

A Formalization of Strong Normalization for Simply Typed Lambda Calculus and System F

Kevin Donnelly and Hongwei Xi

Boston University

Work partly funded by NSF grant CCR-0229480

What is ATS/LF?

- ATS/LF is a subsystem in the programming language ATS that supports proof verification. It is primarily designed to support a programming paradigm which we refer to as *programming with theorem-proving*.

Statics and Dynamics in ATS/LF

- There are two components in ATS/LF: a static one (statics) and a dynamic one (dynamics).
 - The statics of ATS/LF is essentially a simply typed lambda-calculus extended with some constants. We use the name *sort* to refer to a type in the statics.
 - The dynamics of ATS/LF is more or less a dependently typed language, though polymorphism is also supported. We use the name *prop* to refer to a type in the dynamics.

Some Built-in Sorts

The following built-in sorts in ATS/LF are of particular interest and importance

- *prop* : A sort for static terms that represent types of proofs.
- *int* : A sort for static integer terms. There are constants for each integer ($\dots, -1, 0, 1, \dots : int$) and for addition ($+ : (int, int) \rightarrow int$) and subtraction ($- : (int, int) \rightarrow int$).
- *bool* : A sort for static boolean conditions. There are constants for truth values (*true*, *false* : *bool*) and equality and inequality on integers ($=, < : (int, int) \rightarrow bool$).

Representing Simple Types in Statics

It is allowed to declare new sorts in the statics. For instance, the following declared datasort tp is for representing simple types in the statics of ATS/LF

```
datasort tp = TPbas
           | TPfun of (tp, tp)
```

We use $TPbas$ for some unspecified base type and $TPfun$ for the function type constructor.

Representing Lambda-Terms in Statics

The following declared datasort tm is for static terms that represent lambda-terms:

```
datasort tm = TMlam of (tm -> tm)
             | TMapp of (tm, tm)
```

This is a standard example of higher-order abstract syntax. For instance, the lambda-term $\lambda x.\lambda y.y(x)$ is represented as follows:

$$TMlam(\underline{lam} x \Rightarrow TMlam(\underline{lam} y \Rightarrow TMapp(y, x)))$$

where \underline{lam} is the lambda-binder in the statics.

Simply-Typed λ -Terms are SN

This statement can be expressed as the following type:

$$\forall t : tm. \forall T : tp. DER0(t, T) \rightarrow SN0(t)$$

where

- $DER0(t, T)$ is the type for a typing derivation that assigns the type T to the term t , and
- $SN0(t)$ is the type for a proof showing that the term t is strongly normalizing.

If this type is inhabited, then every simply-typed λ -term is strongly normalizing.

Rules for β -Reduction

$$\frac{\underline{t} \longrightarrow \underline{t}'}{\lambda x. \underline{t} \longrightarrow \lambda x. \underline{t}'} \quad (\text{REDlam})$$

$$\frac{\underline{t}_1 \longrightarrow \underline{t}'_1}{\underline{t}_1(\underline{t}_2) \longrightarrow \underline{t}'_1(\underline{t}_2)} \quad (\text{REDapp1})$$

$$\frac{\underline{t}_2 \longrightarrow \underline{t}'_2}{\underline{t}_1(\underline{t}_2) \longrightarrow \underline{t}_1(\underline{t}'_2)} \quad (\text{REDapp2})$$

$$\frac{}{(\lambda x. \underline{t}_1) \underline{t}_2 \longrightarrow \underline{t}_1[\underline{t}_2/x]} \quad (\text{REDapp3})$$

Representing β -Reduction in Dynamics

```
dataprop RED (tm, tm, int) = // integer stands for size
| {f:tm->tm, f':tm->tm, n:nat}
  REDlam (TMLam f, TMLam f', n+1) of
    {x:tm} RED (f x, f' x, n)
| {t1:tm, t2:tm, t1':tm, n:nat}
  REDapp1 (TMapp (t1, t2), TMapp (t1', t2), n+1) of
    RED (t1, t1', n)
| {t1:tm, t2:tm, t2':tm, n:nat}
  REDapp2 (TMapp (t1, t2), TMapp (t1, t2'), n+1) of
    RED (t2, t2', n)
| {f:tm->tm, t:tm}
  REDapp3 (TMapp (TMLam f, t), f t, 0)

propdef RED0 (t:tm, t':tm) = [n:nat] RED (t, t', n)
```

Representing β -Reduction in Dynamics

$$\begin{aligned} \text{REDlam} & : \quad \forall f : tm \rightarrow tm. \forall f' : tm \rightarrow tm. \forall n : nat. \\ & \quad (\forall x : tm. \text{RED}(f x, f' x, n)) \rightarrow \text{RED}(\text{TMlam } f, \text{TMlam } f', n + 1) \end{aligned}$$

$$\begin{aligned} \text{REDapp1} & : \quad \forall t_1 : tm. \forall t'_1 : tm. \forall t_2 : tm. \forall n : nat. \\ & \quad \text{RED}(t_1, t'_1, n) \rightarrow \text{RED}(\text{TMapp}(t_1, t_2), \text{TMapp}(t'_1, t_2), n + 1) \end{aligned}$$

$$\begin{aligned} \text{REDapp2} & : \quad \forall t_1 : tm. \forall t_2 : tm. \forall t'_2 : tm. \forall n : nat. \\ & \quad \text{RED}(t_2, t'_2, n) \rightarrow \text{RED}(\text{TMapp}(t_1, t_2), \text{TMapp}(t_1, t'_2), n + 1) \end{aligned}$$

$$\text{REDapp3} : \quad \forall f : tm \rightarrow tm. \forall t : tm. \text{RED}(\text{TMapp}(\text{TMlam } f, t), f t, 0)$$

$$\text{RED0}(t : tm, t' : tm) = \exists n : nat. \text{RED}(t, t', n)$$

Representing SN in dynamics

```
dataprop SN (tm, int) =  
  | {t:tm, n:nat} SN (t,n) of  
    {t':tm} (REDO(t,t') -> [n':nat | n'<n] SN(t',n'))
```

Formally, this means:

$SN : \forall t : tm. \forall n : nat.$

$(\forall t' : tm. REDO(t, t') \rightarrow \exists n' < n. SN(t', n')) \rightarrow SN(t, n)$

```
propdef SNO (t: tm) = [n:nat] SN (t, n)
```

If $SNO(t)$ is inhabited, then t is strongly normalizing.

Static Contexts and Dynamic de Bruijn Indices

```
datasort ctx = CTXnil | CTXcons of (tm, tp, ctx)
```

```
(*  
datasort tms = TMSnil | TMScons of (tm, tms)  
datasort tps = TPSnil | TPScons of (tp, tps)  
sortdef ctx = '(tms, tps)  
*)
```

```
dataprop INCTX(tm, tp, ctx, int) = // de Bruijn indices  
  | {G:ctx, t:tm, T:tp} INCTXone(t, T, CTXcons(t, T, G), 0)  
  | {G:ctx, t:tm, t':tm, T:tp, T':tp, n:nat}  
    INCTXshi(t, T, CTXcons(t', T', G), n+1) of INCTX(t, T, G, n)
```

```
propdef INCTX0(t:tm, T:tp, G:ctx) = [n:nat] INCTX(t, T, G, n)
```

What does a static context mean?

A static term Γ of the sort ctx represents a pair $(\underline{\Gamma}, \theta)$, where $\underline{\Gamma}$ is a context of the form $x_1 : \underline{T}_1, \dots, x_n : \underline{T}_n$ and θ is a substitution of the form $[x_1 \mapsto \underline{t}_1, \dots, x_n \mapsto \underline{t}_n]$. Note that there is no relation between \underline{t}_i and \underline{T}_i for $i = 1, \dots, n$.

We write $\underline{\Gamma}_\theta$ for the pair $(\underline{\Gamma}, \theta)$.

Rules for Typing Derivations

$$\frac{(x : \underline{T}) \in \underline{\Gamma} \quad \vdash \underline{T}}{\underline{\Gamma} \vdash_0 x : \underline{T}} \quad (DERvar)$$

$$\frac{\underline{\Gamma}, x : \underline{T}_1 \vdash_n \underline{t} : \underline{T}_2 \quad \vdash \underline{T}_1}{\underline{\Gamma} \vdash_{n+1} \lambda x. \underline{t} : \underline{T}_1 \rightarrow \underline{T}_2} \quad (DERlam)$$

$$\frac{\underline{\Gamma} \vdash_{n_1} \underline{t}_1 : \underline{T}_1 \rightarrow \underline{T}_2 \quad \underline{\Gamma} \vdash_{n_2} \underline{t}_2 : \underline{T}_1}{\underline{\Gamma} \vdash_{n_1+n_2+1} \underline{t}_1(\underline{t}_2) : \underline{T}_2} \quad (DERapp)$$

We write $\underline{\Gamma}_\theta \vdash_n \underline{t} : \underline{T}$ to mean that $\underline{\Gamma} \vdash_n \underline{t}_0 : \underline{T}$ for some \underline{t}_0 such that $\underline{t} = \underline{t}_0[\theta]$. Clearly, these two forms of judgments coincide when θ is empty.

Representing Typing Derivations in Dynamics

```
dataprop DER (ctx,tm,tp,int) =
  | {G:ctx, t:tm, T:tp}
    DERvar(G,t,T,0) of (INCTX0(t,T,G), TP0 T)

  | {G:ctx, f:tm->tm, T1:tp, T2:tp, n:nat}
    DERlam (G,TMlam f, TPfun(T1,T2), n+1) of
      (TP0 T1, {x:tm} DER (CTXcons (x,T1,G),f x,T2,n))

  | {G:ctx, t1:tm, t2:tm, T1:tp, T2:tp, n1:nat, n2:nat}
    DERapp (G, TMapp(t1,t2), T2, n1+n2+1) of
      (DER (G, t1, TPfun(T1,T2), n1), DER (G,t2,T1,n2))

propdef DER0 (t:tm,T:tp) = [n:nat] DER (CTXnil,t,T,n)
```

Representing Typing Derivations in Dynamics

$$\begin{aligned} \mathit{DERvar} & : \forall G : \mathit{ctx}. \forall t : \mathit{tm}. \forall T : \mathit{tp}. \\ & (\mathit{INCTX0}(t, T, G), \mathit{TP0}(T)) \rightarrow \mathit{DER}(G, t, T, 0) \end{aligned}$$

$$\begin{aligned} \mathit{DERlam} & : \forall G : \mathit{ctx}. \forall f : \mathit{tm} \rightarrow \mathit{tm}. \forall T_1 : \mathit{tp}. \forall T_2 : \mathit{tp}. \forall n : \mathit{nat}. \forall l : \mathit{nat}. \\ & (\mathit{TP0}(T_1), \forall x. \mathit{DER}(\mathit{CTXcons}(x, T_1, G), f\ x, T_2, n)) \rightarrow \\ & \mathit{DER}(G, \mathit{TMlam}\ f, \mathit{TPfun}(T_1, T_2), n + 1) \end{aligned}$$

$$\begin{aligned} \mathit{DERapp} & : \forall G : \mathit{ctx}. \forall t_1 : \mathit{tm}. \forall t_2 : \mathit{tm}. \forall T_1 : \mathit{tp}. \forall T_2 : \mathit{tp}. \forall n_1 : \mathit{nat}. \forall n_2 : \mathit{nat}. \\ & (\mathit{DER}(G, t_1, \mathit{TPfun}(T_1, T_2), n_1), \mathit{DER}(G, t_2, T_1, n_2)) \rightarrow \\ & \mathit{DER}(G, \mathit{TMapp}(t_1, t_2), T_2, n_1 + n_2 + 1) \end{aligned}$$

What does $DER(\Gamma, t, T, n)$ mean?

Suppose that Γ , t and T represent $\underline{\Gamma}_\theta$, \underline{t} and \underline{T} , respectively. Then $DER(\Gamma, t, T, n)$ means a derivation of $\underline{\Gamma}_\theta \vdash_n \underline{t} : \underline{T}$, that is, $\underline{\Gamma} \vdash_n \underline{t}_0 : \underline{T}$ for some \underline{t}_0 such that $\underline{t} = \underline{t}_0[\theta]$. Clearly, \underline{t} coincides with \underline{t}_0 when θ is empty.

Assume that t and T represent \underline{t} and \underline{T} , respectively. Then $DER0(t, T)$ means that the (closed) term \underline{t} can be assigned the type \underline{T} .

Reducibility Predicates

A lambda-term \underline{t} is reducible at a type \underline{T} , written as $\underline{R}_{\underline{T}}(\underline{t})$, if:

1. \underline{T} is a base type (that is, \underline{B} in our case) and \underline{t} is strongly normalizing, or
2. \underline{T} is $\underline{T}_1 \rightarrow \underline{T}_2$ and for all \underline{t}' , $\underline{R}_{\underline{T}_1}(\underline{t}')$ implies $\underline{R}_{\underline{T}_2}(\underline{t}(\underline{t}'))$.

It should be emphasized that $\underline{R}_{\underline{T}}(\underline{t})$ does not necessarily imply that t can be assigned the type \underline{T} .

For instance, we have $\underline{R}_{\underline{B}}(\omega)$ for $\omega = \lambda x.x(x)$ according to the definition. Also, it is clear that we cannot have $\underline{R}_{\underline{B} \rightarrow \underline{B}}(\omega)$ as it would otherwise imply $\underline{R}_{\underline{B}}(\omega(\omega))$, which is a contradiction since $\omega(\omega)$ is not normalizing.

Representing Reducibility Predicates

```
dataprop R(tm, tp) = // note the negative occurrence of R
| {t:tm} Rbas(t, TPbas) of SN0(t)
| {t:tm, T1:tp, T2:tp}
  Rfun(t, TPfun(T1, T2)) of
  {t1:tm} R(t1, T1) -> R(TMapp(t, t1), T2)
```

Rbas : $\forall t : tm. SN0\ t \rightarrow R(t, TPbas)$

Rfun : $\forall t : tm. \forall T_1 : tp. \forall T_2 : tp.$

$(\forall t_1 : tm. R(t_1, T_1) \rightarrow R(TMapp(t, t_1), T_2)) \rightarrow R(t, TPfun(T_1, T_2))$

Representing Reducibility Predicate Sequences

Given a substitution θ and a context $\underline{\Gamma}$, we say that θ is reducible at $\underline{\Gamma}$ if $\theta(x)$ is reducible at $\underline{\Gamma}(x)$ for each $x \in \mathbf{dom}(\theta) = \mathbf{dom}(\underline{\Gamma})$.

```
// sequences of reducibility predicates
dataprop RS (ctx,int) =
  | RSnil(CTXnil,0)
  | {t:tm, T:tp, G:ctx, n:nat}
    RScons(CTXcons(t,T,G), n+1) of (R(t,T), RS(G,n))

propdef RS0(G:ctx) = [n:nat] RS(G, n)
```

Main Lemma

Assume that $\underline{\Gamma}_\theta \vdash \underline{t} : \underline{T}$ is derivable and θ is reducible at $\underline{\Gamma}$. Then \underline{t} is reducible at \underline{T} . This statement is encoded as follows:

$$\forall \Gamma : \text{ctx}. \forall t : \text{tm}. \forall T : \text{tp}. \forall n : \text{nat}.$$
$$(\mathbf{DER}(\Gamma, t, T, n), \mathbf{RSO}(\Gamma)) \rightarrow \mathbf{R}(t, T)$$

CR1, CR2, CR3 and CR4

CR 1: If $\underline{R}_{\underline{T}}(\underline{t})$ then \underline{t} is strongly normalizing.

CR 2: If $\underline{R}_{\underline{T}}(\underline{t})$ and $\underline{t} \longrightarrow \underline{t}'$ then $\underline{R}_{\underline{T}}(\underline{t}')$,

CR 3: If \underline{t} is neutral, that is, \underline{t} is either a variable or an application, and for all \underline{t}' , $\underline{t} \longrightarrow \underline{t}'$ implies $\underline{R}_{\underline{T}}(\underline{t}')$, then $\underline{R}_{\underline{T}}(\underline{t})$, and

CR 4: $\underline{R}_{\underline{T}}(x)$ for any \underline{T} , which is a special case of CR 3.

Encoding a Proof of CR2

```
prfun cr2 {t:tm, t':tm, T:tp, n:nat} .<n>.
  (tp: TP (T,n), r: R(t,T), rd : RED0(t,t')): R(t',T) =
  case* r of // indicates exhaustive pattern matching
    | Rbas (sn) => Rbas (forwardSN (sn, rd))
    | Rfun{_, T1, _} (fr) =>
      let
        prval TPfun (_, tp2) = tp
      in
        Rfun(lam {t1:tm} (r:R(t1,T1)) =>
          cr2(tp2, fr r, REDapp1 rd))
      end
  end
```

If structural induction were supported, we could do:

```
prfun cr2 {t:tm, t':tm, T:tp} .<T>.
  (tp: TP0 T, r: R(t,T), rd : RED0(t,t')): R(t',T) = ...
```

Encoding a Proof of Main Lemma

```
prfun reduceLemma {G:ctx, t:tm, T:tp, n:nat} .<n>.
  (der: DER(G,t,T,n), rs: RS0 G): R (t, T) =
case* der of
| DERvar (i,_) => rGet (i, rs)
| DERlam {_,f,T1,T2,_} (_, derf) => ...
| DERapp (der1, der2) =>
  let
    prval r1 = reduceLemma(der1, rs)
    prval Rfun fr = r1
    prval r2 = reduceLemma(der2, rs)
  in
    fr r2
end
```

Representing Reducibility Candidates

$$CR1(R) \equiv \forall t : tm. R(t) \rightarrow SN0(t)$$

$$CR2(R) \equiv \forall t : tm. \forall t' : tm. (R(t), RED0(t, t')) \rightarrow R(t')$$

$$CR3(R) \equiv \forall t : tm. (NEU(t), \forall t' : tm. RED0(t, t') \rightarrow R(t')) \rightarrow R(t)$$

$$RC(R) \equiv (CR1(R), CR2(R), CR3(R))$$

Conclusion (1)

- We have presented formalizations of proofs of strong normalization for STLC and System F which use HOAS and Tait's and Girard's methods (respectively).
- The unique features of ATS/LF (in particular the separation between statics and dynamics) allow for the encoding of powerful logical relations arguments over the simple and elegant language encodings enabled by HOAS.

Conclusion (2)

- In these proofs we found that HOAS made it much easier to deal with the mundane details of naming and substitution, which often take the majority of the effort in first-order encoding.
- As a result, we are able to define the syntax and semantics of STLC and prove strong normalization as described, all in less than 300 lines of commented ATS/LF code! For System F, the proof is likewise short, under 900 lines.