6: Computational translation of terms

$$
\begin{array}{rcl}
[\![\mathbf{nat}]\!]^C & = & \mathbf{nat} \\
[\![\Pi^\theta x{:}T.T']\!]^C & = & [\![T]\!] \to [\![T']\!] \\
[\![t = t']\!]^C & = & \mathbf{nat} \\
[\![\mathbf{Terminates}\ t]\!]^C & = & \mathbf{nat}
\end{array}
$$

$$
\begin{array}{rcl}
[\![\mathbf{nat}]\!]^L\ t & = & \mathbf{True} \\
[\![\Pi^\theta x{:}T.T']\!]^L\ t & = & \forall x : [\![T]\!]^C.[\![T]\!]^L_\downarrow\ x\ \Rightarrow [\![T]\!]^L_\theta\ (t\ x) \\
[\![t_1 = t_2]\!]^L\ t & = & [\![t_1]\!]^C = [\![t_2]\!]^C \\
[\![\mathbf{Terminates}\ t']\!]^L\ t & = & \mathbf{Terminates}\ [\![t']\!]^C
\end{array}
$$

$$
\begin{array}{rcl}
[\![T]\!]^L_\downarrow & = & \mathbf{Terminates}\ [\![t]\!]^C\ \wedge\ [\![T]\!]^L[\![t]\!]^C \\
[\![T]\!]^L_? & = & \mathbf{Terminates}\ [\![t]\!]^C\ \Rightarrow\ [\![T]\!]^L[\![t]\!]^C
\end{array}
$$

Figure 7: Interpretation of types

terminating. This corresponds to the fact that x_1 has type $\Pi^{\downarrow} x{:}\mathbf{nat}.\mathbf{nat}$ in the original $\mathtt{T}^{\mathtt{eq}\downarrow}$ type.

Example 4 (logical). Suppose we have defined *plus* in a standard way by recursion on its first argument, and we wish to prove that for all x, $plus\ x\ 0 = x$. The type we would use is $\Pi \downarrow x : \mathbf{nat}.\, plus\ x\ 0 = x$. The translations of this type are:

$$
\begin{array}{rcl}
[\![\Pi \downarrow x : \mathbf{nat}.\, plus\ x\ 0 = x]\!]^C & = & \mathbf{nat} \to \mathbf{nat} \\
[\![\Pi \downarrow x : \mathbf{nat}.\, plus\ x\ 0 = x]\!]^L\ p & = & \forall x : \mathbf{nat}.\, \mathbf{Terminates}\ x \wedge \mathbf{True} \Rightarrow \mathbf{Terminates}\ (p\ x)\ \wedge\ plus\ x\ 0 = x \\
 & & \wedge\ \mathbf{True}
\end{array}
$$

$$\begin{array}{rcl} [\![\cdot]\!]^C & = & [\![\cdot]\!] \\ [\![\Gamma, x : T]\!]^C & = & [\![\Gamma]\!], x : [\![T]\!]^C \end{array}$$

$$\begin{array}{rcl} [\![\cdot]\!]^L & = & [\![\cdot]\!] \\ [\![\Gamma, x : T]\!]^L & = & [\![\Gamma]\!], [\![T]\!]^L_{\downarrow}\, x \end{array}$$

Figure 8: Interpretation of contexts

So the logical translation of the type contains a formulation of the original property about *plus*.

4.5 Translation of contexts

Figure 8 gives a similar 2-part translation of typing contexts. The translation $[\![\cdot]\!]^C$ produces a simple-typing context Σ, while the translation $[\![\cdot]\!]^L$ produces a logical context Δ, which asserts, for each variable x, that x terminates and has the property given by the $[\![\cdot]\!]^L$ translation of its type.

4.6 Translation of typing judgments

We are now in a position to state the main theorems of this paper. The proofs are given in the Appendix. Theorem 2 shows that the logical translation of types is sound: the property expressed by $[\![T]\!]^L_\theta$ can indeed be proved to hold for the translation $[\![t]\!]^C$ of terms of type T.

Theorem 1 (Soundness of Computational Translation)**.** *If* $\Gamma \vdash t : T\ \theta$*, then* $[\![\Gamma]\!]^C \vdash [\![t]\!]^C : [\![T]\!]^C$.

Theorem 2 (Soundness of Logical Translation)**.** *If* $\Gamma \vdash t : T\ \theta$*, then* $[\![\Gamma]\!]^C; [\![\Gamma]\!]^L \vdash [\![T]\!]^L_\theta\ [\![t]\!]^C$.

5 Annotated $\mathtt{T}^{\mathtt{eq}\downarrow}$

In $\mathtt{T}^{\mathtt{eq}\downarrow}$, type inference is not algorithmic. Given a context Γ, a term t and effect θ, it is not clear how to determine if there is some T such that $\Gamma \vdash t : T\ \theta$ holds. This is not a fault of the effect system–indeed, the role of effects is syntax-directed. The problem lies with the terms that do not contain enough information to indicate how to construct a typing derivation.

Fortunately, it is straightforward to produce an annotated version of $\mathtt{T}^{\mathtt{eq}\downarrow}$ where the type checking algorithm is fully determined. Below we give the syntax of the annotated terms. The full typing rules for the annotated system appear in Appendix C. The judgment form is $\Gamma \Vdash a : T\ \theta$, where algorithmically, Γ, a, and θ are inputs to the type checker, and T is the output.

$$\begin{array}{rcl} a & ::= & y \mid a\,a' \mid \lambda\, y : T.a \mid 0 \mid \mathbf{S}\,a \\ & \mid & \mathbf{rec_{nat}}\ g(y\,p) : T = a \mid \mathbf{rec}\ g(y : T) : T' = a \mid \mathbf{C}\ x.T\ a\ a'\ a'' \\ & \mid & \mathbf{join}\,a\,a' \mid \mathbf{conv}\,x.T\,a'\,a \mid \mathbf{terminates}\,a \mid \mathbf{reflect}\,a\,a' \mid \mathbf{inv}\,a\,a' \mid \mathbf{contra}\,T\,a \mid \mathbf{abort}\,T \end{array}$$

Most annotated term forms have direct correspondence to the unannotated terms. Notably, there are two different forms of recursion, based on which typing rule should be used. Furthermore, the syntax includes terms ($\mathbf{conv}\,x.T\,a'\,a$ and $\mathbf{reflect}\,a\,a'$) that mark where type conversions and termination casts should occur—these are implicit in the unannotated system.

$$
\begin{aligned}
(\Gamma \vdash \mathbf{nat})^\dagger &\equiv \mathtt{nat} \\
(\Gamma \vdash T \xrightarrow{\downarrow} T')^\dagger &\equiv (T)^\dagger \to (T')^\dagger \\
(\Gamma \vdash T \xrightarrow{?} T')^\dagger &\equiv (T)^\dagger \to \mathtt{Partial}\ (T')^\dagger \\
(\Gamma \vdash \mathbf{Terminates}\ t)^\dagger &\equiv \mathtt{Terminates}\ (\Gamma \vdash t : T\ \theta)^*
\end{aligned}
$$

Figure 9: The type translation $(\cdot)^\dagger$ of a simply-typed fragment of $\mathtt{T}^{\mathtt{eq}\downarrow}$.

The annotated system uses the same types as the type assignment system. These types include unannotated terms, so the annotated system must erase annotations (using the erasure operator $|a|$) when terms appear in types. One place where this operator is needed is in the application rule:

$$
\frac{\Gamma \Vdash a : \Pi^\rho x{:}T'.T\ \theta \quad \Gamma \Vdash a' : T'\ \theta \quad \rho \le \theta}{\Gamma \Vdash a\,a' : [|a'|/x]T\ \theta}\ \text{AT\_APP}
$$

Simple comparison of the typing rules of the two systems in a straightforward inductive proof shows that the annotated system is sound with respect to the unannotated system.

Proposition 3 (Soundness of annotated system)**.** *If* $\Gamma \Vdash a : T\ \theta$ *then* $\Gamma \vdash |a| : T\ \theta$.

Note however, that although type inference is algorithmic, it is only decidable in the annotated system if there is some cut-off in normalization in the join rule, shown below.

$$
\frac{\begin{array}{c}|a| \rightsquigarrow^N t \quad |a'| \rightsquigarrow^N t \\ \Gamma \Vdash a : T\ ? \quad \Gamma \Vdash a' : T'\ ?\end{array}}{\Gamma \Vdash \mathbf{join}\,a\,a' : |a| = |a'|\ \theta}\ \text{AT\_JOIN}
$$

Even if we were to require a and a' to have the total effect in this rule, this restriction would not ensure decidability. An inconsistent context could type a looping term with a total effect.

6 Comparison with Capretta's Partiality monad

For simply-typed languages, Wadler and Thiemann [16] showed that effects systems and monadic type systems are equivalent; more precisely they describe how to translate programs written in a language with effects into programs written with monads in a way which preserves typing judgements and operational semantics. Since Capretta [6] gave an account of general recursion in terms of a monadic type constructor, it is natural to ask if it is possible to adapt this translation for $\mathtt{T}^{\mathtt{eq}\downarrow}$. This would give a shallow embedding of $\mathtt{T}^{\mathtt{eq}\downarrow}$ into Coq + Capretta's Partiality library.

If we restrict our attention to a fragment of $\mathtt{T}^{\mathtt{eq}\downarrow}$ which omits the dependently typed features, that is equality types and dependent function types, this approach seems essentially feasible. Figure 6 indicates what the type translation would look like. The unannotated terms t on their own do not contain enough information (for instance, we need to know whether to translate **rec** terms using the general recursion combinator `Y` or using structural recursion), so the translation is from entire typing derivations. The corresponding term translation $(\Gamma \vdash t : T\ \theta^*)$ has not been worked out in detail and is omitted.

The most interesting question when giving such a translation is how to treat the **Terminates** t type and the **recnat** recursor. However, it is not hard to write Coq terms of the following types

```
Terminates : forall (A : Set), Partial A -> Set
reflect : forall (A : Set) (x : Partial A), Terminates x -> A
REC_nat_total
     : forall phi : Set,
       (forall (f : nat -> Partial phi) (x : nat),
        (forall y : nat, x = S y -> Terminates (f y)) -> phi) ->
       nat -> phi
```

which have the right types to interpret these. Here the `Terminates` predicate inductively defines that a partial value leads toa value in finite time, the `reflect` function extracts the value from such a terminating value, and the `REC_nat_total` function works by calling the functional F with a function f that checks if its argument y is equal to $x-1$ and diverges if it is not.

However, if we instead consider the full, dependently typed $\mathtt{T}^{\mathtt{eq}\downarrow}$ language, two difficulties arise. First, we must decide how to interpret the equality type. In $\mathtt{T}^{\mathtt{eq}\downarrow}$, general recursive function calls are equal to their unfoldings, and equal can be substituted for equal in types. Functions defined using Capretta's `Y` operator are coinductive objects, and will not in general be equal to their unfoldings under Coq's propositional equality $=$. Instead we can only prove that $Y\ (\lambda f\ x.a)\ v \stackrel{v}{=} [v/x, Y\ (\lambda f\ x.a)/f]a$, where $\stackrel{v}{=}$ is a coinductively defined relation which identifies terms that compute the same value, or terms that diverge. We can not use $\stackrel{v}{=}$ to do arbitrary rewrites in types.

Secondly, in $\mathtt{T}^{\mathtt{eq}\downarrow}$ functions can have a return type which depends on a potentially nonterminating argument. It is not clear how to represent this in a monadic framework. For example, if we imagine a version of $\mathtt{T}^{\mathtt{eq}\downarrow}$ extended with option types, and suppose we are given a decision procedure for equality of **nat**s and a partial function which computes the minimum zero of a function:

$$\begin{array}{l} \mathit{eqDec} : \Pi^{\downarrow} x{:}\mathbf{nat}.\Pi^{\downarrow} x'{:}\mathbf{nat}.\mathbf{Maybe}\,(x = x') \\ \mathit{minZero} : \Pi^{?} f{:}(\Pi^{\downarrow} x{:}\mathbf{nat}.\mathbf{nat}).\mathbf{nat} \end{array}$$

Then we can easily compose these to make a function to test if two functions have the same least zero:

$$\begin{array}{l} \lambda f\,.\,\lambda f'\,.\,\mathit{eqDec}\,(\mathit{minZero} f)\,(\mathit{minZero} f') \\ \quad : \Pi^{\downarrow} f{:}(\Pi^{\downarrow} x{:}\mathbf{nat}.\mathbf{nat}).\Pi^{?} f'{:}(\Pi^{\downarrow} x{:}\mathbf{nat}.\mathbf{nat}).\mathbf{Maybe}\,(\mathit{minZero} f = \mathit{minZero} f') \end{array}$$

However the naive translation of this into monadic form,

$$\lambda f.\lambda f'.(\mathit{minZero} f) >\!\!>\!\!= (\lambda m.(\mathit{minZero} f') >\!\!>\!\!= (\lambda m'.\mathbf{return}\ (\mathit{eqDec}\ m\ m'))),$$

is not well typed, since the monadic bind $>\!\!>\!\!= : \forall A\ B.(A \to B^{v}) \to A^{v} \to B$ does not have a way to propagate the type dependency.

7 Other Related Work

Another approach, not depending on coinductive types, is explored by Capretta and Bove, who define a special-purpose accessibility predicate for each general recursive function, and then define the function by structural recursion on the proof of accessibility for the function's input [5]. ATS and GURU both separate the domains of proofs and programs, and can thus allow general recursion without endangering logical soundness [15, 7]. Systems like Cayenne [2], ΩMEGA [13]. and CONCOQTION [11] support dependent types and general recursion, but do not seek to identify a fragment of the term language which is sound as a proof system (although CONCOQTION uses COQ proofs for reasoning about type indices).

8 Conclusion

$\mathtt{T}^{\mathtt{eq}\downarrow}$ combines equality types and general recursion, using an effect system to distinguish total from possibly partial terms. Termination casts are used to change the type system's view of the termination behavior of a term. Like other casts, termination casts have no computational relevance and are erased in passing from the annotated to unannotated type system. We have given a logical semantics for $\mathtt{T}^{\mathtt{eq}\downarrow}$ in terms of a multi-sorted first-order theory of general-recursive functions. Future work includes further meta-theory, including type soundness for $\mathtt{T}^{\mathtt{eq}\downarrow}$ and further analysis of the proposed theory W'; as well as incorporation of other typing features, in particular polymorphism and large eliminations.

Acknowledgments. All syntax definitions in this paper were typeset and type-checked with the help of the very useful OTT tool [12]. Thanks also to other members of the TRELLYS project, especially Tim Sheard, for helpful conversations on these ideas. This work was partially supported by the the U.S. National Science Foundation under grants 0910510 and 0910786.

References

[1] Andreas Abel. *A Polymorphic Lambda-Calculus with Sized Higher-Order Types*. PhD thesis, Ludwig-Maximilians-Universität München, 2006.

[2] Lennart Augustsson. Cayenne–a language with dependent types. In *Proc. 3rd ACM International Conference on Functional Programming (ICFP)*, pages 239–250, 1998.

[3] G. Barthe, M. Frade, E. Giménez, L. Pinto, and T. Uustalu. Type-based termination of recursive definitions. *Mathematical Structures in Computer Science*, 14(1):97–141, 2004.

[4] M. Beeson. *Foundations of Constructive Mathematics: Metamathematical Studies*. Springer, 1985.

[5] A. Bove and V. Capretta. Modelling general recursion in type theory. *Mathematical Structures in Computer Science*, 15:671–708, February 2005. Cambridge University Press.

[6] V. Capretta. General Recursion via Coinductive Types. *Logical Methods in Computer Science*, 1(2):1–28, 2005.

[7] C. Chen and H. Xi. Combining Programming with Theorem Proving. In *Proceedings of the 10th International Conference on Functional Programming (ICFP05)*, Tallinn, Estonia, September 2005.

[8] K. Crary, D. Walker, and G. Morrisett. Typed Memory Management in a Calculus of Capabilities. In *POPL '99: Proceedings of the 26th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 262–275. ACM, 1999.

[9] S. Feferman. *In the Light of Logic*. Oxford University Press, 1998.

[10] A. Miquel. The Implicit Calculus of Constructions. In *Typed Lambda Calculi and Applications*, pages 344–359, 2001.

[11] E Pasalic, J. Siek, W. Taha, and S. Fogarty. Concoqtion: Indexed Types Now! In G. Ramalingam and E. Visser, editors, *ACM SIGPLAN 2007 Workshop on Partial Evaluation and Program Manipulation*, 2007.

[12] P. Sewell, F. Nardelli, S. Owens, G. Peskine, T. Ridge, S. Sarkar, and R. Strnisa. Ott: Effective tool support for the working semanticist. *J. Funct. Program.*, 20(1):71–122, 2010.

[13] T. Sheard. Type-Level Computation Using Narrowing in Ωmega. In *Programming Languages meets Program Verification*, 2006.

[14] M. Sozeau. Subset Coercions in Coq. In T. Altenkirch and C. McBride, editors, *Types for Proofs and Programs, International Workshop, TYPES 2006, Nottingham, UK, April 18-21, 2006, Revised Selected Papers*, pages 237–252, 2006.

[15] A. Stump, M. Deters, A. Petcher, T. Schiller, and T. Simpson. Verified Programming in Guru. In T. Altenkirch and T. Millstein, editors, *Programming Languges meets Program Verification (PLPV)*, 2009.

[16] P. Wadler and P. Thiemann. The marriage of effects and monads. *ACM Trans. Comput. Logic*, 4(1):1–32, 2003.

A Proof of Theorem 1 (Soundness of Computational Translation)

The proof is a routine induction on the assumed $\mathtt{T}^{\mathtt{eq}\downarrow}$ typing derivation, which we include here for thoroughness:

A.1 Case: T_VAR

$$\frac{\Gamma(x) = T \quad \Gamma \vdash \mathbf{Ok}}{\Gamma \vdash x : T\ \theta} \quad \text{T\_VAR}$$

This case follows directly from the easily proven fact that $\Gamma(x) = T$ implies $[\![\Gamma]\!]^C(x) = [\![T]\!]^C$.

A.2 Case: T_JOIN

$$\frac{\begin{array}{c} t \rightsquigarrow^* t_0 \quad t' \rightsquigarrow^* t_0 \\ \Gamma \vdash t : T\ ? \quad \Gamma \vdash t' : T'\ ? \end{array}}{\Gamma \vdash \mathbf{join} : t = t'\ \theta} \quad \text{T\_JOIN}$$

The interpretation of the conclusion is just an instance of the STY_VAR rule. This is also true for the rules T_REIFY, T_INV, and T_CONTRA, so we omit cases for those rules.

A.3 Case: T_CONV

$$\frac{\Gamma \vdash t : [t_2/x]T\ \theta \quad \Gamma \vdash t' : t_1 = t_2\ \downarrow}{\Gamma \vdash t : [t_1/x]T\ \theta} \quad \text{T\_CONV}$$

By the IH we have $[\![\Gamma]\!]^c \vdash [\![t]\!]^C : [\![[t_2/x]T]\!]^C$. We omit the straightforward proof that $[\![[t_2/x]T]\!]^C = [\![T]\!]^C = [\![[t_1/x]T]\!]^C$, for any t_1, t_2, x, and T. So the fact we have from the first premise is what is required for the conclusion. The case for T_REFLECT is similar, and so is omitted.

A.4 Case: T_ABS

$$\frac{\Gamma, x : T' \vdash t : T\ \theta \quad \Gamma \vdash \Pi^\theta x{:}T'.T}{\Gamma \vdash \lambda x.t : \Pi^\theta x{:}T'.T\ \theta'} \quad \text{T\_ABS}$$

By the IH we have $[\![\Gamma, x : T']\!]^C \vdash [\![t]\!]^C : [\![T]\!]^C$. This is equivalent to $[\![\Gamma]\!]^C, x : [\![T']\!]^C \vdash [\![t]\!]^C : [\![T]\!]^C$, to which we can apply the simple typing rule STY_ABS to obtain $[\![\Gamma]\!]^C \vdash \lambda x.[\![t]\!]^C : [\![T']\!]^C \to [\![T]\!]^C$, which suffices by the definition of $[\![\cdot]\!]^C$.

A.5 Case: T_APP

$$\frac{\Gamma \vdash t : \Pi^\rho x{:}T'.T\ \theta \quad \Gamma \vdash t' : T'\ \theta \quad \rho \leq \theta}{\Gamma \vdash t\,t' : [t'/x]T\ \theta} \quad \text{T\_APP}$$

By the IH and the definition of $[\![\cdot]\!]^C$, we have $[\![\Gamma]\!]^C \vdash [\![t]\!]^C : [\![T']\!]^C \to [\![T]\!]^C$ and also $[\![\Gamma]\!]^C \vdash [\![t']\!]^C : [\![T']\!]^C$. We may apply the simple typing rule STY_APP to get $[\![\Gamma]\!]^C \vdash [\![t]\!]^C\ [\![t']\!]^C : [\![T]\!]^C$, which suffices, using again the definition of $[\![\cdot]\!]^C$, and also the fact used above that $[\![[t'/x]T]\!]^C = [\![T]\!]^C$.

A.6 Case: T_ZERO

$$\frac{\Gamma \vdash \mathbf{Ok}}{\Gamma \vdash 0 : \mathbf{nat}\ \theta}\ \text{T\_ZERO}$$

The desired conclusion is an instance of STY_ZERO.

A.7 Case: T_SUC

$$\frac{\Gamma \vdash t : \mathbf{nat}\ \theta}{\Gamma \vdash \mathbf{S}t : \mathbf{nat}\ \theta}\ \text{T\_SUC}$$

This case follows from the IH and then applying STY_SUC.

A.8 Case: T_RECNAT

$$\frac{\begin{array}{l} p \notin \mathbf{fv}\,t \\ \Gamma, f : \Pi^{?} x{:}\mathbf{nat}.T, x : \mathbf{nat}, p : \Pi^{\downarrow} x_1{:}\mathbf{nat}.\Pi^{\downarrow} q{:}x = \mathbf{S}x_1.\mathbf{Terminates}\ (f\,x_1) \vdash t : T\ \downarrow \end{array}}{\Gamma \vdash \mathbf{rec}\,f(x) = t : \Pi^{\downarrow} x{:}\mathbf{nat}.T\ \theta}\ \text{T\_RECNAT}$$

By the IH, we have:

$$[\![\Gamma, f : \Pi^{?} x{:}\mathbf{nat}.T, x : \mathbf{nat}, p : \Pi^{\downarrow} x_1{:}\mathbf{nat}.\Pi^{\downarrow} x_2{:}x = \mathbf{S}x_1.\mathbf{Terminates}\ (f\,x_1)]\!]^C \vdash [\![t]\!]^C : [\![T]\!]^C$$

This is equivalent to:

$$[\![\Gamma]\!]^C, f : \mathbf{nat} \to [\![T]\!]^C, x : nat, p : \mathbf{nat} \to \mathbf{nat} \to \mathbf{nat} \vdash [\![t]\!]^C : [\![T]\!]^C$$

We omit the straightforward proof that $\mathbf{fv}[\![t]\!]^C = \mathbf{fv}t$, which gives us $p \notin \mathbf{fv}[\![t]\!]^C$. We also omit the straightforward proof of Strengthening for our simply typed system, which says $\Sigma, x : A \vdash t : A'$ implies $\Sigma \vdash t : A'$ if $x \notin \mathbf{fv}\,t$. Using this Strengthening property for the simply typed system, we then have:

$$[\![\Gamma]\!]^C, f : \mathbf{nat} \to [\![T]\!]^C, x : nat \vdash [\![t]\!]^C : [\![T]\!]^C$$

We may now just apply the rule STY_REC, to conclude the desired $[\![\Gamma]\!]^C \vdash \mathbf{rec}\,f\,(x) = [\![t]\!]^C : \mathbf{nat} \to [\![T]\!]^C$. The case for T_REC is the same as the last part of this case, and so is omitted.

A.9 Case: T_CASENAT

$$\frac{\begin{array}{l} \Gamma \vdash t : \mathbf{nat}\ \theta \quad \Gamma \vdash t' : [0/x]T\ \theta \\ \Gamma \vdash t'' : \Pi^{\rho} x'{:}\mathbf{nat}.[\mathbf{S}x'/x]T\ \theta \quad \rho \leq \theta \end{array}}{\Gamma \vdash \mathbf{C}\ t\ t'\ t'' : [t/x]T\ \theta}\ \text{T\_CASENAT}$$

By the IH and the definition of $[\![\cdot]\!]^C$, we have:

- $[\![\Gamma]\!]^C \vdash [\![t]\!]^C : \mathbf{nat}$
- $[\![\Gamma]\!]^C \vdash [\![t']\!]^C : [\![[0/x]T]\!]^C$
- $[\![\Gamma]\!]^C \vdash [\![t'']\!]^C : \mathbf{nat} \to [\![[Sx'/x]T]\!]^C$

Using again the property mentioned above, that $[\![[t/x]T]\!]^C = [\![T]\!]^C$, we may then apply the rule STY_CASENAT to obtain the desired conclusion.

A.10 Case: T_ABORT

$$\frac{\Gamma \vdash \mathbf{Ok}}{\Gamma \vdash \mathbf{abort} : T\ ?}\ \text{T\_ABORT}$$

The desired conclusion is just an instance of STY_ABORT.

B Proof of Theorem 2 (Soundness of Logical Translation)

We prove this by induction on the structure of the assumed derivation. Note first that if the interpretation of a $\mathtt{T}^{\mathtt{eq}\downarrow}$ typing judgment with effect $\downarrow$ holds – that is, if we have $[\![\Gamma]\!]^C; [\![\Gamma]\!]^L \vdash \mathbf{Terminates}\ [\![t]\!]^C\ \wedge\ [\![T]\!]^L [\![t]\!]^C$ – then we also have $[\![\Gamma]\!]^C; [\![\Gamma]\!]^L \vdash \mathbf{Terminates}\ [\![t]\!]^C\ \Rightarrow\ [\![T]\!]^L [\![t]\!]^C$, which is the interpretation of the similar $\mathtt{T}^{\mathtt{eq}\downarrow}$ typing judgment with effect ?. So in cases where we can prove the interpretation of the judgment with $\downarrow$, we can omit the proof of the interpretation of the judgment with ?.

B.1 Case: T_VAR

$$\frac{\Gamma(x) = T \quad \Gamma \vdash \mathbf{Ok}}{\Gamma \vdash x : T\ \theta}\ \text{T\_VAR}$$

This case follows directly from the fact that the logical interpretation of the $\mathtt{T}^{\mathtt{eq}\downarrow}$ typing context Γ must contain **Terminates** x and $[\![T]\!]^L\ x$, since $\Gamma(x) = T$.

B.2 Case: T_JOIN

$$\frac{\begin{array}{c} t \rightsquigarrow^* t_0 \quad t' \rightsquigarrow^* t_0 \\ \Gamma \vdash t : T\ ? \quad \Gamma \vdash t' : T'\ ? \end{array}}{\Gamma \vdash \mathbf{join} : t = t'\ \theta}\ \text{T\_JOIN}$$

From the fact that $t \rightsquigarrow^* t_0$ implies $[\![t]\!]^C \rightsquigarrow^* [\![t_0]\!]^C$ (we omit the easy proof), we have that $[\![t]\!]^C$ and $[\![t']\!]^C$ are joinable. Our equational theory allows us to prove that joinable terms are equal. Hence, we can indeed prove the logical interpretation of the type in the conclusion, namely $[\![t]\!]^C = [\![t']\!]^C$.

B.3 Case: T_CONV

$$\frac{\Gamma \vdash t : [t_2/x]T\ \theta \quad \Gamma \vdash t' : t_1 = t_2\ \downarrow}{\Gamma \vdash t : [t_1/x]T\ \theta}\ \text{T\_CONV}$$

By the IH we have:

$$[\![\Gamma]\!]^C; [\![\Gamma]\!]^L \vdash [\![[t_2/x]T]\!]^L_\theta\ [\![t]\!]^C$$

We omit the straightforward proof that

$$[\![[t_2/x]T]\!]^L_\theta\ [\![t]\!]^C = [[\![t_2]\!]^C/x]([\![T]\!]^L_\theta\ [\![t]\!]^C)$$

Using this fact, it suffices to prove the similar statement, except with $[\![t_1]\!]^C$ in place of $[\![t_2]\!]^C$. But this follows by PV_SUBST, using the fact that $[\![\Gamma]\!]^C; [\![\Gamma]\!]^L \vdash [\![t_1]\!]^C = [\![t_2]\!]^C$. We have this from the formula we get by the IH for the second premise, noting that

$$[\![t_1 = t_2]\!]^L_\downarrow\ [\![t']\!]^C\ =\ \mathbf{Terminates}\ [\![t']\!]^C\ \wedge\ [\![t_1]\!]^C = [\![t_2]\!]^c$$

B.4 Case: T_REFLECT

$$\frac{\Gamma \vdash t : T\ \theta \quad \Gamma \vdash t' : \textbf{Terminates}\ t\ \downarrow}{\Gamma \vdash t : T\ \theta'}\ \text{T\_REFLECT}$$

If $\theta = \theta'$, the desired result follows immediately from the IH applied to the first premise. If $\theta \neq \theta'$ but $\theta = \downarrow$, we have already observed above that we can obtain the logical translation of a $\mathtt{T}^{\mathtt{eq}\downarrow}$ typing judgment with effect ? if we have the similar translation with effect $\downarrow$. So it suffices to consider just the case where $\theta = ?$ but $\theta' = \downarrow$. By the IH for the second premise, we have:

$$[\![\Gamma]\!]^C; [\![\Gamma]\!]^L \vdash \textbf{Terminates}\ [\![t']\!]^C\ \wedge\ \textbf{Terminates}\ [\![t]\!]^C$$

The second conjunct of this is exactly what we need to obtain the desired conclusion from what we get from the IH applied to the first premise, which is:

$$[\![\Gamma]\!]^C; [\![\Gamma]\!]^L \vdash \textbf{Terminates}\ [\![t]\!]^C\ \Rightarrow\ [\![T]\!]^L\ [\![t]\!]^C$$

B.5 Case: T_REIFY

$$\frac{\Gamma \vdash t : T\ \downarrow}{\Gamma \vdash \textbf{terminates} : \textbf{Terminates}\ t\ \theta}\ \text{T\_REIFY}$$

The IH for the premise is:

$$[\![\Gamma]\!]^C; [\![\Gamma]\!]^L \vdash \textbf{Terminates}\ [\![t]\!]^C\ \wedge\ [\![T]\!]^L\ [\![t]\!]^C$$

From this by PV_ANDE1 we obtain the translation of the conclusion, using also the axiom PV_TERMINATES0 (to show $\textbf{Terminates}\ [\![\textbf{terminates}]\!]^C$).

B.6 Case: T_CTXTERM

$$\frac{\Gamma \vdash t : \textbf{Terminates}\ \mathscr{C}[t']\ \theta}{\Gamma \vdash \textbf{inv} : \textbf{Terminates}\ t'\ \theta}\ \text{T\_CTXTERM}$$

It is sufficient to show $\textbf{Terminates}\ [\![\mathscr{C}]\!]^C[[\![t']\!]^C]\ \Rightarrow\ \textbf{Terminates}\ [\![t']\!]^C$, where $[\![\mathscr{C}]\!]^C$ is the context determined by the obvious extension of $[\![\cdot]\!]^C$ from terms to contexts. This formula easily follows using PV_TERMINATESINV.

B.7 Case: T_ABS

$$\frac{\Gamma, x : T' \vdash t : T\ \theta \quad \Gamma \vdash \Pi^\theta x{:}T'.T}{\Gamma \vdash \lambda\, x.t : \Pi^\theta x{:}T'.T\ \theta'}\ \text{T\_ABS}$$

Applying the IH to the first premise gives us:

$$[\![\Gamma]\!]^C, x : [\![T']\!]^C; [\![\Gamma]\!]^L, [\![T']\!]^L_\downarrow\ x \vdash [\![T]\!]^L_\theta [\![t]\!]^C$$

As remarked above, it suffices to prove the conclusion for when $\theta' = \downarrow$, since this implies the case when $\theta' = ?$. So we must prove:

$$[\![\Gamma]\!]^C; [\![\Gamma]\!]^L \vdash \textbf{Terminates}\ \lambda\, x.t \wedge \forall x : [\![T']\!]^C.\ [\![T']\!]^L_\downarrow\ x\ \Rightarrow [\![T]\!]^L_\theta\ [\![(\lambda\, x.t)\, x]\!]^C$$

The first conjunct is provable by PV_TERMINATESABS. The second follows easily from the fact we obtained from the IH, by applying PV_SUBST with the equation $(\lambda\, x.t)\, x = t$, which holds by PV_OPSEM (and then using also PV_ALLI and PV_IMPIE).

B.8 Case: T_App

$$\frac{\Gamma \vdash t : \Pi^{\rho} x{:}T'.T\ \theta \quad \Gamma \vdash t' : T'\ \theta \quad \rho \leq \theta}{\Gamma \vdash t\,t' : [t'/x]\,T\ \theta}\ \text{T\_App}$$

We first case-split on whether $\theta = ?$ or $\theta = \downarrow$. If $\theta = ?$, then by the IH, we have:

- $[\![\Gamma]\!]^C; [\![\Gamma]\!]^L \vdash \textbf{Terminates}\,[\![t]\!]^C \Rightarrow \forall x : [\![T']\!]^C.\ [\![T']\!]^L_{\downarrow}\ x\ \Rightarrow [\![T]\!]^L_{\rho}\ [\![t]\!]^C\ x$
- $[\![\Gamma]\!]^C; [\![\Gamma]\!]^L \vdash \textbf{Terminates}\,[\![t']\!]^C \Rightarrow [\![T']\!]^L\ [\![t']\!]^C$

We must prove:

$$[\![\Gamma]\!]^C; [\![\Gamma]\!]^L \vdash \textbf{Terminates}\,[\![t\,t']\!]^C \Rightarrow [\![[t'/x]\,T]\!]^L\ [\![t\,t']\!]^C$$

So (using Pv_Impi) assume $\textbf{Terminates}\,[\![t\,t']\!]^C$, and prove $[\![[t'/x]\,T]\!]^L\ [\![t\,t']\!]^C$. By Pv_TerminatesInv, we have $\textbf{Terminates}\,[\![t]\!]^C$ and $\textbf{Terminates}\,[\![t']\!]^C$. So from the facts we obtained above by the IH, we get (using Pv_Impe):

- $[\![\Gamma]\!]^C; [\![\Gamma]\!]^L \vdash \forall x : [\![T']\!]^C.\ [\![T']\!]^L_{\downarrow}\ x\ \Rightarrow [\![T]\!]^L_{\rho}\ [\![t]\!]^C\ x$
- $[\![\Gamma]\!]^C; [\![\Gamma]\!]^L \vdash [\![T']\!]^L\ [\![t']\!]^C$

We can instantiate the quantifier in the first fact, using Pv_Alle and Theorem 1 (to get $[\![\Gamma]\!]^C \vdash [\![t']\!]^C : [\![T']\!]^C$). This gives us the following from the first fact:

$$[\![\Gamma]\!]^C; [\![\Gamma]\!]^L \vdash [\![T']\!]^L_{\downarrow}\ x\ \Rightarrow [[\![t']\!]^C/x][\![T]\!]^L_{\rho}\ [\![t]\!]^C\ [\![t']\!]^C$$

The antecedent of this implication is provable from the second fact above and $\textbf{Terminates}\,[\![t']\!]^C$, giving us:

$$[\![\Gamma]\!]^C; [\![\Gamma]\!]^L \vdash [[\![t']\!]^C/x][\![T]\!]^L_{\rho}\ [\![t]\!]^C\ [\![t']\!]^C$$

Since we already have $\textbf{Terminates}\,[\![t]\!]^C\ [\![t']\!]^C$, from this we obtain (no matter what the value of ρ is)

$$[\![\Gamma]\!]^C; [\![\Gamma]\!]^L \vdash [[\![t']\!]^C/x][\![T]\!]^L\ [\![t]\!]^C\ [\![t']\!]^C$$

The desired conclusion then follows from the fact that $[[\![t']\!]^C/x][\![T]\!]^L_{\rho} = [\![[t'/x]\,T]\!]^L_{\rho}$. We omit the straightforward proof of this fact.

Now we must consider the case where $\theta = \downarrow$, and hence $\rho = \downarrow$ (from the rule's third premise). In this case, the IH gives us:

- $[\![\Gamma]\!]^C; [\![\Gamma]\!]^L \vdash \textbf{Terminates}\,[\![t]\!]^C \wedge \forall x : [\![T']\!]^C.\ [\![T']\!]^L_{\downarrow}\ x\ \Rightarrow [\![T]\!]^L_{\downarrow}\ [\![t]\!]^C\ x$
- $[\![\Gamma]\!]^C; [\![\Gamma]\!]^L \vdash \textbf{Terminates}\,[\![t']\!]^C \wedge [\![T']\!]^L\ [\![t']\!]^C$

We must prove:

$$[\![\Gamma]\!]^C; [\![\Gamma]\!]^L \vdash [\![[t'/x]\,T]\!]^L_{\downarrow}\ [\![t\,t']\!]^C$$

By the same reasoning as in the previous case, we obtain:

$$[\![\Gamma]\!]^C; [\![\Gamma]\!]^L \vdash [\![[t'/x]\,T]\!]^L_{\rho}\ [\![t]\!]^C\ [\![t']\!]^C$$

But this is exactly what we must prove, since $\rho = \downarrow$.

B.9 Case: T_ZERO

$$\frac{\Gamma \vdash \mathbf{Ok}}{\Gamma \vdash 0 : \mathbf{nat}\ \theta}\ \text{T\_ZERO}$$

It suffices to prove $[\![\Gamma]\!]^C; [\![\Gamma]\!]^L \vdash \mathbf{Terminates}\ 0 \wedge \mathbf{True}$, which follows easily using PV_TERMINATES0 and PV_TRUEI .

B.10 Case: T_SUC

$$\frac{\Gamma \vdash t : \mathbf{nat}\ \theta}{\Gamma \vdash \mathbf{S}\, t : \mathbf{nat}\ \theta}\ \text{T\_SUC}$$

We again case-split on whether $\theta = ?$ or $\theta = \downarrow$. In the former case, the IH gives us:

$$[\![\Gamma]\!]^C; [\![\Gamma]\!]^L \vdash \mathbf{Terminates}\, [\![t]\!]^C \wedge \mathbf{True}$$

We must prove:

$$[\![\Gamma]\!]^C; [\![\Gamma]\!]^L \vdash \mathbf{Terminates}\, S\ [\![t]\!]^C \wedge \mathbf{True}$$

This follows by PV_TERMINATESS. If $\theta = \downarrow$, we must prove

$$[\![\Gamma]\!]^C; [\![\Gamma]\!]^L \vdash \mathbf{Terminates}\, S\ [\![t]\!]^C \Rightarrow \mathbf{True}$$

But this is holds just by PV_IMPI and PV_TRUEI.

B.11 Case: T_REC

$$\frac{\Gamma, f : \Pi^? x{:}T'.T, x : T' \vdash t : T\ ?}{\Gamma \vdash \mathbf{rec}\, f(x) = t : \Pi^? x{:}T'.T\ \theta}\ \text{T\_REC}$$

By the IH, we have:

$$[\![\Gamma]\!]^C, f : [\![T']\!]^C \to [\![T]\!]^C, x : [\![T']\!]^C\ ;\ [\![\Gamma]\!]^L, \forall x : [\![T']\!]^C.\ [\![T']\!]^L_\downarrow\, x \Rightarrow [\![T]\!]^L_?\ (f\ x), [\![T']\!]^L_\downarrow x\ \vdash\ [\![T]\!]^L_?\ [\![t]\!]^C$$

Applying PV_IMPI and PV_ALLI, we get:

$$[\![\Gamma]\!]^C, f : [\![T']\!]^C \to [\![T]\!]^C\ ;\ [\![\Gamma]\!]^L, \forall x : [\![T']\!]^C.\ [\![T']\!]^L_\downarrow\, x \Rightarrow [\![T]\!]^L_?\ (f\ x)\ \vdash\ \forall x : [\![T']\!]^C.\ [\![T']\!]^L_\downarrow x \Rightarrow [\![T]\!]^L_?\ [\![t]\!]^C$$

This exactly matches the logical premise of the PV_COMPIND rule, with F taken to be $\forall x : [\![T']\!]^C.\ [\![T']\!]^L_\downarrow x \Rightarrow [\![T]\!]^L_?\ z$:

$$\frac{\Sigma, f : A' \to A; H, \forall x : A'.[f\,x/z]F \vdash \forall x : A'.[t/z]F \quad \Sigma \vdash \mathbf{rec}\, f(x) = t : A' \to A}{\Sigma; H \vdash \forall x : A'.\mathbf{Terminates}\ (\mathbf{rec}\, f(x) = t)\, x \Rightarrow [(\mathbf{rec}\, f(x) = t)\, x/z]F}\ \text{PV\_COMPIND}$$

So applying PV_COMPIND, we get the following fact (call it (J)):

$$[\![\Gamma]\!]^C\ ;\ [\![\Gamma]\!]^L\ \vdash\ \forall x : [\![T']\!]^C.\ \mathbf{Terminates}\, (\mathbf{rec}\ f(x) = [\![t]\!]^C)\ x \Rightarrow [\![T']\!]^L_\downarrow x \Rightarrow [\![T]\!]^L_?\ (\mathbf{rec}\ f(x) = [\![t]\!]^C)\ x$$

Note that the consequent $[\![T]\!]^L_?\ (\mathbf{rec}\ f(x) = [\![t]\!]^C)\ x$ of this implication is, by definition of $[\![\cdot]\!]^L_?$:

$$\mathbf{Terminates}\ (\mathbf{rec}\ f(x) = [\![t]\!]^C)\ x \Rightarrow [\![T]\!]^L\ (\mathbf{rec}\ f(x) = [\![t]\!]^C)\ x$$

So the premise of the implication in (J) is redundant, and we can deduce the following (J'):

$$[\![\Gamma]\!]^C \; ; \; [\![\Gamma]\!]^L \; \vdash \; \forall x : [\![T']\!]^C .\; [\![T']\!]^L_{\downarrow} x \Rightarrow [\![T]\!]^L_? \; (\mathbf{rec}\; f(x) = [\![t]\!]^C)\; x$$

It suffices now to prove:

$$[\![\Gamma]\!]^C ; [\![\Gamma]\!]^L \vdash \mathbf{Terminates\ rec}\; f(x) = [\![t]\!]^C \; \wedge \forall x : [\![T']\!]^C .\; [\![T']\!]^L_{\downarrow}\; x \Rightarrow [\![T]\!]^L_? \; (\mathbf{rec}\; f(x) = [\![t]\!]^C)\; x$$

The first conjunct is provable by PV_TERMINATESREC. The second now follows directly by what we have just deduced above.

B.12 Case: T_RECNAT

$$\frac{\begin{array}{l} p \notin \mathbf{fv}\, t \\ \Gamma, f : \Pi^? x{:}\mathbf{nat}.T, x : \mathbf{nat}, p : \Pi^{\downarrow} x_1{:}\mathbf{nat}.\Pi^{\downarrow} q{:}x = \mathbf{S}x_1.\mathbf{Terminates}\ (f\,x_1) \vdash t : T\ \downarrow \end{array}}{\Gamma \vdash \mathbf{rec}\, f(x) = t : \Pi^{\downarrow} x{:}\mathbf{nat}.T\ \theta} \quad \text{T\_RECNAT}$$

By the IH, we have

$$\begin{array}{l} [\![\Gamma]\!]^C, f : \mathbf{nat} \to [\![T]\!]^C, x : \mathbf{nat}, p : \mathbf{nat} \to (\mathbf{nat} \to \mathbf{nat})\ ; \\ [\![\Gamma]\!]^L, \mathbf{Terminates}\, f \ \wedge\ F_1, \mathbf{Terminates}\ x \ \wedge\ \mathbf{True}, \mathbf{Terminates}\ p \ \wedge\ F_2 \ \vdash\ [\![T]\!]^L_{\downarrow}\ [\![t]\!]^C \end{array}$$

where:

$$\begin{array}{rcl} F_1 & = & \forall x_1 : \mathbf{nat}.\, \mathbf{Terminates}\ x_1 \ \wedge\ \mathbf{True} \ \Rightarrow\ [\![T]\!]^L_?\ (f x_1) \\ F_2 & = & \forall x_1 : \mathbf{nat}.\, \mathbf{Terminates}\ x_1 \ \wedge\ \mathbf{True} \ \Rightarrow\ \mathbf{Terminates}\ (p\,x_1) \ \wedge \\ & & \forall x_2 : \mathbf{nat}.\, \mathbf{Terminates}\ x_2 \ \wedge\ x = \mathbf{S}x_1 \ \Rightarrow \mathbf{Terminates}\ ((p\,x_1)\,x_2) \ \wedge \\ & & \mathbf{Terminates}\ (f\,x_1) \end{array}$$

We may easily show that we can replace F_1 and F_2 by the following simplified versions:

$$\begin{array}{rcl} F'_1 & = & \forall x_1 : \mathbf{nat}.\, \mathbf{Terminates}\ x_1 \ \Rightarrow\ [\![T]\!]^L_?\ (f x_1) \\ F'_2 & = & \forall x_1 : \mathbf{nat}.\, \mathbf{Terminates}\ x_1 \ \Rightarrow\ x = \mathbf{S}x_1 \ \Rightarrow\ \mathbf{Terminates}\ (f\,x_1) \end{array}$$

This (and similar simplifications), followed by some uses of PV_ALLI and PV_IMPI, and also supplying an arbitrary lambda-abstraction of type $\mathbf{nat} \to (\mathbf{nat} \to \mathbf{nat})$ for p gives us the following central assumption (call it (J)) from the judgment above:

$$[\![\Gamma]\!]^C ; [\![\Gamma]\!]^L \ \vdash\ \forall x : \mathbf{nat}.\mathbf{Terminates}\ x \ \Rightarrow\ \forall f : \mathbf{nat} \to [\![T]\!]^C .\; (\mathbf{Terminates}\, f \ \wedge\ F'_1 \ \wedge\ F'_2) \ \Rightarrow\ [\![T]\!]^L_{\downarrow}\ [\![t]\!]^C$$

It suffices to show:

$$[\![\Gamma]\!]^C ; [\![\Gamma]\!]^L \vdash \mathbf{Terminates\ rec}\, f(x) = [\![t]\!]^C \ \wedge\ \forall x : \mathbf{nat}.\, \mathbf{Terminates}\ x \ \wedge\ \mathbf{True} \ \Rightarrow\ [\![T]\!]^L_{\downarrow}\ (\mathbf{rec}\, f(x) = [\![t]\!]^C)\; x$$

We have the first conjunct by PV_TERMINATESREC. For the second, it suffices to show:

$$[\![\Gamma]\!]^C ; [\![\Gamma]\!]^L \vdash \forall x : \mathbf{nat}.\, \mathbf{Terminates}\ x \ \Rightarrow\ [\![T]\!]^L_{\downarrow}\ (\mathbf{rec}\, f(x) = [\![t]\!]^C)\; x$$

We do this by induction (using PV_IND). First, though, we observe that the reasoning we used in the previous case (for T_REC) to prove what we called (J') applies here (except that here we have some additional assumptions in the context). This lets us deduce the following, which we will call (J') (essentially the (J') from the previous case, with T' replaced by **nat**):

$$\forall x : \mathbf{nat}.\, \mathbf{Terminates}\ x \ \Rightarrow\ [\![T]\!]^L_?\ (\mathbf{rec}\, f(x) = [\![t]\!]^C)\; x$$

So now for the base case, we must prove:

$$[\![\Gamma]\!]^C; [\![\Gamma]\!]^L \vdash [\![[\,0\,/x\,]\,T]\!]^L_{\downarrow}\ (\mathbf{rec}\, f(x) = [\![t]\!]^C)\ 0$$

This follows easily using PV_SUBST and PV_OPSEM from:

$$[\![\Gamma]\!]^C; [\![\Gamma]\!]^L \vdash [\![[\,0\,/x\,]\,T]\!]^L_{\downarrow}\ ([0/x][\mathbf{rec}\, f(x) = [\![t]\!]^C/f][\![t]\!]^C$$

We obtain this by instantiating (J) above with 0 for x, and $\mathbf{rec}\, f(x) = [\![t]\!]^C$ for f. We have the required proofs of termination by PV_TERMINATES0 and PV_TERMINATESREC. We have a proof of the appropriately instantiated premise F'_1 of (J), since this is exactly (J'). We easily prove the instantiation of premise F'_2 of (J), since this is:

$$\forall x_1 : \mathbf{nat}.\, \mathbf{Terminates}\ x_1\ \Rightarrow\ 0 = \mathbf{S}x_1\ \Rightarrow\ \mathbf{Terminates}\ ((\mathbf{rec}\, f(x) = [\![t]\!]^C)x1)$$

This formula is easily proved using PV_CONTRA with premise $0 = \mathbf{S}x_1$. So from (J), with these instantiations and proven premises, we obtain the following, which is exactly what we had to prove:

$$[\![\Gamma]\!]^C; [\![\Gamma]\!]^L \vdash [\![[\,0\,/x\,]\,T]\!]^L_{\downarrow}\ (\mathbf{rec}\, f(x) = [\![t]\!]^C)\ 0$$

For the step case, we must now prove:

$$[\![\Gamma]\!]^C, x' : \mathbf{nat}; [\![\Gamma]\!]^L, \mathbf{Terminates}\ x', [\![[\,x'\,/x\,]\,T]\!]^L_{\downarrow}\ (\mathbf{rec}\, f(x) = [\![t]\!]^C)\ x' \vdash [\![[\,\mathbf{S}x'\,/x\,]\,T]\!]^L_{\downarrow}\ (\mathbf{rec}\, f(x) = [\![t]\!]^C)\ (\mathbf{S}\,x')$$

Now we instantiate (J) above with $\mathbf{S}x'$ for x, and again $\mathbf{rec}\, f(x) = [\![t]\!]^C$ for f. We easily obtain the required proofs of termination. The instantiated F'_1 we again have by (J'). The instantiated premise F'_2 is:

$$\forall x_1 : \mathbf{nat}.\, \mathbf{Terminates}\ x_1\ \Rightarrow\ \mathbf{S}x' = \mathbf{S}x_1\ \Rightarrow\ \mathbf{Terminates}\ ((\mathbf{rec}\, f(x) = [\![t]\!]^C)x1)$$

We can prove this premise as follows. Assume arbitrary terminating x_1 of sort **nat**, and assume $\mathbf{S}x' = \mathbf{S}x_1$. Using PV_SUBST and PV_OPSEM, we can derive $x' = x_1$ from this:

$$\dfrac{\dfrac{\mathbf{S}x' = \mathbf{S}x_1 \quad \dfrac{}{\mathbf{C}\ (\mathbf{S}x')\ 0\ \lambda\, z.\, z = x'}\ \text{PV\_OPSEM}}{\mathbf{C}\ (\mathbf{S}x_1)\ 0\ \lambda\, z.\, z = x'}\ \text{PV\_SUBST} \qquad \dfrac{}{\mathbf{C}\ (\mathbf{S}x_1)\ 0\ \lambda\, z.\, z = x_1}\ \text{PV\_OPSEM}}{x' = x_1}\ \text{PV\_SUBST}$$

So now to complete the proof of the instantiated premise F'_2, we need only prove

$$\mathbf{Terminates}\ (\mathbf{rec}\, f(x) = [\![t]\!]^C)\ x'$$

But this follows directly from the assumption we have in this step case of PV_IND:

$$[\![[\,x'\,/x\,]\,T]\!]^L_{\downarrow}\ (\mathbf{rec}\, f(x) = [\![t]\!]^C)\ x'$$

So we have all the premises required by (J), and we can prove the following (applying a derived weakening rule, whose easy proof is omitted, to (J) to add our other assumptions to its contexts):

$$[\![[\,\mathbf{S}x'\,/x\,]\,T]\!]^L_{\downarrow}\ [\mathbf{S}x'/x][(\mathbf{rec}\, f(x) = [\![t]\!]^C)/f][\![t]\!]^C$$

This now easily implies the required conclusion, using PV_SUBST with the following equation, which holds by PV_OPSEM:

$$[\mathbf{S}x'/x][(\mathbf{rec}\, f(x) = [\![t]\!]^C)/f][\![t]\!]^C = (\mathbf{rec}\, f(x) = [\![t]\!]^C)(\mathbf{S}x')$$

B.13 Case: T_CaseNat

$$\frac{\Gamma \vdash t : \mathbf{nat}\ \theta \quad \Gamma \vdash t' : [0/x]T\ \theta \quad \Gamma \vdash t'' : \Pi^{\rho} x' : \mathbf{nat}.[\mathbf{S}x'/x]T\ \theta \quad \rho \leq \theta}{\Gamma \vdash \mathbf{C}\ t\ t'\ t'' : [t/x]T\ \theta}\ \text{T\_CaseNat}$$

As for some cases above, we begin by case-splitting on whether $\theta = ?$ or $\theta = \downarrow$. Suppose $\theta = ?$. Then applying the IH to the second and third premises, and then a few basic logical simplifications, gives us:

1. $[\![\Gamma]\!]^C; [\![\Gamma]\!]^L \vdash \mathbf{Terminates}\,[\![t']\!]^C \Rightarrow [\![[0/x]T]\!]^L [\![t']\!]^C$
2. $[\![\Gamma]\!]^C; [\![\Gamma]\!]^L \vdash \mathbf{Terminates}\,[\![t'']\!]^C \Rightarrow \forall x' : \mathbf{nat}.\mathbf{Terminates}\,x' \Rightarrow [\![[\mathbf{S}x'/x]T]\!]^L_{\rho}\ ([\![t'']\!]^C\ x')$

We must show:

$$[\![\Gamma]\!]^C; [\![\Gamma]\!]^L \vdash \mathbf{Terminates}\,[\![\mathbf{C}\ t\ t'\ t'']\!]^C \Rightarrow [\![[t/x]T]\!]^L [\![\mathbf{C}\ t\ t'\ t'']\!]^C$$

So assume $\mathbf{Terminates}\,[\![\mathbf{C}\ t\ t'\ t'']\!]^C$, and show $[\![[t/x]T]\!]^L [\![\mathbf{C}\ t\ t'\ t'']\!]^C$. By Pv_TerminatesInv, this assumption implies $\mathbf{Terminates}\,[\![t]\!]^C$. Since $[\![\Gamma]\!]^C \vdash [\![t]\!]^C : \mathbf{nat}$ by Theorem 1, we will now seek to prove the following by Pv_Ind:

$$[\![\Gamma]\!]^C; [\![\Gamma]\!]^L \vdash \forall x : \mathbf{nat}.\mathbf{Terminates}\ x \Rightarrow \mathbf{Terminates}\,[\![\mathbf{C}\ x\ t'\ t'']\!]^C \Rightarrow [\![T]\!]^L [\![\mathbf{C}\ x\ t'\ t'']\!]^C$$

If we can derive this judgment, then we can instantiate x with $[\![t]\!]^C$ (for which we have $\mathbf{Terminates}\,[\![t]\!]^C$) to conclude the desired

$$[\![\Gamma]\!]^C; [\![\Gamma]\!]^L \vdash \mathbf{Terminates}\,[\![\mathbf{C}\ t\ t'\ t'']\!]^C \Rightarrow [\![[t/x]T]\!]^L [\![\mathbf{C}\ t\ t'\ t'']\!]^C$$

To apply Pv_Ind as desired, we must prove the base case and step case of the induction:

- $[\![\Gamma]\!]^C; [\![\Gamma]\!]^L \vdash \mathbf{Terminates}\,[\![\mathbf{C}\ 0\ t'\ t'']\!]^C \Rightarrow [\![[0/x]T]\!]^L [\![\mathbf{C}\ 0\ t'\ t'']\!]^C$
- $[\![\Gamma]\!]^C, x' : \mathbf{nat}; [\![\Gamma]\!]^L, \mathbf{Terminates}\ x' \vdash \mathbf{Terminates}\,[\![\mathbf{C}\ (\mathbf{S}x')\ t'\ t'']\!]^C \Rightarrow [\![[\mathbf{S}x'/x]T]\!]^L [\![\mathbf{C}\ (\mathbf{S}x')\ t'\ t'']\!]^C$

This base case follows from fact (1) above, using the equation $[\![\mathbf{C}\ 0\ t'\ t'']\!]^C = [\![t']\!]^C$. This equation is easily shown by the definition of $[\![\cdot]\!]^C$ and Pv_OpSem. So we now prove the step case. First, we simplify the desired formula using the easily proved equation $[\![\mathbf{C}\ (\mathbf{S}x')\ t'\ t'']\!]^C = [\![t''\ x']\!]^C$. So our new goal formula is

$$[\![\Gamma]\!]^C, x' : \mathbf{nat}; [\![\Gamma]\!]^L, \mathbf{Terminates}\ x' \vdash \mathbf{Terminates}\,[\![t''\ x']\!]^C \Rightarrow [\![[\mathbf{S}x'/x]T]\!]^L [\![t''\ x']\!]^C$$

So assume $\mathbf{Terminates}\ x'$ and $\mathbf{Terminates}\,[\![t''\ x']\!]^C$, and show $[\![[\mathbf{S}x'/x]T]\!]^L [\![t''\ x']\!]^C$. Instantiating fact (2) above with x' and these assumptions, we get:

$$[\![[\mathbf{S}x'/x]T]\!]^L_{\rho}\ ([\![t'']\!]^C\ x')$$

This implies the desired formula in either possible case for ρ.

Now let us assume $\theta = \downarrow$. This case is similar to the above, so we give fewer details. The IH for the three premises gives us:

1. $[\![\Gamma]\!]^C; [\![\Gamma]\!]^L \vdash \mathbf{Terminates}\,[\![t]\!]^C \wedge \mathbf{True}$
2. $[\![\Gamma]\!]^C; [\![\Gamma]\!]^L \vdash \mathbf{Terminates}\,[\![t']\!]^C \wedge [\![[0/x]T]\!]^L [\![t']\!]^C$
3. $[\![\Gamma]\!]^C; [\![\Gamma]\!]^L \vdash \mathbf{Terminates}\,[\![t'']\!]^C \wedge \forall x' : \mathbf{nat}.\mathbf{Terminates}\,x' \Rightarrow [\![[\mathbf{S}x'/x]T]\!]^L_{\rho}\ ([\![t'']\!]^C\ x')$

We must show

$$[\![\Gamma]\!]^C; [\![\Gamma]\!]^L \vdash \textbf{Terminates}\, [\![\mathbf{C}\ t\ t'\ t'']\!]^C \wedge [\![[t/x]\,T]\!]^L [\![\mathbf{C}\ t\ t'\ t'']\!]^C$$

Since we have **Terminates** $[\![t]\!]^C$ and $[\![\Gamma]\!]^C \vdash [\![t]\!]^C : \textbf{nat}$, it suffices to prove the following more general statement:

$$[\![\Gamma]\!]^C; [\![\Gamma]\!]^L \vdash \forall x : \textbf{nat}.\textbf{Terminates}\ x \Rightarrow \textbf{Terminates}\, [\![\mathbf{C}\ x\ t'\ t'']\!]^C \wedge [\![T]\!]^L [\![\mathbf{C}\ x\ t'\ t'']\!]^C$$

We again apply PV_IND. The base case is again immediate using $[\![\mathbf{C}\ 0\ t'\ t'']\!]^C = [\![t']\!]^C$. Similarly reasoning as for the step case above gives us:

$$[\![[\mathbf{S}x'/x]\,T]\!]^L_\rho\ ([\![t'']\!]^C\ x')$$

Since ρ must be $\downarrow$ in this case, we obtain from this fact the desired **Terminates** $([\![t'']\!]^C\ x')$, as well as

$$[\![[\mathbf{S}x'/x]\,T]\!]^L\ ([\![t'']\!]^C\ x')$$

B.14 Case: T_CONTRA

$$\frac{\Gamma \vdash t : 0 = \mathbf{S}t'\ \downarrow}{\Gamma \vdash \textbf{contra} : T\ \theta}\ \text{T\_CONTRA}$$

By the IH we have $[\![\Gamma]\!]^C; [\![\Gamma]\!]^L \vdash \textbf{Terminates}\ 0\ \wedge\ 0 = \mathbf{S}\ [\![t']\!]^C$. We must show $[\![\Gamma]\!]^C; [\![\Gamma]\!]^L \vdash [\![T]\!]^L_\theta\ 0$. But this fact follows directly from the second conjunct of the fact we have, using PV_CONTRA.

B.15 Case: T_ABORT

$$\frac{\Gamma \vdash \textbf{Ok}}{\Gamma \vdash \textbf{abort} : T\ ?}\ \text{T\_ABORT}$$

We must prove $[\![\Gamma]\!]^C; [\![\Gamma]\!]^L \vdash \textbf{Terminates}\ \textbf{abort} \Rightarrow [\![T]\!]^L\ \textbf{abort}$. But this follows directly by PV_IMPI from PV_NOTTERMINATESABORT.

C Typing rules for annotated system

$\boxed{\Gamma \Vdash a : T\ \theta}$

$$\frac{\Gamma(y) = T \quad \Gamma \vdash \textbf{Ok}}{\Gamma \Vdash y : T\ \theta}\ \text{AT\_VAR}$$

$$\frac{\begin{array}{c}|a| \rightsquigarrow^N t \quad |a'| \rightsquigarrow^N t \\ \Gamma \Vdash a : T\ ? \quad \Gamma \Vdash a' : T'\ ?\end{array}}{\Gamma \Vdash \textbf{join}\, a\, a' : |a| = |a'|\ \theta}\ \text{AT\_JOIN}$$

$$\frac{\Gamma \Vdash a : [t_2/x]\,T\ \theta \quad \Gamma \Vdash a' : t_1 = t_2\ \downarrow}{\Gamma \Vdash \textbf{conv}\, x.T\, a\, a' : [t_1/x]\,T\ \theta}\ \text{AT\_CONV}$$

$$\frac{\Gamma \Vdash a : T\ \theta \quad \Gamma \Vdash a' : \textbf{Terminates}\ |a|\ \downarrow}{\Gamma \Vdash \textbf{reflect}\, a\, a' : T\ \theta'}\ \text{AT\_REFLECT}$$

$$\frac{\Gamma \Vdash a : T\ \downarrow}{\Gamma \Vdash \textbf{terminates}\, a : \textbf{Terminates}\ |a|\ \theta}\ \text{AT\_REIFY}$$

$$\frac{\Gamma \Vdash a : \textbf{Terminates}\ \mathscr{C}[\,|a'|\,]\ \theta}{\Gamma \Vdash \textbf{inv}\, a\, a' : \textbf{Terminates}\ |a'|\ \theta}\ \text{AT\_CTXTERM}$$

$$\frac{\Gamma, y : T' \Vdash a : [\,|y|\,/x]T\ \theta \quad \Gamma \vdash \Pi^{\theta} x{:}T'.T}{\Gamma \Vdash \lambda\, y : T'.\, a : \Pi^{\theta} x{:}T'.T\ \theta'}\ \text{AT\_ABS}$$

$$\frac{\Gamma \Vdash a : \Pi^{\rho} x{:}T'.T\ \theta \quad \Gamma \Vdash a' : T'\ \theta \quad \rho \leq \theta}{\Gamma \Vdash a\, a' : [\,|a'|\,/x]T\ \theta}\ \text{AT\_APP}$$

$$\frac{\Gamma \vdash \textbf{Ok}}{\Gamma \Vdash 0 : \textbf{nat}\ \theta}\ \text{AT\_ZERO}$$

$$\frac{\Gamma \Vdash a : \textbf{nat}\ \theta}{\Gamma \Vdash \mathbf{S}\, a : \textbf{nat}\ \theta}\ \text{AT\_SUC}$$

$$\frac{\Gamma, g : \Pi^{?} x{:}T'.T, y : T' \Vdash a : [\,|y|\,/x]T\ ?}{\Gamma \Vdash \textbf{rec}\ g(y : T') : T = a : \Pi^{?} x{:}T'.T\ \theta}\ \text{AT\_REC}$$

$$\frac{\begin{array}{l} p \notin \textbf{fv}\,|a| \\ \Gamma, g : \Pi^{?} x{:}\textbf{nat}.T, y : \textbf{nat}, p : \Pi^{\downarrow} x_1{:}\textbf{nat}.\Pi^{\downarrow} u{:}|y| = \mathbf{S}\, x_1.\textbf{Terminates}\ (f\, x_1) \Vdash a : [\,|y|\,/x]T\ \downarrow \end{array}}{\Gamma \Vdash \textbf{rec}_{\textbf{nat}}\ g(y\, p) : T = a : \Pi^{\downarrow} x{:}\textbf{nat}.T\ \theta}\ \text{AT\_RECNAT}$$

$$\frac{\begin{array}{l} \Gamma \Vdash a : \textbf{nat}\ \theta \quad \Gamma \Vdash a' : [0/x]T\ \theta \\ \Gamma \Vdash a'' : \Pi^{\rho} x'{:}\textbf{nat}.[\mathbf{S}\, x'/x]T\ \theta \quad \rho \leq \theta \end{array}}{\Gamma \Vdash \mathbf{C}\, x.T\ a\ a'\ a'' : [\,|a|\,/x]T\ \theta}\ \text{AT\_CASENAT}$$

$$\frac{\Gamma \Vdash a : 0 = \mathbf{S}\, t'\ \downarrow}{\Gamma \Vdash \textbf{contra}\, T\, a : T\ \theta}\ \text{AT\_CONTRA}$$

$$\frac{\Gamma \vdash \textbf{Ok}}{\Gamma \Vdash \textbf{abort}\, T : T\ ?}\ \text{AT\_ABORT}$$

D Functions for interpreting terminating recursion in Coq

The following Coq definitions indicate how one could interpret **Terminates** t in terms of the partiality monad.

```
Section Terminates.

Variable A:Set.
Inductive Terminates : Partial A -> Set :=
  terminates_return : forall a:A, Terminates (rtrn a)
| terminates_step : forall (x:Partial A), Terminates x -> Terminates (step x).

Fixpoint reflect (x : Partial A) (p : Terminates x) {struct p} :A :=
  match p with
  | terminates_return a => a
  | terminates_step x' p' => reflect p'
  end.
End Terminates.
```

```
Section REC_nat_total.

(* With the given assumptions we know that we can write
   down a lambda expression of the following type: *)

Variable phi : Set.
Variable F :
  forall (f : forall (x:nat), Partial phi)
         (x : nat)
         (v : forall (y:nat), x = S y -> Terminates (f y)),
    phi.
Implicit Arguments F [].

Require Import Peano_dec.

Definition REC_nat_total (x : nat)  : phi.
 intros x. induction x.
 apply (F (fun y => infinite_step phi) 0).
   intros; congruence.
 pose (f := fun (y:nat) => match eq_nat_dec x (S y) with
                   | left pf => rtrn IHx
                   | right pf => infinite_step phi
                   end).
 apply (F f x).
   intros. unfold f. destruct (eq_nat_dec x (S y)).
     apply terminates_return.
     congruence.
Defined.
End REC_nat_total.
```