Structural Logical Relations*

Carsten Schürmann
IT University of Copenhagen
Copenhagen, Denmark
carsten@itu.dk

Jeffrey Sarnat
Yale University
New Haven, CT, USA
sarnat@cs.yale.edu

Abstract

Tait's method (a.k.a. proof by logical relations) is a powerful proof technique frequently used for showing foundational properties of languages based on typed λ-calculi. Historically, these proofs have been extremely difficult to formalize in proof assistants with weak meta-logics, such as Twelf, and yet they are often straightforward in proof assistants with stronger meta-logics. In this paper, we propose structural logical relations *as a technique for conducting these proofs in systems with limited meta-logical strength by explicitly representing and reasoning about an auxiliary logic. In support of our claims, we give a Twelf-checked proof of the completeness of an algorithm for checking equality of simply typed λ-terms.*

1. Introduction

When formalizing meta-theoretic proofs about formal systems, one is usually first confronted with choosing a proof assistant. This decision is usually driven by concerns regarding the expressive power of the proof assistant, the trust in its logical foundation, performance, the features of the tactic language, and the ease with which we can encode the formal system.

For example, Thomas Hales has chosen HOL Light [Har96] for his work on the formalization of Kepler's conjecture. George Necula has chosen variants of the LF type theory for his original work on proof-carrying code [Nec97]. Daniel Lee, Karl Crary, and Bob Harper have chosen the Twelf system for formalizing the proof of the soundness of an SML type system [LCH07] and Georges Gonthier has chosen Coq for his formalized proof of the four color theorem [Gon05]. The solutions posted to the POPLmark challenge employ a whole range of different proof assistants, including Twelf, Coq, Isabelle/HOL, Matita, Alpha Prolog, and ATS.

*This research has been funded by the National Science Foundation under grant CCR-0325808.

The proof theoretic strength of the different systems vary significantly. Once the proof assistant question is settled and a proof development has begun, there is often no turning back. If unforeseen challenges in the formalization of a meta-theoretic argument arise, one can find oneself in a situation where one is seemingly stuck. This might suggest that the best proof assistant is necessarily the most expressive, but such a perspective is too naive with respect to other considerations that influence the choice of proof assistant.

In this paper we present a technique called *structural logical relations* that avails proofs by logical relations to systems with limited meta-logical strength by explicitly representing and reasoning about an auxiliary logic. Proofs by structural logical relations follow the same line of reasoning as their informal counterparts. The central idea is to formalize the relevant part of the argument in this auxiliary logic, which we refer to as the *assertion logic*. Structural logical relations are therefore best employed in proof assistants based on logical frameworks, such as Twelf, that provide adequate, elegant, and convenient to work with encodings of proofs in the assertion logic.

We show how to formalize structural logical relations in practice using the meta-logical framework Twelf. But this paper is not just about the *how*, it also highlights some insights into the proof itself. For example, we illustrate that a proof by logical relations can be viewed as a recipe to reduce the provability of a theorem to the normalizability of proofs in the assertion logic. We present the results in such a way that the reader interested in the nature of logical proofs, but not in their formalization, should be able to follow this paper by ignoring the formalized material.

To best follow the paper, however, it is important that we distinguish carefully between the *logical framework*, in our case LF [HHP93] that serves the formalization of judgments, the *assertion logic* that we will encode in LF, and the *meta-logic* of Twelf, in which we carry out proofs by structural logical relations. Our proofs are constructive and executable, laying to rest the common misconception that the meta-logical framework Twelf is unable to reason with proofs by logical relations. The $FOL^{\Delta N}$

system [MM97] has a similar distinction between logical framework and meta-logic; we suspect that structural logical relations would be convenient in that setting as well.

This paper is structured as follows: In Section 2 we define the term *structural logical relation* and give a simple yet useful example of an assertion logic, namely minimal first-order logic, suitable for reasoning about simply typed languages. In Section 3, we briefly comment on the formalization of the technique in the proof assistant Twelf. We give an introductory example of a structural logical relations proof, namely the weak normalization for the simply typed λ-calculus in Section 4 before we address the goal of this paper, the completeness of equivalence checking in Section 5. An identifying feature of our structural logical relations proofs is the cut-elimination property of the assertion logic. Therefore, we study the features and implications of cut-elimination of the assertion logic in Section 6. We conclude with Section 7 where we discuss implications and possible extensions. The accompanying Twelf source code [SS08] is available at `http://www.twelf.org/slr`.

2. Structural Logical Relations

A proof by logical relations (also sometimes referred to as Tait's method) relies on an interpretation of types as relations between objects. On paper, these interpretations are usually formulated in set theory. In the spirit of intuitionism, we not only want to know that a certain property holds, but we would like to be able to produce a constructive witness of this fact.

Consider the simply typed λ-calculus.

$$\begin{array}{lrcl} \text{Types} & \tau & ::= & \mathsf{o} \mid \tau_1 \Rightarrow \tau_2 \\ \text{Expressions} & e & ::= & x^\tau \mid \mathsf{lam}\ x^\tau.\, e \mid \mathsf{app}\ e_1\ e_2 \end{array}$$

We refer to a property that is logic independent as a *judgment*. Examples of judgments J include, for example, e *reduces to a normal form*, e *weak-head reduces to* e', or e *is* $\beta\eta$*-equivalent to* e'. Given some judgment $J(e)$, a unary logical relation $[\![\tau]\!]$ can be defined as the smallest set that satisfies the following two conditions.

$$\begin{array}{lll} e \in [\![\mathsf{o}]\!] & \text{iff} & J(e) \\ e \in [\![\tau_2 \Rightarrow \tau_1]\!] & \text{iff} & \text{for all } e_2, \text{ if } e_2 \in [\![\tau_2]\!] \\ & & \quad \text{then } \mathsf{app}\ e\ e_2 \in [\![\tau_1]\!] \end{array}$$

The hallmark characteristic of a logical relation is the definition at function types: functions are related if they map related arguments to related results. A proof by logical relations typically proceeds in two stages: first, it is shown that $e \in [\![\tau]\!]$ implies $J(e)$, then it is shown that, for every e of type τ, $e \in [\![\tau]\!]$. A paper proof for the weak normalization of the simply typed λ-calculus can be found in [Pfe92]; a proof using structural logical relations for the same property is given in Section 4.

Taking a step back, we notice that the logical relation defined above relies on two things: structural induction on τ, judgments of the form of J, and connectives "for all" and "if ... then." Logical relations are typically formulated with the understanding that the induction principle, judgments, and connectives live on the same level. The idea of *structural logical relations* is to separate these concepts: induction lives on the meta-level, whereas the judgments and connectives live elsewhere. To this end, we declare a logic, the *assertion logic*, that is expressive enough to define both a logical relation in terms of logical connectives and atomic formulas P that represent the judgments J in the logic. The formulas of our assertion logic that we use throughout this paper are defined by the following.

$$\begin{array}{lrcl} \text{Formulas} & F, G & ::= & P \mid F \supset G \mid \forall x^\tau.\, F \\ \text{Assumptions} & \Delta & ::= & \cdot \mid \Delta, F \mid \Delta, x^\tau \end{array}$$

In the definition of expressions, every variable carries its own type, which means that we can restrict the universal quantifier to range over well-typed expressions even if these contain free variables. We write e^τ for an expression e of type τ and omit the superscript τ whenever it can be easily inferred.

We define the judgment of derivability in a sequent calculus $\Delta \vdash F$ by the following rules. If Δ is $\cdot$, then we simply drop it from the judgment.

$$\frac{F \in \Delta}{\Delta \vdash F}\,\mathsf{ax} \qquad \frac{\Delta, F \vdash G}{\Delta \vdash F \supset G}\,\mathsf{impR}$$

$$\frac{\Delta, F \supset G \vdash F \quad \Delta, F \supset G, G \vdash H}{\Delta, F \supset G \vdash H}\,\mathsf{impL}$$

$$\frac{\Delta, x^\tau \vdash F}{\Delta \vdash \forall x^\tau.\, F}\,\mathsf{allR} \qquad \frac{\Delta, \forall x^\tau.\, F, [e^\tau/x]F \vdash H}{\Delta, \forall x^\tau.\, F \vdash H}\,\mathsf{allL}$$

The rules are standard. We let the universal quantifier range over expressions e^τ. We write $[e^\tau/x]F$ for the usual notion of capture-avoiding substitution.

For practical reasons we will work with the assertion logic with cut. The judgment $\Delta \stackrel{\text{cut}}{\vdash} F$ is defined by inference rules similar to the ones above except with $\stackrel{\text{cut}}{\vdash}$ instead of $\vdash$, plus one additional rule.

$$\frac{\Delta \stackrel{\text{cut}}{\vdash} F \quad \Delta, F \stackrel{\text{cut}}{\vdash} G}{\Delta \stackrel{\text{cut}}{\vdash} G}\,\mathsf{cut}$$

The rules for defining the atomic formulas P vary for each structural logical relation. Hence they are defined in Sections 4 and 5, where we discuss concrete applications of our technique. We show later in the paper that the choice of

inference rules cannot be arbitrary: they must not destroy the cut-elimination property of the sequent-calculus.

Returning to the definition of *structural logical relations*, we can now define $[\![\tau]\!]$ more formally by meta-level induction on τ:

$$\begin{array}{rcl} [\![\mathsf{o}]\!](e) & = & P(e) \\ [\![\tau_2 \Rightarrow \tau_1]\!](e) & = & \forall e'^{\tau_2}.\, [\![\tau_2]\!](e') \supset [\![\tau_1]\!](\mathsf{app}\ e\ e') \end{array}$$

And more generally, the definition of an n-ary structural logic relation is given by the following formula:

$$\begin{array}{rcl} [\![\mathsf{o}]\!](e_1 \ldots e_n) & = & P(e_1 \ldots e_n) \\ [\![\tau_2 \Rightarrow \tau_1]\!](e_1 \ldots e_n) & & \\ & = & \forall e_1'^{\tau_2}. \ldots \forall e_n'^{\tau_2}. [\![\tau_2]\!](e_1' \ldots e_n') \\ & & \quad \supset [\![\tau_1]\!](\mathsf{app}\ e_1\ e_1' \ldots \mathsf{app}\ e_n\ e_n') \end{array}$$

An example of a binary structural logical relation can be found in Section 5. Next, we derive the proof obligations justifying a proof by structural logical relations in the unary case.

1. If e^τ then $\vdash^{\text{cut}} [\![\tau]\!](e)$. (Fundamental Theorem)
2. $\vdash^{\text{cut}} [\![\tau]\!](e) \supset P(e)$. (Escape Lemma)
3. If $\vdash^{\text{cut}} P(e)$ then $\vdash P(e)$. (Cut-Elimination)
4. If $\vdash P(e)$ then $J(e)$. (Extraction Theorem)

Note that, instead of defining the structural logical relation using meta-level induction, we could have chosen to extend the assertion logic axiomatically by an induction principle over types. However, this would have not only been less convenient, but would have also complicated the cut-elimination proof for our assertion logic, which we discuss in Sections 4 and 5.

3. The Meta-Logical Framework Twelf

Turning to the formalization of a structural logical relations argument, we follow [Pfe95], and describe adequate encodings of the simply typed λ-calculus and the sequent calculus in the logical framework LF [HHP93] using Twelf notation. First we address the formalization of simple types.

```
tp : type.
o  : tp.
=> : tp -> tp -> tp. %infix right 10 =>.
```

Any LF object made from arrows `=>` and base types `o` corresponds to a well-formed simple type, and vice versa. As `=>` is declared infix, `o => o => o` is the representation of a function type that expects two arguments. In Twelf, we write `type` for the kind of a type, and use curly braces `{t:tp}` for dependent types and dependent kinds. We write `->` (infix) if the dependency is vacuous. Furthermore, we write `[t:tp]` for λ-abstraction, `t:tp` for type ascription, and juxtaposition for application in the logical framework.

Next, we define the syntactic category of terms e^τ. If τ is represented by `T:tp`, then we write `tm T` for the LF type representing simply typed expressions e^τ. The definition takes advantage of the fact that, in LF, type families may be indexed by LF objects.

```
tm  : tp -> type.
lam : (tm T1 -> tm T2) -> tm (T1 => T2).
app : tm (T2 => T1) -> tm T2 -> tm T1.
```

The type of `lam` illustrates our use of higher-order abstract syntax, which we use pervasively throughout this paper: variables of the simply typed λ-calculus and assumptions about the being well-typed are represented by variables of LF.

Formulas of the assertion logic are encoded as LF objects of type `form`. We defer the formalization of atomic formulas to the subsequent relevant sections in this paper.

```
form   : type.
forall : (tm T -> form) -> form.
==>    : form -> form -> form.
                  %infix right 10 ==>.
```

Every sequent derivation of $F_1, \ldots, F_n \vdash G$ can be adequately represented in LF as an object of type `conc` G in a context $u_1 : \mathtt{hyp}\ F_1, \ldots, u_n : \mathtt{hyp}\ F_n$. We distinguish two variants of the assertion logic, the one without the cut rule is depicted first.

```
hyp    : form -> type.
conc   : form -> type.
ax     : hyp F -> conc F.
impr   : (hyp F -> conc G)
         -> conc (F ==> G).
impl   : conc F -> (hyp G -> conc H)
         -> (hyp (F ==> G) -> conc H).
forallr: ({x:tm T} conc (F x))
         -> conc (forall F).
foralll: {E: tm T} (hyp (F E) -> conc H)
         -> (hyp (forall F) -> conc H).
```

Analogously, every derivation of $F_1, \ldots, F_n \vdash^{\text{cut}} G$ can be adequately represented in LF as an object of type `conc*` G. We give the encoding of the cut rule, the others are obtained from the former by copying and replacing `conc` by `conc*`.

```
conc*  : form -> type.
cut    : {F: form} conc* F
         -> (hyp F -> conc* G)
         -> conc* G.
```

In the following two sections, we demonstrate the use of structural logical relations for the proof of weak normalization of the simply typed λ-calculus and the completeness of an equivalence checking algorithm.

4. Normalization of Simply Typed λ-Terms

Our first demonstration of a proof by structural logical relations takes place in the setting of the simply typed λ-calculus. We show that every simply typed λ-term is weakly normalizing, by which we mean that it can be reduced to a β-short and η-long normal form. This is different from the usual statement of weak normalization to η-short forms. We define two judgments for canonical and atomic forms of well-typed terms e^τ, written judgmentally as $e \Uparrow \tau$ and $e \Downarrow \tau$. Canonical forms are defined over the structure of τ and atomic forms are defined over the structure of e.

$$\frac{}{x \Downarrow \tau_1}\, u$$

$$\vdots$$

$$\frac{e \Downarrow \mathsf{o}}{e \Uparrow \mathsf{o}}\,\mathsf{can\_o} \qquad \frac{e \Uparrow \tau_2}{\mathsf{lam}\ x^{\tau_1}.\, e \Uparrow \tau_1 \Rightarrow \tau_2}\,\mathsf{can\_arr}^{x,u}$$

$$\frac{e_1 \Downarrow \tau_2 \Rightarrow \tau_1 \quad e_2 \Uparrow \tau_2}{\mathsf{app}\ e_1\ e_2 \Downarrow \tau_1}\,\mathsf{atm\_app}$$

Next we define three judgments $e \longrightarrow e'$, $e \longrightarrow^* e'$, and $e \overset{\text{whr}}{\rightarrow} e'$.

$$\frac{}{\mathsf{app}\ (\mathsf{lam}\ x^\tau.\, e_1)\ e_2 \longrightarrow e_1[e_2/x]}\,\mathsf{red\_beta}$$

$$\frac{}{e \longrightarrow \mathsf{lam}\ x^\tau.\mathsf{app}\ e\ x}\,\mathsf{exp\_eta}$$

$$\frac{e_1 \longrightarrow e_1'}{\mathsf{app}\ e_1\ e_2 \longrightarrow \mathsf{app}\ e_1'\ e_2}\,\mathsf{cong\_app1}$$

$$\frac{e_2 \longrightarrow e_2'}{\mathsf{app}\ e_1\ e_2 \longrightarrow \mathsf{app}\ e_1\ e_2'}\,\mathsf{cong\_app2}$$

$$\frac{e \longrightarrow e'}{\mathsf{lam}\ x^\tau.\, e \longrightarrow \mathsf{lam}\ x^\tau.\, e'}\,\mathsf{cong\_lam}^x$$

red_beta and exp_eta define the rules for β-reduction and η-expansion, respectively, and rules cong_app1, cong_app2, and cong_lam define the usual congruence rules. The following two rules describe the reflexive transitive closure of $\longrightarrow$.

$$\frac{}{e \longrightarrow^* e}\,\mathsf{refl} \qquad \frac{e_1 \longrightarrow e_2 \quad e_2 \longrightarrow^* e_3}{e_1 \longrightarrow^* e_3}\,\mathsf{trans}$$

Next, we define weak head reduction.

$$\frac{}{\mathsf{app}\ (\mathsf{lam}\ x^\tau.\, e_1)\ e_2 \overset{\text{whr}}{\rightarrow} e_1[e_2/x]}\,\mathsf{whr\_beta}$$

$$\frac{e_1 \overset{\text{whr}}{\rightarrow} e_1'}{\mathsf{app}\ e_1\ e_2 \overset{\text{whr}}{\rightarrow} \mathsf{app}\ e_1'\ e_2}\,\mathsf{whr\_cong}$$

whr_beta executes a β-reduction step, and whr_cong defines the congruence closure for weak-head reduction. As an example, we give an adequate encoding of the weak-head reduction judgments and rules in LF.

```
whr: tm T -> tm T -> type.
whr_beta: whr (app (lam E1) E2) (E1 E2).
whr_cong: whr E1 E1'
        -> whr (app E1 E2) (app E1' E2).
```

We give the LF declarations of the remaining judgments, without specifying the encoding of their rules. We will refer to them, when we specify lemmas and theorems in Twelf below. The type families

```
can : tm T -> type.
atm : tm T -> type.
red : tm T -> tm T -> type.
reds: tm T -> tm T -> type.
```

encode the judgments $e \Uparrow \tau$, $e \Downarrow \tau$, $e \longrightarrow e'$ and $e \longrightarrow^* e'$, respectively. The complete encoding of the judgment can be found at [SS08].

4.1. The Structural Logical Relation

We shall prove that for every derivation of e^τ, there exists a v^τ, s.t. $e \longrightarrow^* v$ and $v \Uparrow \tau$ via a unary structural logical relation. The challenge, however, is the choice of a predicate P. It may come as a surprise that it is sufficient to characterize the fact *that a term has a normal form* without making explicit what the normal form actually is or how to obtain it. This information, as it turns out, can be derived from the proof of P in the assertion logic. We extend the assertion logic by three families of atomic predicates $\mathsf{hc}^\tau(e)$, $\mathsf{ha}^\tau(e)$, and $\mathsf{wh}^\tau(e, e')$, which stand for e^τ has a canonical form, e^τ has an atomic form, and e^τ weak-head reduces to e'^τ, respectively. The corresponding proof rules are declared as follows.

$$\frac{\Delta \vdash \mathsf{ha}^{\tau_2 \Rightarrow \tau_1}(e_1) \quad \Delta \vdash \mathsf{hc}^{\tau_2}(e_2)}{\Delta \vdash \mathsf{ha}^{\tau_1}(\mathsf{app}\ e_1\ e_2)}\ \mathsf{ha\_app}$$

$$\frac{\Delta, x^{\tau_1}, \mathsf{ha}^{\tau_1}(x) \vdash \mathsf{hc}^{\tau_2}(\mathsf{app}\ e\ x)}{\Delta \vdash \mathsf{hc}^{\tau_1 \Rightarrow \tau_2}(e)}\ \mathsf{hc\_arr}$$

$$\frac{\Delta \vdash \mathsf{wh}^{\circ}(e, e') \quad \Delta \vdash \mathsf{hc}^{\circ}(e')}{\Delta \vdash \mathsf{hc}^{\circ}(e)}\ \mathsf{hc\_wh}$$

$$\frac{\Delta \vdash \mathsf{ha}^{\circ}(e)}{\Delta \vdash \mathsf{hc}^{\circ}(e)}\ \mathsf{hc\_atm}$$

$$\frac{}{\Delta \vdash \mathsf{wh}^{\tau}(\mathsf{app}\ (\mathsf{lam}\ x.\, e_1)\ e_2, e_1[e_2/x])}\ \mathsf{wh\_beta}$$

$$\frac{\Delta \vdash \mathsf{wh}^{\tau_2 \Rightarrow \tau_1}(e_1, e_1')}{\Delta \vdash \mathsf{wh}^{\tau_1}(\mathsf{app}\ e_1\ e_2, \mathsf{app}\ e_1'\ e_2)}\ \mathsf{wh\_cong}$$

The formulation of the sequent calculus with cut is extended in the same fashion. We formalize this in LF as follows.

```
hc: tm T -> form.
ha: tm T -> form.
wh: tm T -> tm T -> form.

ha_app : conc (ha E1) -> conc (hc E2)
         -> conc (ha (app E1 E2)).
hc_arr : ({x: tm T1} hyp (ha x)
          -> conc (hc (app E x)))
         -> conc (hc E).
hc_wh  : {E:tm o} conc (wh E E')
         -> conc (hc E')
         -> conc (hc E).
hc_atm : {E: tm o} conc (ha E)
         -> conc (hc E).
wh_beta: conc (wh (app (lam E1) E2) (E1 E2)).
wh_cong: conc (wh E1 E1')
         -> conc (wh (app E1 E2) (app E1' E2)).
```

The rules extending `conc*` are analogous. The next lemma and theorem illustrate how we can extract the canonical form together with a reduction trace from the cut-free derivation of $\vdash \mathsf{hc}^{\tau}(e)$. In Twelf, theorems are formalized as relations between universally and existentially quantified derivations. We give the relations as LF type families at the end of selected proofs. The role that each argument to the relation plays can always be read out of the informal statement of the theorem.

Lemma 4.1 (Congruence)

1. *If* $e \longrightarrow^* e'$ *then* $\mathsf{lam}\ x^{\tau}.\, e \longrightarrow^* \mathsf{lam}\ x^{\tau}.\, e'$.
2. *If* $e_1 \longrightarrow^* e_1'$ *and* $e_2 \longrightarrow^* e_2'$ *then* $\mathsf{app}\ e_1\ e_2 \longrightarrow^* \mathsf{app}\ e_1\ e_2'$.
3. *If* $e \overset{whr}{\rightarrow} e'$ *then* $e \longrightarrow e'$.

Proof: By straightforward structural inductions. □

Theorem 4.2 (Extraction) *All derivations in the sequent calculus are assumed to be cut-free. Let* $\Delta = x_1^{\tau_1}, \mathsf{ha}^{\tau_1}(x_1), \ldots, x_n^{\tau_n}, \mathsf{ha}^{\tau_n}(x_n)$.

1. *For all derivations of* $\Delta \vdash \mathsf{hc}^{\tau}(e)$ *there exist derivations of*

$$\frac{}{x_1 \Downarrow \tau_1}\, u_1 \quad \cdots \quad \frac{}{x_n \Downarrow \tau_n}\, u_n$$
$$\ddots \ \vdots \ \iddots$$
$$v \Uparrow \tau$$

   and $e \longrightarrow^* v$.

2. *For all derivations of* $\Delta \vdash \mathsf{ha}^{\tau}(e)$ *there exist derivations of*

$$\frac{}{x_1 \Downarrow \tau_1}\, u_1 \quad \cdots \quad \frac{}{x_n \Downarrow \tau_n}\, u_n$$
$$\ddots \ \vdots \ \iddots$$
$$v \Downarrow \tau$$

   and $e \longrightarrow^* v$.

3. *For all derivations of* $\Delta \vdash \mathsf{wh}^{\tau}(e, e')$ *there exists a derivation of* $e \overset{whr}{\rightarrow} e'$.

Proof: By mutual induction on the derivations of $\Delta \vdash \mathsf{hc}^{\tau}(e)$, $\Delta \vdash \mathsf{ha}^{\tau}(e)$, and $\Delta \vdash \mathsf{wh}^{\tau}(e, e')$. 1. follows from Lemma 4.1 (1), (3) and rules trans, exp_eta, and n1, n2. 2. follows from Lemma 4.1 (2) and rule atm_app. 3. follows from the application of rules whr_beta, whr_cong.

```
ext1 : conc (hc E)
       -> {V} can V -> reds E V -> type.
ext2 : conc (ha E)
       -> {V} atm V -> reds E V -> type.
ext3 : conc (wh E E') -> whr E V -> type.
```

□

We define the logical relation as a predicate in the assertion logic, by instantiating P (from the general definition of a unary logical relation from Section 2) with hc°.

Definition 4.3 (Logical relation)

$$\begin{array}{lcl} [\![\mathsf{o}]\!](e) & = & \mathsf{hc}^{\circ}(e) \\ [\![\tau_2 \Rightarrow \tau_1]\!](e) & = & \forall e_2^{\tau_2}.\, [\![\tau_2]\!](e_2) \supset [\![\tau_1]\!](\mathsf{app}\ e\ e_2) \end{array}$$

```
lr : {T:tp} (tm T -> form)-> type.
lr1: lr o (hc: tm o -> form).
lr2: lr T2 F -> lr T1 G
     -> lr (T2 => T1) ([e: tm (T2 => T1)]
           (forall [e_2:tm T2] (F e2)
              ==> (G (app e e2))))
```

4.2. Meta-Theory

All proofs in the remainder of this section are inductive over the structure of types, typing derivations, and proofs in the assertion logics. No definition of simultaneous substitutions is needed. No extension of the concept of logical relation to simultaneous substitutions is needed. And finally, no Kripke-style explicit contexts are needed in the definition of the logical relation. This information is already implicitly contained within the assertion logic proofs.

Lemma 4.4 (Escape) *1. For all types τ and assumptions Δ, there exists a derivation of $\Delta \overset{\text{cut}}{\vdash} \forall e^{\tau}.\, [\![\tau]\!](e) \supset \mathsf{hc}^{\tau}(e)$.*

2. For all types τ and assumptions Δ, there exists a derivation of $\Delta \overset{\text{cut}}{\vdash} \forall e^{\tau}.\, \mathsf{ha}^{\tau}(e) \supset [\![\tau]\!](e)$.

Proof: By mutual induction on τ.

Case: $\tau = \mathsf{o}$. Direct, by reasoning using Def. 4.3 and rules allR, impR, and ax.

Case: $\tau = \tau_2 \Rightarrow \tau_1$. By induction hypothesis, we obtain that for all Δ'

$$\Delta' \overset{\text{cut}}{\vdash} \forall e^{\tau_1}.\, [\![\tau_1]\!](e) \supset \mathsf{hc}^{\tau_1}(e) \quad (1)$$
$$\Delta' \overset{\text{cut}}{\vdash} \forall e^{\tau_1}.\, \mathsf{ha}^{\tau_1}(e) \supset [\![\tau_1]\!](e) \quad (2)$$
$$\Delta' \overset{\text{cut}}{\vdash} \forall e^{\tau_2}.\, [\![\tau_2]\!](e) \supset \mathsf{hc}^{\tau_2}(e) \quad (3)$$
$$\Delta' \overset{\text{cut}}{\vdash} \forall e^{\tau_2}.\, \mathsf{ha}^{\tau_2}(e) \supset [\![\tau_2]\!](e) \quad (4)$$

1. Follows by induction hypotheses (1) and (4), using Def. 4.3 and rules allR, impR, allL, impL, cut, and hc_arr.
2. Follows by induction hypotheses (2) and (3), using rules allR, impR, allL, cut, and ha_app.

```
esc1: {T:tp} {F:(tm T) -> form} lr T F
      -> (conc* (forall [e] F e
                   ==> hc e)) -> type.
esc2: {T:tp} {F:(tm T) -> form} lr T F
      -> (conc* (forall [e] ha e
                   ==> F e)) -> type.
```

□

Lemma 4.5 (Closure under weak head expansion) *For all types τ and assumptions Δ, there exists a derivation of $\Delta \overset{\text{cut}}{\vdash} \forall e^{\tau}.\forall e'^{\tau}.\, \mathsf{wh}^{\tau}(e, e') \supset [\![\tau]\!](e') \supset [\![\tau]\!](e)$ in the assertion logic.*

Proof: By induction on τ.

Case: $\tau = \mathsf{o}$. Direct, by reasoning using Def. 4.3 and rules allR, impR, ax, and hc_wh.

Case: $\tau = \tau_2 \Rightarrow \tau_1$. By induction hypothesis, we obtain that for all Δ'

$$\Delta' \overset{\text{cut}}{\vdash} \forall e^{\tau_1}.\forall e'^{\tau_1}.\, \mathsf{wh}^{\tau_1}(e, e') \supset [\![\tau_1]\!](e') \supset [\![\tau_1]\!](e)$$

The claim follows from Def. 4.3 and rules allR, impR, allL, impL, cut, and wh_cong.

```
cwhe: {T: tp} {F: tm T -> form} lr T F
      -> (conc* (forall [e] forall [e']
            wh e e' ==> F e' ==> F e))
      -> type.
```

□

Theorem 4.6 (Fundamental) *For all types τ and assumptions $\Delta = x_1^{\tau_1}, [\![\tau_1]\!](x_1), \ldots, x_n^{\tau_n}, [\![\tau_n]\!](x_n)$ and for all e^{τ} with free variables among $x_1 \ldots x_n$, there exists a derivation of $\Delta \overset{\text{cut}}{\vdash} [\![\tau]\!](e)$.*

Proof: By induction on the structure of $e : \tau$.

Case: $\tau = \tau_i, e = x_i$. Direct, by rule ax.

Case: $e = \mathsf{app}\ e_1\ e_2$. The claim follows from the induction hypotheses $\Delta \overset{\text{cut}}{\vdash} [\![\tau]\!](e_1)$ and $\Delta \overset{\text{cut}}{\vdash} [\![\tau]\!](e_2)$ using Def. 4.3 and rules ax, allL, impL.

Case: $e = \mathsf{lam}\ x^{\tau_{n+1}}.\, e'$. The claim follows from the induction hypothesis $\Delta, x_{n+1}^{\tau}, [\![\tau_{n+1}]\!](x) \overset{\text{cut}}{\vdash} [\![\tau']\!](e')$ using Def. 4.3, Lemma 4.5, and rule wh_beta.

```
fund: {E:tm T} lr T F
      -> conc* (F E) -> type.
```

□

In order to complete the proof, we need to appeal to a proof normalization technique for our assertion logic: cut-elimination. For any well-typed e^{τ}, after applying the Fundamental Theorem 4.6, followed by the Escape Lemma 4.4, followed by a few applications of the rules cut, forallI, and impI, we obtain a *cut-full* proof of $\overset{\text{cut}}{\vdash} \mathsf{hc}^{\tau}(e)$. The Extraction Theorem 4.2, however, requires the proof of $\vdash \mathsf{hc}^{\tau}(e)$ to be *cut-free*. It is this step where the true work of the structural logical relation proof takes place, as we need to eliminate all cuts in order to apply the Extraction Theorem.

Theorem 4.7 (Cut-elimination) *Let Δ denote assumptions. Any cut-full sequent derivation of $\Delta \overset{\text{cut}}{\vdash} F$ can be converted into $\Delta \vdash F$.*

```
ce : conc* F -> conc F -> type.
```

Proof: See: Pfenning [Pfe95]. The extra cases introduced by our atomic propositions all fall into the category of "right commutative conversions." □

The following theorem summarizes all the work we have done so far. It shows that the simply typed λ-calculus is weakly normalizing.

Theorem 4.8 (Weak normalization) *Let τ be a type and e^τ a (closed) well-typed expression; then there exists v^τ, s.t. $v \Uparrow \tau$ and $e \longrightarrow^* v$.*

```
thm : {E:tm T} {V:tm T} can V
      -> reds E V -> type.
```

5. Decidability of Equivalence Checking

The POPLmark challenge [ABF+05] aims to compare different proof systems according to their ability to represent and reason about the meta-theory of programming languages. The proof of soundness and completeness of the equivalence checking algorithm described in [Cra05] is a potential POPLmark challenge problem. Out of the two directions, the completeness proof is much more interesting, because it employs a binary logical relation. A formalization of this proof in nominal Isabelle was given recently [NU08]. In the remainder of this section, we demonstrate how this proof can be formalized in Twelf via binary structural logical relations. Since many of the judgments, axioms, and inference rules are so similar to the ones in the previous section, we have taken the liberty of reusing their names.

First, we formally define definitional equality on terms. We say that two terms e^τ and e'^τ are definitionally equivalent at a type τ, written judgmentally as $e \equiv e' : \tau$ and specified by

$$\frac{e_1 \equiv e_1' : \tau_2 \Rightarrow \tau_1 \quad e_2 \equiv e_2' : \tau_2}{\mathsf{app}\ e_1\ e_2 \equiv \mathsf{app}\ e_1'\ e_2' : \tau_1}\ \mathsf{qapp}$$

$$\frac{\begin{array}{c}\overline{}\,u \\ x \equiv y : \tau_1 \\ \vdots \\ e \equiv e' : \tau_2\end{array}}{\mathsf{lam}\ x.\ e \equiv \mathsf{lam}\ y.\ e' : \tau_1 \Rightarrow \tau_2}\ \mathsf{qabs}^{x,y,u}$$

$$\frac{\begin{array}{c}\overline{}\,u \\ x \equiv y : \tau_2 \\ \vdots \\ e_1 \equiv e_1' : \tau_1 \qquad e_2 \equiv e_2' : \tau_2\end{array}}{\mathsf{app}\ (\mathsf{lam}\ x.\ e_1)\ e_2 \equiv [e_2'/y]e_1' : \tau_1}\ \mathsf{beta}^{x,y,u}$$

$$\frac{\begin{array}{c}\overline{}\,u \\ x \equiv y : \tau_1 \\ \vdots \\ \mathsf{app}\ e\ x \equiv \mathsf{app}\ e'\ y : \tau_2\end{array}}{e \equiv e' : \tau_1 \Rightarrow \tau_2}\ \mathsf{qext}^{x,y,u}$$

$$\frac{e \equiv e' : \tau}{e' \equiv e : \tau}\ \mathsf{qsymm} \qquad \frac{e_1 \equiv e_2 : \tau \quad e_2 \equiv e_3 : \tau}{e_1 \equiv e_3 : \tau}\ \mathsf{qtrans}$$

We remark that this is not the exact definition of $e_1 \equiv e_2$ given in [Cra05]; however, any derivation of $e_1 \equiv e_2$ in Crary's sense can be easily mapped into a derivation of $e_1 \equiv e_2$ using the rules above. This has been formally proved in [SS08].

We sketch the implementation in LF by giving an encoding of judgments and theorems, but neither rules nor proofs. The judgment $e \equiv e' : \tau$ is represented as infix operator ==.

```
== : tm T -> tm T -> type.
                     %infix right 8 ==.
```

Second, we formally define the algorithm that decides the equality on terms. Recall the definition of weak head reduction from Section 4, for which we wrote $e \xrightarrow{\text{whr}} e'$. For its complement, which says that e^τ does not weak-head reduce to anything, we declare two judgments $e \overset{\text{whr}\downarrow}{\not\to}$ and $e \overset{\text{whr}\uparrow}{\not\to}$ mimicking the distinction of atomic and canonical forms. Their meaning is defined by the following set of rules.

$$\frac{}{\mathsf{lam}\ x.\ e \overset{\text{whr}\uparrow}{\not\to}}\ \mathsf{nw\_lam} \qquad \frac{e \overset{\text{whr}\downarrow}{\not\to}}{e \overset{\text{whr}\uparrow}{\not\to}}\ \mathsf{nw\_o}$$

$$\frac{}{x \overset{\text{whr}\downarrow}{\not\to}}\ \mathsf{nw\_var} \qquad \frac{e_1 \overset{\text{whr}\downarrow}{\not\to}}{\mathsf{app}\ e_1\ e_2 \overset{\text{whr}\downarrow}{\not\to}}\ \mathsf{nw\_app}$$

We define the weak head normalization portion of the algorithm to quasi normal forms as $e \rightsquigarrow e' : \tau$ below.

$$\frac{e_1 \xrightarrow{\text{whr}} e_2 : \tau \quad e_2 \rightsquigarrow e_3 : \tau}{e_1 \rightsquigarrow e_3 : \tau}\ \mathsf{qan\_red} \qquad \frac{e \overset{\text{whr}\uparrow}{\not\to}}{e \rightsquigarrow e : \tau}\ \mathsf{qan\_nor}$$

In LF, we declare the type families

```
nwC : tm T -> type.
nwA : tm T -> type.
qnf : tm T -> tm T -> type.
```

for the judgments $e \overset{\text{whr}\downarrow}{\not\to}$, $e \overset{\text{whr}\uparrow}{\not\to}$, and $e \rightsquigarrow e' : \tau$, respectively. Next we define the equivalence algorithm for two well-typed expressions e^τ and e'^τ via two judgments $e \sim e' \uparrow \tau$ and $e \sim e' \downarrow \tau$, whose completeness properties we will study in this section.

$$\frac{e_1 \rightsquigarrow e_1' \quad e_2 \rightsquigarrow e_2' \quad e_1' \sim e_2' \downarrow \mathsf{o}}{e_1 \sim e_2 \uparrow \mathsf{o}}\ \mathsf{qat\_base}$$

$$\frac{\dfrac{}{x \sim x \downarrow \tau}\,u \\ \vdots \\ \mathsf{app}\ e\ x \sim \mathsf{app}\ e'\ x \uparrow \tau_2}{e \sim e' \uparrow \tau_1 \Rightarrow \tau_2}\ \mathsf{qat\_arrow}^{x,u}$$

$$\frac{e_1 \sim e_1' \downarrow \tau_2 \Rightarrow \tau_1 \quad e_2 \sim e_2' \uparrow \tau_2}{\mathsf{app}\ e_1\ e_2 \sim \mathsf{app}\ e_1'\ e_2' \downarrow \tau_1}\ \mathsf{qap\_app}$$

We write

```
algC : tm T -> tm T -> type.
algA : tm T -> tm T -> type.
```

for $e \sim e' \uparrow \tau$ and $e \sim e' \downarrow \tau$, respectively.

5.1. The Structural Logical Relation

We shall prove that if $e \equiv e' : \tau$ then $e \sim e' \uparrow \tau$. Following the example in Section 4, we extend the assertion logic defined in Section 2 by binary versions of the predicates $\mathsf{ha}^\tau(e, e')$ and $\mathsf{hc}^\tau(e, e')$ at types τ, and the corresponding inference rules. As for weak-head reduction, we simply reuse the definition of the binary predicate $\mathsf{wh}^\tau(e, e')$ and specify the meaning of the new predicates by the following rules.

$$\frac{\Delta \vdash \mathsf{ha}^{\tau_2 \Rightarrow \tau_1}(e_1, e_1') \quad \Delta \vdash \mathsf{hc}^{\tau_2}(e_2, e_2')}{\Delta \vdash \mathsf{ha}^{\tau_1}(\mathsf{app}\ e_1\ e_2, \mathsf{app}\ e_1'\ e_2')}\ \mathsf{ha\_app}$$

$$\frac{\Delta, x_1^\tau, \mathsf{ha}^{\tau_1}(x, x) \vdash \mathsf{hc}^{\tau_2}(\mathsf{app}\ e\ x, \mathsf{app}\ e'\ x)}{\Delta \vdash \mathsf{hc}^{\tau_1 \Rightarrow \tau_2}(e, e')}\ \mathsf{hc\_arr}$$

$$\frac{\Delta \vdash \mathsf{wh}^{\mathsf{o}}(e_1, e_2) \quad \Delta \vdash \mathsf{hc}^{\mathsf{o}}(e_2, e')}{\Delta \vdash \mathsf{hc}^{\mathsf{o}}(e_1, e')}\ \mathsf{hc\_wh}$$

$$\frac{\Delta \vdash \mathsf{ha}^{\mathsf{o}}(e, e')}{\Delta \vdash \mathsf{hc}^{\mathsf{o}}(e, e')}\ \mathsf{hc\_atm}$$

We also give rules for symmetry and transitivity.

$$\frac{\Delta \vdash \mathsf{hc}^\tau(e', e)}{\Delta \vdash \mathsf{hc}^\tau(e, e')}\ \mathsf{hc\_sym}$$

$$\frac{\Delta \vdash \mathsf{hc}^\tau(e_1, e_2) \quad \Delta \vdash \mathsf{hc}^\tau(e_2, e_3)}{\Delta \vdash \mathsf{hc}^\tau(e_1, e_3)}\ \mathsf{hc\_tr}$$

In LF all of these rules are encoded as follows:

```
ha_app: conc (ha E1 E1') -> conc (hc E2 E2')
        -> conc (ha (app E1 E2) (app E1' E2')).
hc_arr: ({x: tm T} hyp (ha x x)
        -> conc (hc (app E x) (app E' x)))
        -> conc (hc E E').
hc_wh : conc (wh (E1 : tm o) E2)
        -> conc (hc E2 E')
        -> conc (hc E1 E').
hc_atm: conc (ha (E : tm o) E')
        -> conc (hc E E').
hc_sym: conc (hc E' E) -> conc (hc E E').
hc_tr : conc (hc E1 E2) -> conc (hc E2 E3)
        -> conc (hc E1 E3).
```

The rules for `conc*` are defined analogously. The next lemma and theorem illustrate how we can extract the execution trace of the equivalence checking algorithm from a cut-free derivation of $\mathsf{hc}^\tau(e, e')$. The formulation of the extraction theorem and its proof follow roughly the same structure as Theorem 4.2.

Lemma 5.1 (Congruence)

1. *If* $e_1' \sim e_2 \uparrow \tau$ *and* $e_1 \overset{whr}{\longrightarrow} e_1'$ *then* $e_1 \sim e_2 \uparrow \tau$.
2. *If* $e_1 \sim e_2 \downarrow \tau$ *then* $e_1 \overset{whr\downarrow}{\not\longrightarrow}$ *and* $e_2 \overset{whr\downarrow}{\not\longrightarrow}$.
3. *If* $e \sim e' \downarrow \tau$ *then* $e' \sim e \downarrow \tau$.
4. *If* $e \sim e' \uparrow \tau$ *then* $e' \sim e \uparrow \tau$.
5. *If* $e \sim e' \downarrow \tau$ *and* $e' \sim e'' \downarrow \tau$ *then* $e \sim e'' \downarrow \tau$.
6. *If* $e \sim e' \uparrow \tau$ *and* $e' \sim e'' \uparrow \tau$ *then* $e \sim e'' \uparrow \tau$.

Proof: By straightforward structural inductions. □

Theorem 5.2 (Extraction) *All derivations in the sequent calculus are assumed to be cut-free. Let*

$$\Delta = x_1^{\tau_1}, \mathsf{ha}^{\tau_1}(x_1, x_1), \ldots, x_n :^{\tau_n}, \mathsf{ha}^{\tau_n}(x_n, x_n).$$

1. *For all derivations of* $\Delta \vdash \mathsf{hc}^\tau(e, e')$ *there exists a derivation of*

$$\frac{\dfrac{}{x_1 \sim x_1 \downarrow \tau_1}\,u_1 \quad \cdots \quad \dfrac{}{x_n \sim x_n \downarrow \tau_n}\,u_n}{e \sim e' \uparrow \tau.}$$

2. *For all derivations of* $\Delta \vdash \mathsf{ha}^\tau(e, e')$ *there exists a derivation of*

$$\frac{\dfrac{}{x_1 \sim x_1 \downarrow \tau_1}\,u_1 \quad \cdots \quad \dfrac{}{x_n \sim x_n \downarrow \tau_n}\,u_n}{e \sim e' \downarrow \tau.}$$

Proof: By induction on the cut-free assertion logic proofs, using Lemma 5.1.

```
ext1 : conc (hc E E')
       -> algC E E' -> type.
ext2 : conc (ha E E')
       -> algA E E' -> type.
```

□

We define the logical relation as a predicate in the assertion logic by instantiating P (from the general definition of a binary logical relation from Section 2) with hc^{o}.

Definition 5.3 (Binary logical relation)

$$\begin{array}{lcl} [\![\mathsf{o}]\!](e, e') & = & \mathsf{hc}^{\mathsf{o}}(e, e') \\ [\![\tau_2 \Rightarrow \tau_1]\!](e, e') & = & \forall e_2^{\tau_2}.\, \forall e_2'^{\tau_2}.\, [\![\tau_2]\!](e_2, e_2') \\ & & \quad \supset [\![\tau_1]\!](\mathsf{app}\ e\ e_2, \mathsf{app}\ e'\ e_2') \end{array}$$

```
lr  : {T:tp} (tm T -> tm T -> form)-> type.
lr_o: lr o hc.
lr_a: lr T2 F -> lr T1 G
      -> lr (T2 => T1) ([e] [e']
         forall [e2] forall [e2'] (F e2 e2')
            ==> (G (app e e2) (app e' e2')))
```

5.2. Meta-Theory

Next, we extend all theorems from the previous section to the binary logical relation.

Lemma 5.4 (Escape) *1. For all types τ and assumptions Δ, there exists a derivation of $\Delta \overset{\text{cut}}{\vdash} \forall e^{\tau}.\, \forall e'^{\tau}.\, [\![\tau]\!](e, e') \supset \mathsf{hc}^{\tau}(e, e')$.*

2. For all types τ and assumptions Δ, there exists a derivation of $\Delta \overset{\text{cut}}{\vdash} \forall e^{\tau}.\, \forall e^{\tau}.\, \mathsf{ha}^{\tau}(e, e') \supset [\![\tau]\!](e, e')$.

Proof: By induction on τ. The proof follows exactly the same structure as the proof of Theorem 4.4, and is hence omitted.

```
esc1: {T:tp} {F:(tm T) -> (tm T) -> form}
       lr T F
       -> (conc* (forall [e] forall [e']
         F e e' ==> hc e e')) -> type.
esc2: {T:tp} {F:(tm T) -> (tm T) -> form}
       lr T F
       -> (conc* (forall [e] forall [e']
         ha e e' ==> F e e')) -> type.
```

□

Lemma 5.5 (Closure under weak head expansion) *For all types τ and assumptions Δ , there exists a derivation of*

$$\begin{array}{l} \Delta \overset{\text{cut}}{\vdash} \forall e_1^{\tau}.\, \forall e_2^{\tau}.\, \forall e'^{\tau}. \\ \qquad \mathsf{wh}^{\tau}(e_1, e_2) \supset [\![\tau]\!](e_2, e') \supset [\![\tau]\!](e_1, e') \end{array}$$

and

$$\begin{array}{l} \Delta \overset{\text{cut}}{\vdash} \forall e^{\tau}.\, \forall e_1'^{\tau}.\, \forall e_2'^{\tau}. \\ \qquad \mathsf{wh}^{\tau}(e_1', e_2') \supset [\![\tau]\!](e, e_2') \supset [\![\tau]\!](e, e_1') \end{array}$$

in the assertion logic.

Proof: By induction on τ. Again, the proof follows exactly the same structure as the proof of Lemma 4.5.

```
cwhe: {T: tp} {F: tm T -> tm T -> form}
     lr T F
     -> (conc* (forall [e1] forall [e2]
           forall [e'] wh e1 e2 ==>
                  F e2 e' ==> F e1 e'))
     -> (conc* (forall [e] forall [e1']
           forall [e2'] wh e1' e2' ==>
                  F e e2' ==> F e e1'))
     -> type.
```

□

Lemma 5.6 (Symmetry) *For all types τ, we show that*

$$\Delta \overset{\text{cut}}{\vdash} \forall e^{\tau}.\, \forall e'^{\tau}.\, [\![\tau]\!](e, e') \supset [\![\tau]\!](e', e).$$

Proof: By induction on τ, using Def. 5.3, and rules allR, impR, ax, allL, impL, hc_sym, and cut.

```
sym: lr T F
     -> conc* (forall[e] forall [e']
         F e e' ==> F e' e) -> type.
```

□

Lemma 5.7 (Transitivity) *For all types τ, we show that*

$$\begin{array}{l} \Delta \vdash \forall e^{\tau}.\, \forall e'^{\tau}.\, \forall e''^{\tau}. \\ \qquad [\![\tau]\!](e, e') \supset [\![\tau]\!](e', e'') \supset [\![\tau]\!](e, e''). \end{array}$$

Proof: By induction on τ using Def. 5.3, Lemma 5.6, and rules ax, hc_trans, allR, impR, allL, impL, and cut.

```
tr: lr T F
    -> conc* (forall [e] forall [e']
              forall [e''] F e e' ==>
                F e' e'' ==> F e e'')
    -> type.
```

□

Theorem 5.8 (Fundamental) *For all types τ and assumptions*

$$\Delta = x_1^{\tau_1}, y_1^{\tau_1}, [\![\tau_1]\!](x_1, y_1), \ldots, x_n^{\tau_n}, y_n^{\tau_n}, [\![\tau_n]\!](x_n, x_n)$$

and for all e with free variables among $x_1 \ldots x_n$ and e' with free variables among $y_1 \ldots y_n$. For every derivation of

$$\dfrac{}{x_1 \equiv y_1 : \tau_1}\, u_1 \quad \cdots \quad \dfrac{}{x_n \equiv y_n : \tau_n}\, u_n$$
$$\ddots \quad \vdots \quad \iddots$$
$$e \equiv e' : \tau$$

there exists a derivation of $\Delta \overset{\text{cut}}{\vdash} [\![\tau]\!](e, e')$.

Proof: By structural induction on $e \equiv e' : \tau$ using Def. 5.3, Lemma 5.5, Lemma 5.6, Lemma 5.7, and rules ax, allL, allR, impR, impL, cut and wh_beta.

```
fund: {E:tm T}{E'} E == E' -> lr T F
      -> conc* (F E E') -> type.
```

□

Theorem 5.9 (Cut-elimination) *Let Δ denote assumptions. Any cut-full sequent derivation of $\Delta \overset{\text{cut}}{\vdash} F$ can be converted into $\Delta \vdash F$.*

Proof: See: Pfenning [Pfe95]. The extra cases introduced by our atomic propositions all fall into the category of "right commutative conversions."

```
ce : conc* F -> conc F -> type.
```

□

The following theorem summarizes all the work we have done so far. It shows that convertible terms considered equal, can always been recognized as such.

Theorem 5.10 (Completeness) *For every derivation of $e \equiv e' : \tau$ there exists a derivation of $e \sim e' \uparrow \tau$.*

Proof:

$e \equiv e' : \tau$	by assumption
$\overset{\text{cut}}{\vdash} [\![\tau]\!](e, e')$	by Theorem 5.8
$\overset{\text{cut}}{\vdash} \forall e^{\tau}.\, \forall e'^{\tau}.\, [\![\tau]\!](e, e') \supset \mathsf{hc}^{\tau}(e, e')$	by Lemma 5.4
$\overset{\text{cut}}{\vdash} \mathsf{hc}^{\tau}(e, e')$	by ax, allL, impE
$\vdash \mathsf{hc}^{\tau}(e, e')$	by Theorem 5.9
$e \sim e' \uparrow \tau$	by Theorem 5.2

```
thm: {E:tm T} {E':tm T} eq E E'
     -> algC E E' -> type.
```

□

All proofs presented in this paper are verified by Twelf [SS08].

6. Cut-Elimination

Gödel's second incompleteness theorem tells us that no consistent logic can prove its own consistency, provided the logic is powerful enough to express some very basic coding machinery. Through the lens of the Curry-Howard isomorphism, any typed λ-calculus can be viewed as a logic. Both of the example theorems in this paper, weak normalization and the completeness of a decidable equivalence checking algorithm, imply the consistency of the simply typed λ-calculus when viewed as a logic. This is typical: most theorems that are proven using logical relations are essentially consistency theorems. Thus, it will be impossible to completely formalize any such theorem about a λ-calculus whose proof-theoretic strength exceeds the proof-theoretic strength of the meta-logic.

Recall that, after appealing to the Fundamental Theorem 4.6 and Escape Lemma 4.4 in the proof of Theorem 4.8, we obtain a cut-full derivation of $\overset{\text{cut}}{\vdash} \mathsf{hc}^{\tau}(e)$. We could have stopped here if we were certain that this were a meaningful witness to the normalization of e; i.e. if we trusted the consistency of the assertion logic. To be sure, we prove cut-elimination and produce a real witness from the cut-free derivation $\vdash \mathsf{hc}^{\tau}(e)$. In general, if the meta-logic is proof-theoretically stronger than the assertion-logic, then it should be possible to prove cut-elimination explicitly.

Our example proofs by structural logical relations reduce the validity of a consistency theorem to the consistency of the assertion logic. Because these reductions use only primitive-recursive machinery, the proof-theoretic strength of the assertion logic must be at least as great as the proof-theoretic strength of the λ-calculus that we are reasoning about. Thus, to scale our examples to Gödel's T, the assertion logic should provide some notion of induction; to System F, the assertion logic should provide some notion of second-order quantification.

It is worth mentioning that logical consistency is usually proven using either model-theoretic machinery, transfinite induction, or logical relations. Model theoretic machinery is justified by the assumption that set-theory is consistent; transfinite induction is justified by the assumption that a particular ordinal is well founded; a typical logical relations argument is justified by appealing implicitly to the validity of the logical connectives used to define the relation. Structural logical relations bring these assumptions to the surface.

7. Conclusion

In this paper we describe the technique of structural logical relations and show how to use it to prove interesting theorems, such as the completeness of the equivalence checking algorithm for the simply typed λ-calculus.

Structural logical relations work well when formalizing proofs by logical relations in proof assistants with a proof-theoretically weak meta-logic but good representational support for working with complex data structures, such as, for example, proof derivations in a sequent calculus.

We do not claim to be the first to characterize a logical relation in terms of the provability of a logical predicate. Indeed, for informal proofs, the idea goes back to Tait's original paper [Tai67]. However, to our knowledge, this is the first work where we separate the roles of the assertion logic and the meta-logic for the purpose of formalization. In proof assistants with strong induction principles, for example Lego or Coq, logical relations are formalized via inductive types [Alt93, BW97, DX06], and reasoned about using elimination forms. Structural logical relations provide not only an alternative, but they make explicit the logical foundations a particular proof is based on.

Proofs by logical relations are popular in large part because they tend to scale well. We believe that our technique preserves this property. For example, in Section 4 and 5 we successfully apply our technique to normalization proofs about the simply typed λ-calculus. Further experiments, such as normalization properties of the monadic λ-calculus (inspired by [LS05]), and about Gödel's T, have shown that the basic structure of the proof remains unchanged but the assertion logic needs to be suitably extended by new inference rules.

Closely related to our work is that of Andreas Abel [Abe08]. He also proves weak normalization directly in Twelf, albeit only for the simply typed λ-calculus. Though he does not use a logical relation, there are striking similarities between the structure of his proof and ours, which should be investigated further.

Acknowledgments: We would like to thank Frank Pfenning for pointing out this interesting problem many years ago, and we would also like to thank Andrzej Filinski and Alberto Momigliano for many discussions and comments on earlier drafts of this paper.

References

[Abe08] Andreas Abel. Normalization for the simply-typed lambda-calculus in Twelf. In Carsten Schürmann, editor, *Proceedings of the Fourth International Workshop on Logical Frameworks and Meta-Languages*, pages 3 – 16. Elsevier, ENTCS 199, February 2008.

[ABF+05] B. Aydemir, A. Bohannon, M. Fairbairn, J. Foster, B. Pierce, P. Sewell, D. Vytiniotis, G. Washburn, S. Weirich, and S. Zdancewic. Mechanized metatheory for the masses: The POPLmark challenge. In Joe Hurd and Tom Melham, editors, *Proceedings of the Eighteenth International Conference on Theorem Proving in Higher Order Logics*, pages 50–65, Oxford, UK, August 2005. Springer Verlag, LNCS 3603.

[Alt93] Thorsten Altenkirch. A formalization of the strong normalization proof for System F in LEGO. In M. Bezem and J.F. Groote, editors, *Proceedings of the International Conference on Typed Lambda Calculi and Applications*, pages 13–28, Utrecht, The Netherlands, March 1993. Springer-Verlag LNCS 664.

[BW97] B. Barras and B. Werner. Coq in Coq. Soumis, 1997.

[Cra05] Karl Crary. Logical relations and a case study in equivalence checking. In Benjamin C. Pierce, editor, *Advanced Topics in Types and Programming Languages*, pages 223–244. MIT Press, 2005.

[DX06] Kevin Donnelly and Hongwei Xi. A formalization of strong normalization for simply-typed lambda calculus and System F. In Alberto Momigliano and Brigitte Pientka, editors, *Proceedings of Workshop on Logical Frameworks and Meta-Languages: Theory and Practice*, pages 109–125, Seattle, WA, August 2006. Elsevier, ENTCS 174, Issue 5.

[Gon05] Georges Gonthier. A computer-checked proof of the four colour theorem. unpublished, 2005.

[Har96] John Harrison. HOL Light: A tutorial introduction. In Mandayam Srivas and Albert Camilleri, editors, *Proceedings of the First International Conference on Formal Methods in Computer-Aided Design*, pages 265–269, 1996.

[HHP93] Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery*, 40(1):143–184, January 1993.

[LCH07] Daniel K. Lee, Karl Crary, and Robert Harper. Towards a mechanized metatheory of standard ML. In *Proceedings of the 34th Annual Symposium on Principles of Programming Languages*, pages 173–184, New York, NY, USA, 2007. ACM Press.

[LS05] Sam Lindley and Ian Stark. Reducibility and $\top\top$-lifting for computation types. In Pawel Urzyczyn, editor, *Proceedings of the*

Seventh International Conference of Typed Lambda Calculi and Applications, pages 262–277, Nara, Japan, April 2005. Springer-Verlag, LNCS 3461.

[MM97] Raymond McDowell and Dale Miller. A logic for reasoning with higher-order abstract syntax: An extended abstract. In Glynn Winskel, editor, *Proceedings of the Twelfth Annual Symposium on Logic in Computer Science*, pages 434–445, Warsaw, Poland, June 1997. IEEE Computer Society Press.

[Nec97] George C. Necula. Proof-carrying code. In Neil D. Jones, editor, *Proceedings of the 24th Annual Symposium on Principles of Programming Languages*, pages 106–119, Paris, France, January 1997. ACM Press.

[NU08] Julien Narboux and Christian Urban. Formalising in Nominal Isabelle Crary's completeness proof for equivalence checking. In Brigitte Pientka and Carsten Schürmann, editors, *Proceedings of the Second International Workshop on Logical Frameworks and Meta-Languages*, pages 3–18. Elsevier, ENTCS 196, January 2008.

[Pfe92] Frank Pfenning. Computation and deduction. Unpublished lecture notes, 277 pp. Revised May 1994, April 1996, May 1992.

[Pfe95] Frank Pfenning. Structural cut elimination. In D. Kozen, editor, *Proceedings of the Tenth Annual Symposium on Logic in Computer Science*, pages 156–166, San Diego, California, June 1995. IEEE Computer Society Press.

[SS08] Jeffrey Sarnat and Carsten Schürmann. Structural logical relations. Project page available at `http://www.twelf.org/slr`, January 2008.

[Tai67] W. W. Tait. Intensional interpretation of functionals of finite type I. *Journal of Symbolic Logic*, 32:198–212, 1967.