Weak Normalization for the Simply-Typed Lambda-Calculus in Twelf

(Extended Abstract)

Andreas Abel

Department of Computer Science, Chalmers University of Technology
Rännvägen 6, SWE-41296 Göteborg, Sweden

Abstract. Weak normalization for the simply-typed λ-calculus is proven in Twelf, an implementation of the Edinburgh Logical Framework. Since due to proof-theoretical restrictions Twelf Tait's computability method does not seem to be directly usable, a combinatorical proof is adapted and formalized instead.

1 Introduction

Twelf is an implementation of the Edinburgh Logical Framework which supports reasoning in full higher-order abstract syntax; therefore it is an ideal candidate for reasoning comfortably about properties of prototypical programming languages with binding. Previous work has focused on properties like subject reduction, confluence, compiler correctness. Even cut elimination for various sequent calculi has been proven successfully, but until recently, there were no formalized proofs of normalization in Twelf. The reason might be that normalization is typically proven by Tait's method, which cannot be applied directly in Twelf. This work explains why Tait's method is at least not directly applicable and provides a combinatorical proof for the simply-typed lambda-calculus.

2 Twelf

The Edinburgh Logical Framework (LF[1]) [HHP93,HP00] is a dependently-typed lambda-calculus with type families and $\beta\eta$-equality, but no polymorphism, inductive data types or recursion. Expressions are divided into three syntactic

[1] This is not to be confused with Martin-Löf's framework for dependent type theory, which is also abbreviated by LF.

classes: kinds, types and terms, generated by the following grammar.

$$
\begin{array}{lll}
\mathbb{K} & ::= \mathtt{type} & \text{kind of types} \\
 & \mid \{\mathbb{X}\!:\!\mathbb{A}\}\mathbb{K} & \text{dependent function kind} \\[4pt]
\mathbb{A} & ::= \mathbb{F}\ \mathbb{M}_1 \ldots \mathbb{M}_n & \text{base type} \\
 & \mid \{\mathbb{X}\!:\!\mathbb{A}\}\mathbb{A} & \text{dependent function type} \\
 & \mid \mathbb{A} \to \mathbb{A} & \text{non-dependent function type} \\[4pt]
\mathbb{M} & ::= \mathbb{C} & \text{term constant} \\
 & \mid \mathbb{X} & \text{term variable} \\
 & \mid [\mathbb{X}\!:\!\mathbb{A}]\mathbb{M} & \text{term abstraction} \\
 & \mid \mathbb{M}\,\mathbb{M} & \text{term application}
\end{array}
$$

Herein, the meta variable $\mathbb{X}$ ranges over a countably infinte set of variable identifiers, while $\mathbb{F}$ resp. $\mathbb{C}$ range over type-family resp. term constants provided in a signature Σ. Note that neither a type nor a kind can depend on a type; consequently, abstraction is missing on the type level [Pfe01, p. 1124].

The framework comes with judgements for typing $\mathbb{M} : \mathbb{A}$, kinding $\mathbb{A} : \mathbb{K}$ and wellformedness of kinds $\mathbb{K}$ *kind* plus $\beta\eta$-equality on for terms, types and kinds [HP00]. An object theory can be described in the framework by providing a suitable signature Σ which adds kinded type family constants $\mathbb{F} : \mathbb{K}$ and typed term constants $\mathbb{C} : \mathbb{A}$.

Syntax.

$$
\begin{array}{lcll}
r, s, t, u & ::= & x \mid \lambda x.t \mid r\,s & \text{untyped terms} \\
A, B, C & ::= & * \mid A \to B & \text{simple types} \\
\Gamma & ::= & \diamond \mid \Gamma,\, x\!:\!A & \text{contexts}
\end{array}
$$

Type assignment $\Gamma \vdash t : A$.

$$
\frac{(x\!:\!A) \in \Gamma}{\Gamma \vdash x : A}\ \mathtt{of\_var}
$$

$$
\frac{\Gamma,\, x\!:\!A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B}\ \mathtt{of\_lam} \qquad \frac{\Gamma \vdash r : A \to B \qquad \Gamma \vdash s : A}{\Gamma \vdash r\,s : B}\ \mathtt{of\_app}
$$

Weak head reduction $t \longrightarrow_{\mathsf{w}} t'$.

$$
\frac{}{(\lambda x.t)\, s \longrightarrow_{\mathsf{w}} [s/x]t}\ \mathtt{beta} \qquad \frac{r \longrightarrow_{\mathsf{w}} r'}{r\,s \longrightarrow_{\mathsf{w}} r'\,s}\ \mathtt{appl}
$$

Fig. 1. Simply-typed λ-calculus and weak head reduction.

Twelf [PS99] is an implementation of LF whose most fundamental task is to check typing (and kinding) of a user given signature Σ, usually provided as a set of ASCII files. Symbols reserved for the framework are the following.

```
: . ( ) [ ] { } -> type
```

All others can be used to denote entities in the object theories.

In the remainder of this section we show how to represent the simply-typed λ-calculus with weak head reduction, as specified in Figure 1 in Twelf. Untyped lambda terms t can be represented by one type family constant `tm` and two term constants:

```
tm     : type.
lam    : (tm -> tm) -> tm.
app    : tm -> tm -> tm.
```

The lack of a construct for variables is due to the use of higher-order abstract syntax: object variables are represented by variables of the framework, e. g., in the code for the twice function:

```
twice = lam [f:tm] lam [x:tm] app f (app f x).
```

A more detailed explanation of higher-order encodings has been given by Schürmann [Sch00, p. 20ff]. Simple types A can be generated from a nullary constant `*` for some base type and a binary constant `=>`, used infix, for function type formation.

```
ty     : type.
*      : ty.
=>     : ty -> ty -> ty.
```

Type assignment for untyped terms, $\Gamma \vdash t : A$, can also be represented by two constants, one for function introduction and one for function elimination. Note that in Twelf syntax, the types of new constants may contain free variables (captial letters), which are regarded as universally quantified on the outside.

```
of     : tm -> ty -> type.
of_lam : ({x:tm} x of A -> (T x) of B)
           -> (lam [x:tm] T x) of (A => B).
of_app : R of (A => B) -> S of A -> (app R S) of B.
```

Again, there is no separate rule for the typing of variables, instead it is part of the rule for abstraction. The premise of rule `of_lam` is to be read as:

> Consider a temporary extension of the signature by a fresh constant `x:tm` and assume `x of A`. Then `(T x) of B` holds.

This adds a *dynamical* typing rule `x of A` for each new variable `x` instead of inserting a typing hypothesis $x : A$ into the typing context Γ. Hence, we do not explicitely encode Γ, but let the framework handle the typing hypotheses.

Similar to the typing relation we can represent weak head reduction $t \longrightarrow_{\mathsf{w}} t'$, which eliminates the head (resp. key) redex in term t but does not step under a binding.

```
-->w  : tm -> tm -> type.
beta  : app (lam T) S -->w T S.
appl  : R -->w R' -> app R S -->w app R' S.
```

Lemma 1 (Weak head reduction is closed under substitution).
If $t \longrightarrow_{\mathsf{w}} t'$ *then* $[u/y]t \longrightarrow_{\mathsf{w}} [u/y]t'$.

How do we represent theorems like Lemma 1? Twelf's internal logic is constructive, therefore the lemma must be interpreted constructively: Given a derivation $\mathcal{P}$ of $t \longrightarrow_{\mathsf{w}} t'$ and a term u, we can construct a derivation $\mathcal{P}'$ of $[u/y]t \longrightarrow_{\mathsf{w}} [u/y]t'$. The lemma itself is a relation between derivations, and, thus, via the propositions-as-types paradigm, just another type family.

```
subst_red : {U:tm} ({y:tm} T y -->w T' y)
                              -> T U -->w T' U -> type.
```

Note that in HOAS, substitution $[u/y]t$ is simply expressed as application `T U` for context `T : tm -> tm`.

Proof of Lemma 1. By induction on the derivation of $t \longrightarrow_{\mathsf{w}} t'$.

– Case $(\lambda x.t)\, s \longrightarrow_{\mathsf{w}} [s/x]t$. W.l.o.g. $x \neq y$ and x not free in u. Then,

$$\begin{array}{rcl} [u/y]((\lambda x.t)\, s) & = & (\lambda x.[u/y]t)\ [u/y]s \\ & \longrightarrow_{\mathsf{w}} & [[u/y]s/x][u/y]t \ = [u/y][s/x]t. \end{array}$$

– Case $r\, s \longrightarrow_{\mathsf{w}} r'\, s$ with $r \longrightarrow_{\mathsf{w}} r'$. By induction hypothesis, $[u/y]r \longrightarrow_{\mathsf{w}} [u/y]r'$. Hence,

$$\begin{array}{rcl} [u/y](r\, s) & = & ([u/y]r)\ ([u/y]s) \\ & \longrightarrow_{\mathsf{w}} & ([u/y]r')\ ([u/y]s) = [u/y](r'\, s) \end{array}$$

$\square$

The Twelf representation of this proof is a logic program which computes a derivation $\mathcal{P}' :: [u/y]t \longrightarrow_{\mathsf{w}} [u/y]t'$ from term u and derivation $\mathcal{P} :: t \longrightarrow_{\mathsf{w}} t'$. The `%mode` statement marks the first two arguments of type familiy `subst_red` as inputs (+) and the third as output (-). The base case of the induction is given by the constants `subst_red_beta` and the step case, which appeals to the induction hypothesis, by `subst_red_appl`. The types of these constants can be viewed as clauses similiar to the ones in PROLOG. Note that most of the tedious computations of the paper proof are hidden in the argument types of `subst_red` which can be infered by Twelf.

```
%mode subst_red +U +P -P'.
subst_red_beta: subst_red U ([y] beta) beta.
subst_red_appl: subst_red U ([y] appl (P y)) (appl P')
                 <- subst_red U P P'.
%terminates P (subst_red _ P _).
```

A logic program in Twelf corresponds to a partial function from inputs to outputs as specified by the mode declaration. Since only total functions correspond to valid inductive proofs we must ensures that the defined function *terminates* on all inputs and *covers* all possible cases. Brigitte Pientka [Pie01] contributed a termination checker which is invoked by the `%terminates` pragma. In our case, the second argument `P` decreases structurally in each recursive call. Case coverage is ensured by an algorithm by Pfenning and Schürmann [SP03] whose implementation is not yet stable. In the following, we will deliberately ignore the issue of coverage.

3 A Formalized Proof of Weak Normalization

In this section, we present a combinatorical proof of weak normalization for the simply-typed lambda-calculus. It is similiar to the textbook proof in Girard, Lafont and Taylor [GLT89, Ch. 4], but avoiding reasoning with numbers altogether. In fact, we follow closely the very syntactical presentation of Joachimski and Matthes [JM03], which has also been implemented in Isabelle/Isar by Nipkow and Berghofer [Ber03]. The main obstacle to a direct formalization in Twelf is the use of a vector notation for terms by Joachimski and Matthes, which allows them to reason on a high level in some cases. In this section, we will see a "devectorized" version of their proof which can be outlined as follows:

1. Define an inductive relation $t \Downarrow A$.
2. Prove that for every term $t : A$ the relation $t \Downarrow A$ holds.
3. Show that every term in the relation is weakly normalizing.

3.1 Inductive Characterization of Weak Normalization

Inductive characterizations of normalization go back to Goguen [Gog95] and van Raamsdonk and Severi [vRS95,vRSSX99]. We introduce a relation $\Gamma \vdash t \Downarrow A$ which stipulates the t is weakly normalizing of type A, and an auxiliary relation $\Gamma \vdash t \downarrow^x A$ which additionally claims that $t = x\,\boldsymbol{s}$ for some sequence of terms $\boldsymbol{s}$, i.e., t is neutral and head-redex free.

$$\frac{(x\!:\!A) \in \Gamma}{\Gamma \vdash x \downarrow^x A} \qquad \frac{\Gamma \vdash r \downarrow^x A \to B \qquad \Gamma \vdash s \Downarrow A}{\Gamma \vdash r\,s \downarrow^x B}\ \texttt{wne\_app} \qquad \frac{\Gamma \vdash r \downarrow^x A}{\Gamma \vdash r \Downarrow A}\ \texttt{wn\_ne}$$

$$\frac{\Gamma, x\!:\!A \vdash t \Downarrow B}{\Gamma \vdash \lambda x.t \Downarrow A \to B}\ \texttt{wn\_lam} \qquad \frac{r \longrightarrow_{\mathsf{w}} r' \qquad \Gamma \vdash r' \Downarrow A}{\Gamma \vdash r \Downarrow A}\ \texttt{wn\_exp}$$

The Twelf representation is similar to the typing relation: Again, Γ and the hypothesis rule are indirectly represented in rule `of_lam`.

```
wne     : tm -> ty -> tm -> type.
wn      : tm -> ty -> type.

wne_app : wne R (A => B) X -> wn S A -> wne (app R S) B X.
wn_ne   : {X:tm} wne R A X -> wn R A.
wn_lam  : ({x:tm} wne x A x -> wn (T x) B)
            -> wn (lam T) (A => B).
wn_exp  : R -->w R' -> wn R' A -> wn R A.
```

3.2 Closure under Application and Substitution

To show that each typed term $t : A$ is in the relation $t \Downarrow A$, we will proceed by induction on the typing derivation. Difficult is the case for an application $(\lambda x.r)\,s$: it can only be shown to be in the relation by rule `wn_exp`, which requires us to prove that $[s/x]r$ is in the relation. If x is head variable of r, substitution might create new redexes. In this case, however, we can argue that the type of r is a smaller type than the one of s. These prelimiary thoughts lead to the following lemma.

Lemma 2 (Application and Substitution). *Let $\mathcal{D} :: \Gamma \vdash s \Downarrow A$.*

1. *If $\mathcal{E} :: \Gamma \vdash r \Downarrow A \to C$ then $\Gamma \vdash r\,s \Downarrow C$.*
2. *If $\mathcal{E} :: \Gamma, x\!:\!A \vdash t \Downarrow C$, then $\Gamma \vdash [s/x]t \Downarrow C$.*
3. *If $\mathcal{E} :: \Gamma, x\!:\!A \vdash t \downarrow^x C$, then $\Gamma \vdash [s/x]t \Downarrow C$ and C is a part of A.*
4. *If $\mathcal{E} :: \Gamma, x\!:\!A \vdash t \downarrow^y C$ with $x \neq y$, then $\Gamma \vdash [s/x]t \downarrow^y C$.*

In Twelf, the lemma is represented by four type families. The invariant that C is a subexpression of A will be expressed via a `%reduces` statement later, which makes is necessary to make type C an explicit argument to type family `subst_x`.

```
app_wn  : {A:ty} wn S A ->
                      wn R (A => C) -> wn (app R S) C -> type.

subst_wn: {A:ty} wn S A ->
 ({x:tm} wne x A x -> wn  (T x) C  ) -> wn  (T S) C   -> type.

subst_x : {A:ty} wn S A -> {C:ty}
 ({x:tm} wne x A x -> wne (T x) C x) -> wn  (T S) C   -> type.

subst_y : {A:ty} wn S A ->
 ({x:tm} wne x A x -> wne (T x) C Y) -> wne (T S) C Y -> type.

%mode app_wn   +A +D    +E -F.
%mode subst_wn +A +D    +E -F.
%mode subst_x  +A +D +C +E -F.
%mode subst_y  +A +D    +E -F.
```

Proof of Lemma 2. Simultaneously by main induction on type A and side induction on the derivation $\mathcal{E}$.

1. Show $\Gamma \vdash r\,s \Downarrow C$. If the last rule of $\mathcal{E}$ was `wn_ne`, hence r is neutral, then $r\,s$ is also neutral by rule `wne_app` hence it is in the relation. If the last rule was `wn_exp`, we can apply the side ind. hyp.. The interesting case is $r = \lambda x.t$ and
$$\frac{\Gamma, x\!:\!A \vdash t \Downarrow C}{\Gamma \vdash \lambda x.t \Downarrow A \to C}\ \texttt{wn\_lam}.$$
   Here, we proceed by side ind. hyp. 2.

```
app_wn_ne   : app_wn A D (wn_ne X E) (wn_ne X (wne_app E D)).
app_wn_exp  : app_wn A D (wn_exp P E) (wn_exp (appl P) F)
               <- app_wn A D E F.
app_wn_lam  : app_wn A D (wn_lam E) (wn_exp beta F)
               <- subst_wn A D E F.
```

2. Show $\Gamma \vdash [s/x]t \Downarrow C$ for $\Gamma, x : A \vdash t \Downarrow C$. If t is not neutral, we conclude by ind. hyp. and possibly Lemma 1. Otherwise, we distinguish on the head variable of t: is it x, then we proceed by side ind. hyp. 3, otherwise by side ind. hyp. 4.

```
subst_wn_lam: subst_wn A D
 ([x][dx] wn_lam ([y][dy] E y dy x dx)) (wn_lam F)
              <- {y}{dy} subst_wn A D (E y dy) (F y dy).

subst_wn_exp: subst_wn A (D : wn S A)
 ([x][dx] wn_exp (P x) (E x dx)) (wn_exp P' E')
              <- subst_wn A D E E'
              <- subst_red S P P'.

subst_wn_x  : subst_wn A D
 ([x][dx] (wn_ne x (E x dx) : wn (T x) C)) F
              <- subst_x A D C E F.

subst_wn_y  : subst_wn A D
 ([x][dx] wn_ne Y (E x dx)) (wn_ne Y F)
              <- subst_y A D E F.
```

3. Show $\Gamma \vdash [s/x]t \Downarrow C$ for $\Gamma' \vdash t \downarrow^x C$ with $\Gamma' := \Gamma, x : A$. In case $t = x$, the type C is trivially a part of $A = C$ and we conclude by assumption $\Gamma \vdash s \Downarrow C$. Otherwise, $t = r\,u$ and the last rule in $\mathcal{E}$ was

$$\frac{\Gamma' \vdash r \downarrow^x B \to C \qquad \Gamma' \vdash u \Downarrow B}{\Gamma' \vdash r\,u \downarrow^x C}\ \texttt{wne\_app}.$$

By side ind. hyp. 3 we know that $B \to C$ is a part of A and $\Gamma \vdash r' \Downarrow B \to C$ where $r' := [s/x]r$. Similarily $\Gamma \vdash u' \Downarrow B$ for $u' := [s/x]u$ by side ind. hyp. 2. Since B is a *strict* part of A, we can apply to the main ind. hyp. 1 and obtain $\Gamma \vdash r'\,u' \Downarrow C$.

```
subst_x_x   : subst_x A D A ([x][dx]dx) D.
subst_x_app : subst_x A D C ([x][dx] wne_app
                                (E x dx)
                                (F x dx : wn (U x) B)) EF
              <- subst_x A D (B => C) E E'
              <- subst_wn A D F F'
              <- app_wn B F' E' EF.
%reduces C <= A (subst_x A D C E F).
```

The `%reduces` declaration states that the type expression C is a subexpression of A. Twelf checks that this invariant is preserved in all possibilities of introducing `subst_x A D C E F`. In case `subst_x_x` it holds because `C` is instantiated to `A`. In case `subst_x_app` it follows from the ind. hyp. which states that already `B => C` is a subexpression of `A`.

4. Show $\Gamma \vdash [s/x]t \downarrow^y C$ for $\Gamma, x\!:\!A \vdash t \downarrow^y C$. Easy, using side ind. hyp. 2 and 4. $\square$

```
subst_y_y   : subst_y A D ([x][dx] E) E.
subst_y_app : subst_y A D ([x][dx] wne_app (E x dx) (F x dx))
                                   (wne_app E' F')
               <- subst_y A D E E'
               <- subst_wn A D F F'.
```

To justify the appeals to the ind. hyp.s we invoke the Twelf termination checker with the following termination order.

```
%terminates {(Ax Ay As Aa) (Ex Ey Es Ea)}
  (subst_x  Ax _ _ Ex _)
  (subst_y  Ay _   Ey _)
  (subst_wn As _   Es _)
  (app_wn   Aa _   Ea _).
```

It expresses that the four type families are mutually recursive and terminate w. r. t. the lexicographic order on pairs $(A, \mathcal{E})$ of types A and typing derivations $\mathcal{E}$. This corresponds on a main induction on A and a side induction on $\mathcal{E}$. To verify termination, Twelf makes use of the `%reduces` declaration.

3.3 Soundness of Inductive Characterization

To complete our proof of weak normalization we need to show that for each term t in the relation $t \Downarrow A$ or $t \downarrow^x A$, there exists a normal form v such that $t \longrightarrow^* v$. After formulating full reduction $\longrightarrow$ with the usual closure properties, the proof is a simple induction on the derivation $\mathcal{E} :: t \Downarrow A$ resp. $\mathcal{E} :: t \downarrow^x A$. For lack of space we exclude the details, an implementation of the proof is available online [Abe04].

4 On Proof-Theoretical Limitations of Twelf

Having successfully completed the proof of weak normalization we are interested whether it could be extended to strong normalization and stronger object theories, like Gödel's T. In this section, we touch these questions, but our answers are speculative and preliminary.

Joachimski and Matthes [JM03] extend their proof to Gödel's T using the infinitary ω-rule state when a recursive function is weakly normalizing. Their proof is not directly transferable since only finitary rules Twelf can be represented in Twelf.

For the same reason, Tait's proof cannot be formalized in Twelf. Its key part is the definition

$$\frac{\forall s.\ \ s \Downarrow A \ \Rightarrow\ r\,s \Downarrow B}{r \Downarrow A \to B}$$

with an infinitary premise. Its literal translation into Twelf

```
wn_arr : ({S:tm} wn S A -> wn (app R S) B) -> wn R (A => B)
```

means something else, namely "if for a fresh term `S` for which we assume `wn S A` it holds that `wn (app R S) B`, then `wn R (A => B)`". Translating this back into mathematical language, we obtain the rule

$$\frac{x \Downarrow A \;\Rightarrow\; r\,x \Downarrow B}{r \Downarrow A \to B} \text{ for a fresh variable } x.$$

Since variables x are weakly normalizing anyway, we can simplify the premise further to $r\,x \Downarrow B$, obtaining clearly something we did not want in the first place.

Due to the interpretation of quantification in Twelf, infinitary rules cannot be represented, which obstructs the definition of the predicate *strongly normalizing* `sn` by the inductive rule

$$\frac{\forall t'.\; t \longrightarrow t' \Rightarrow \mathsf{sn}\, t'}{\mathsf{sn}\, t},$$

expressing that the set of strongly normalizing terms is the accessible part of the reduction relation.

Concludingly one might say that normalization of Gödel's T and proofs of strong normalization are at least difficult to express in Twelf. To see whether it is possible at all, a detailed proof-theoretic analysis of Twelf would be required.

5 Conclusion and Related Work

We have presented a formalization of Joachimski and Matthes' version of an elementary proof of weak normalization of the simply-typed λ-calculus in Twelf. We further have outlined some problems with direct proofs of strong normalization and Tait style proofs.

In the 1990s Andrej Filinski has investigated feasibility of logical relation proofs in the Edinburgh LF, but his findings remained unpublished. According to Frank Pfenning, a possible way is to first define a logic in LF, and then within this logic investigate normalization of λ-calculi. This path is taken in the Isabelle system whose framework is similar to LF but only simply-typed instead of dependently typed. On top of core Isabelle higher-order logic (HOL) is implemented which serves as the meta language in which, in turn, object theories are considered. Rich tactics for HOL make up for the loss of framework mechanism due to the extra indirection level. In Twelf, one could follow this path as well with the drawback that the built-in facilities like termination checker and automated prover [Sch00] would be rendered inapplicable.

Independently of the author, Kevin Watkins and Karl Crary have formalized a normalization algorithm and proof in Twelf, namely for Watkins' concurrent logical framework. It is said to follow the principle of our Lemma 2, namely showing that canonical forms (=normal forms) are closed under eliminations.

Acknowledgements. The author likes to thank Ralph Matthes, Frank Pfenning, Brigitte Pientka, Carsten Schürmann and Ketwin Watkins for discussions on the topic in the years 2000–2004.

References

[Abe04] Andreas Abel. A twelf proof of weak normalization for the simply-typed λ-calculus. http://www.tcs.informatik.uni-muenchen.de/˜abel/wn.tar.gz, 2004.

[Ber03] Stefan Berghofer. *Proofs, Programs and Executable Specifications in Higher-Order Logic*. PhD thesis, Technische Universität München, 2003.

[GLT89] Jean-Yves Girard, Yves Lafont, and Paul Taylor. *Proofs and Types*, volume 7 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1989.

[Gog95] Healfdene Goguen. Typed operational semantics. In M. Deziani-Ciancaglini and G. D. Plotkin, editors, *Typed Lambda Calculi and Applications (TLCA 1995), Proceedings*, volume 902 of *Lecture Notes in Computer Science*, pages 186–200. Springer, 1995.

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

[HP00] Robert Harper and Frank Pfenning. On Equivalence and Canonical Forms in the LF Type Theory. Technical report, Carnegie Mellon University, July 2000.

[JM03] Felix Joachimski and Ralph Matthes. Short proofs of normalization. *Archive of Mathematical Logic*, 42(1):59–87, 2003.

[Pfe01] Frank Pfenning. Logical frameworks. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume 2, pages 1063–1147. Elsevier Science Publishers B.V. and MIT Press, 2001.

[Pie01] Brigitte Pientka. Termination and reduction checking for higher-order logic programs. In Rajeev Goré, Alexander Leitsch, and Tobias Nipkow, editors, *Automated Reasoning, First International Joint Conference, IJCAR 2001*, volume 2083 of *Lecture Notes in Artificial Intelligence*, pages 401–415. Springer-Verlag, 2001.

[PS99] Frank Pfenning and Carsten Schürmann. System description: Twelf - a meta-logical framework for deductive systems. In H. Ganzinger, editor, *Proceedings of the 16th International Conference on Automated Deduction (CADE-16)*, volume 1632 of *Lecture Notes in Artificial Intelligence*, pages 202–206, Trento, Italy, July 1999. Springer.

[Sch00] Carsten Schürmann. *Automating the Meta-Theory of Deductive Systems*. PhD thesis, Carnegie-Mellon University, 2000.

[SP03] Carsten Schürmann and Frank Pfenning. A coverage checking algorithm for LF. In D. Basin and B. Wolff, editors, *Proceedings of the 16th International Conference on Theorem Proving in Higher Order Logics (TPHOLs 2003)*, volume 2758 of *Lecture Notes in Computer Science*, pages 120–135, Rome, Italy, September 2003. Springer.

[vRS95] Femke van Raamsdonk and Paula Severi. On normalisation. Technical Report CS-R9545, CWI, 1995.

[vRSSX99] Femke van Raamsdonk, Paula Severi, Morten Heine Sørensen, and Hongwei Xi. Perpetual reductions in lambda calculus. *Information and Computation*, 149(2):173–225, March 1999.