Structural Recursion over Contextual Objects

Sherry Shanshan Ruan\textsuperscript{1} \hspace{1cm} Brigitte Pientka\textsuperscript{1} \hspace{1cm} Andreas Abel\textsuperscript{2}

McGill University\textsuperscript{1} \hspace{1cm} Chalmers University of Technology\textsuperscript{2}

September 2, 2014
Logical Framework

Logical framework (LF) is an elegant system for specifying formal systems via axioms and inference rules [Harper et al., 1993]

- Model variables in the object language by variables in LF
- Model binders in the object language by binders in LF

Object Language term $M, N$:

- $x$
- $\text{lam } x \cdot M$
- $M \cdot N$

Representation in LF:

```plaintext
datatype term
  type = |
  app : term → term → term |
  lam : (term → term) → term
```

Sherry Shanshan Ruan$^1$, Brigitte Pientka$^1$
Andreas Abel$^2$ (McGill)

Structural Recursion
September 2, 2014 2 / 12
Logical Framework

Logical framework (LF) is an elegant system for specifying formal systems via axioms and inference rules [Harper et al., 1993]

- Model variables in the object language by variables in LF
- Model binders in the object language by binders in LF

Object Language

\[ \text{term } M, N := x \mid \text{lam } x.M \mid M N \]

Representation in LF

```
datatype term : type =
| app : term \to term \to term
| lam : (term \to term) \to term
```
Logical Framework

Logical framework (LF) is an elegant system for specifying formal systems via axioms and inference rules [Harper et al., 1993]

- Model variables in the object language by variables in LF
- Model binders in the object language by binders in LF

Object Language

\[ \text{term } M, N ::= x \mid \text{lam } x.M \mid M N \]

Representation in LF

\[
\begin{align*}
\text{datatype } \text{term} : \text{type} = \\
\mid \text{app} : \text{term} \to \text{term} \to \text{term} \\
\mid \text{lam} : (\text{term} \to \text{term}) \to \text{term}
\end{align*}
\]

\[ \text{lam } x.\text{lam } y.x \ y \leftrightarrow \text{lam} (\lambda x.\text{lam} (\lambda y.\text{app} \ x \ y)) \]
Logical Framework

Logical framework (LF) is an elegant system for specifying formal systems via axioms and inference rules [Harper et al., 1993]

- Model variables in the object language by variables in LF
- Model binders in the object language by binders in LF

Object Language

\[ \text{term } M, N \ := \ x \mid \text{lam } x. M \mid M \ N \]

Representation in LF

\[
\begin{align*}
\text{datatype } & \quad \text{term} : \text{type} = \\
\mid & \quad \text{app} : \text{term} \rightarrow \text{term} \rightarrow \text{term} \\
\mid & \quad \text{lam} : (\text{term} \rightarrow \text{term}) \rightarrow \text{term}
\end{align*}
\]

\[ \text{lam } x. \text{lam } y. x \ y \ \leftrightarrow \ \text{lam } (\lambda x. \text{lam } (\lambda y. \ \text{I’m dependent on } x \ \text{and } y)) \]
As we recursively traverse higher-order abstract syntax trees, we extend the context of assumptions and LF object does not remain closed.
Contextual Modal Type Theory

- As we recursively traverse higher-order abstract syntax trees, we extend the context of assumptions and LF object does not remain closed.

- Contextual object $[\Gamma \vdash M]$ pairs an open term $M$ together with the context $\Gamma$ in which it is meaningful [Nanevski et al., 2008].

As we recursively traverse higher-order abstract syntax trees, we extend the context of assumptions and LF object does not remain closed.

Contextual object \([\Gamma \vdash M]\) pairs an open term \(M\) together with the context \(\Gamma\) in which it is meaningful [Nanevski et al., 2008]

Beluga is a programming environment for reasoning about formal systems in contextual LF [Cave and Pientka, 2012]
Contextual Modal Type Theory

- As we recursively traverse higher-order abstract syntax trees, we extend the context of assumptions and LF object does not remain closed

- Contextual object $[\Gamma \vdash M]$ pairs an open term $M$ together with the context $\Gamma$ in which it is meaningful [Nanevski et al., 2008]

- Beluga is a programming environment for reasoning about formal systems in contextual LF [Cave and Pientka, 2012]

- However, Beluga lacks guarantees that a given program is total
Contextual Modal Type Theory

- As we recursively traverse higher-order abstract syntax trees, we extend the context of assumptions and LF object does not remain closed.

- Contextual object $[\Gamma \vdash M]$ pairs an open term $M$ together with the context $\Gamma$ in which it is meaningful [Nanevski et al., 2008].

- Beluga is a programming environment for reasoning about formal systems in contextual LF [Cave and Pientka, 2012].

- However, Beluga lacks guarantees that a given program is total.

My work:
Develop a well-founded recursion principle and provide a consistency proof.
Example: type uniqueness

Type $T, S ::= \text{bool} \mid T \to S$

Term $M, N ::= x \mid \text{lam } x : T. M \mid M \ N$

Theorem (type uniqueness): If $\Gamma \vdash E : T$ and $\Gamma \vdash E : S$, then $eq T S (\Gamma : \text{ctx})$
Example: type uniqueness

Type \( T, S := \text{bool} \mid T \rightarrow S \)

Term \( M, N := x \mid \text{lam} \ x : T . M \mid M \ N \)

datatype \( \text{tp} : \text{type} = \)
| \( \text{bool} : \text{tp} \)
| \( \text{arr} : \text{tp} \rightarrow \text{tp} \rightarrow \text{tp} \)

datatype \( \text{tm} : \text{type} = \)
| \( \text{lam} : \text{tp} \rightarrow (\text{tm} \rightarrow \text{tm}) \rightarrow \text{tm} \)
| \( \text{app} : \text{tm} \rightarrow \text{tm} \rightarrow \text{tm} \)
Example: type uniqueness

Type $T, S := \text{bool} | T \to S$

Term $M, N := x | \text{lam } x : T . M | M N$

```plaintext
datatype \text{tp} : \text{type} =
  \mid \text{bool} : \text{tp}
  \mid \text{arr} : \text{tp} \to \text{tp} \to \text{tp} ;
datatype \text{tm} : \text{type} =
  \mid \text{lam} : \text{tp} \to (\text{tm} \to \text{tm}) \to \text{tm}
  \mid \text{app} : \text{tm} \to \text{tm} \to \text{tm} ;
```

$$
\Gamma \vdash M : T \to S \quad \Gamma \vdash N : T \quad \frac{}{\Gamma \vdash M \, N : S} \quad t_{-\text{app}}
$$

$$
\Gamma, x : T \vdash M : S \quad \frac{}{\Gamma \vdash \text{lam } x : T . M : T \to S} \quad t_{-\text{lam}}
$$
Example: type uniqueness

Type $T, S := \text{bool} \mid T \rightarrow S$

Term $M, N := x \mid \text{lam } x : T . M \mid M \ N$

datatype $\text{tp} : \text{type} =$
| bool : $\text{tp}$
| arr : $\text{tp} \rightarrow \text{tp} \rightarrow \text{tp}$;

datatype $\text{tm} : \text{type} =$
| lam : $\text{tp} \rightarrow (\text{tm} \rightarrow \text{tm}) \rightarrow \text{tm}$
| app : $\text{tm} \rightarrow \text{tm} \rightarrow \text{tm}$;

datatype $\text{oft} : \text{tm} \rightarrow \text{tp} \rightarrow \text{type} =$
| t_app : $\text{oft } M (\text{arr } T S) \rightarrow \text{oft } N T \rightarrow \text{oft } (\text{app } M N) S$
| t_lam : $(\Pi x : \text{tm}. \text{oft } x T \rightarrow \text{oft } (M x) S) \rightarrow \text{oft } (\text{lam } T M) (\text{arr } T S)$;

\[
\Gamma \vdash M : T \rightarrow S \quad \Gamma \vdash N : T \\
\frac{}{\Gamma \vdash M \ N : S} \quad t_{\text{app}}
\]

\[
\Gamma, x : T \vdash M : S \\
\frac{}{\Gamma \vdash \text{lam } x : T . M : T \rightarrow S} \quad t_{\text{lam}}
\]
Example: type uniqueness

Type $T, S := \text{bool} \mid T \rightarrow S$

Term $M, N := x \mid \lambda x : T. M \mid M \ N$

**datatype** $\text{tp} : \text{type} =$

| $\text{bool} : \text{tp}$
| $\text{arr} : \text{tp} \rightarrow \text{tp} \rightarrow \text{tp}$

**datatype** $\text{tm} : \text{type} =$

| $\lambda : \text{tp} \rightarrow (\text{tm} \rightarrow \text{tm}) \rightarrow \text{tm}$
| $\text{app} : \text{tm} \rightarrow \text{tm} \rightarrow \text{tm}$

\[
\frac{\Gamma \vdash M : T \rightarrow S \quad \Gamma \vdash N : T}{\Gamma \vdash M \ N : S} \quad t\_app
\]

\[
\frac{\Gamma, x : T \vdash M : S}{\Gamma \vdash \lambda x : T. M : T \rightarrow S} \quad t\_lam
\]

**datatype** $\text{oft} : \text{tm} \rightarrow \text{tp} \rightarrow \text{type} =$

| $t\_app : \text{oft} M (\text{arr} T S) \rightarrow \text{oft} N T \rightarrow \text{oft} (\text{app} M N) S$
| $t\_lam : (\Pi x : \text{tm}. \text{oft} x T \rightarrow \text{oft} (M x) S) \rightarrow \text{oft} (\lambda x : T M) \ (\text{arr} T S)$

Theorem (type uniqueness): If $\Gamma \vdash E : T$ and $\Gamma \vdash E : S$, then $eq T S$
Example: type uniqueness

Type $T, S := bool | T \to S$  

Term $M, N := x | lam x: T . M | M N$

datatype $tp : type =$
| bool : tp
| arr : tp $\to$ tp $\to$ tp ;

datatype $tm : type =$
| lam : tp $\to$ (tm $\to$ tm) $\to$ tm
| app : tm $\to$ tm $\to$ tm ;

datatype $oft : tm$ $\to$ tp $\to$ type $=$
| t_app : oft M (arr T S) $\to$ oft N T $\to$ oft (app M N) S
| t_lam : ($\prod$ x:tm. oft x T $\to$ oft (M x) S) $\to$ oft (lam T M) (arr T S) ;

Theorem (type uniqueness): If $\Gamma \vdash E : T$ and $\Gamma \vdash E : S$, then $eq T S$

$(\Gamma:ctx) [\Gamma \vdash oft (E..) T] \to [\Gamma \vdash oft (E..) S] \to [\vdash eq T S]$
Example: type uniqueness

```
rec unique : (Γ:ctx)[Γ ⊢ oft (E..) T]∗ → [Γ ⊢ oft (E..) S] → [ ⊢ eq T S] =
```
Example: type uniqueness

\[\text{rec unique : } (\Gamma:\text{ctx})[\Gamma \vdash \text{oft (E..) } T^*] \rightarrow [\Gamma \vdash \text{oft (E..) } S] \rightarrow [\vdash \text{eq } T \ S] = \]
\[\text{fn } d \Rightarrow \text{fn } f \Rightarrow \text{case } d \ \text{of} \]
\[\mid [\Gamma \vdash \text{t_app (D1..) } (D2..)] \Rightarrow \\
\quad \text{let } [\Gamma \vdash \text{t_app (F1..) } (F2..)] = f \ \text{in} \\
\quad \text{let } [\vdash \text{e_ref}] = \text{unique } [\Gamma \vdash D1..][\Gamma \vdash F1..] \ \text{in} \\
\quad [\vdash \text{e_ref}] \]
\[\mid [\Gamma \vdash \text{t_lam } (\lambda x.\lambda u. \ D .. x \ u)] \Rightarrow \\
\quad \text{let } [\Gamma \vdash \text{t_lam } (\lambda x.\lambda u. \ F .. x \ u)] = f \ \text{in} \\
\quad \text{let } [\vdash \text{e_ref}] = \text{unique } [\Gamma, b:\text{block } x:\text{tm}, t:\text{oft } x \vdash D .. b.1 \ b.2] \\
\quad \quad [\Gamma, b \vdash F .. b.1 \ b.2] \ \text{in} \\
\quad \quad [\vdash \text{e_ref}] \]
\[\mid [\Gamma \vdash \#q.2..] \Rightarrow \\
\quad \text{let } [\Gamma \vdash \#r.2 ..] = f \ \text{in} \\
\quad [\vdash \text{e_ref}] ; \quad \% d : \text{oft } \#q.1 \ T \quad \% f : \text{oft } \#r.1 \ S\]
Contributions

Totality = Coverage + Termination
Contributions

Totality = Coverage + Termination

Theory:
- Describe a measure for contextual objects (Context weakening and reordering)
Contributions

Totality = Coverage + Termination

Theory:
- Describe a measure for contextual objects (Context weakening and reordering)
- Design an algorithm for the generation of simultaneous pattern matching constructs and valid structural recursive calls
Contributions

Totality = Coverage + Termination

Theory:

- Describe a measure for contextual objects (Context weakening and reordering)
- Design an algorithm for the generation of simultaneous pattern matching constructs and valid structural recursive calls
- Prove weak normalization using logical relations
Contributions

Totality = Coverage + Termination

Theory:
- Describe a measure for contextual objects (Context weakening and reordering)
- Design an algorithm for the generation of simultaneous pattern matching constructs and valid structural recursive calls
- Prove weak normalization using logical relations

Implementation:
- Simple generation of recursive calls on contextual objects works (Support many proofs including type preservation, value soundness, type uniqueness)
Contributions

Totality = Coverage + Termination

Theory:
- Describe a measure for contextual objects (Context weakening and reordering)
- Design an algorithm for the generation of simultaneous pattern matching constructs and valid structural recursive calls
- Prove weak normalization using logical relations

Provides a generic proof language for first-order logic with a domain-specific recursion principle and a consistency proof
Future Work

Theory:
- Support lexicographical orderings and mutual recursion
- Generate recursive calls on recursively defined objects
- Design a recursion principle for more general recursive calls

Implementation:
- Generate more sophisticated recursive calls
Future Work

Theory:
- Support lexicographical orderings and mutual recursion
Future Work

Theory:
- Support lexicographical orderings and mutual recursion
- Generate recursive calls on recursively defined objects
Future Work

Theory:
- Support lexicographical orderings and mutual recursion
- Generate recursive calls on recursively defined objects
- Design a recursion principle for more general recursive calls
Future Work

Theory:
- Support lexicographical orderings and mutual recursion
- Generate recursive calls on recursively defined objects
- Design a recursion principle for more general recursive calls

Implementation:
- Generate more sophisticated recursive calls


Thank You