Integrating an interactive proof assistant with automatic theorem provers

Andrés Sicard-Ramírez[1]

(joint work with Ana Bove[2] **and Peter Dybjer**[2]**)**

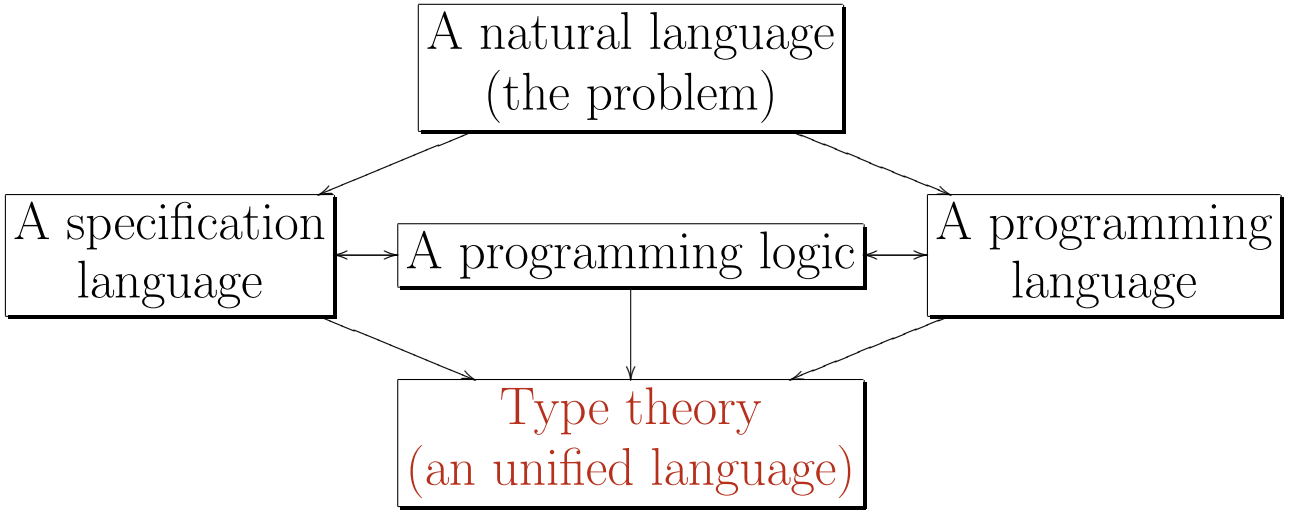
[1]Universidad EAFIT
[2]Chalmers University of Technology

Foro de investigación y docencia
Universidad EAFIT
September 29, 2010

Abstract

Interactive proof assistants are interactive systems for writing and checking proofs which can be used for to write correct programs by construction (under the Curry-Howard isomorphism). Proof assistants based on higher-order logics usually lack a good support of proof automation. We are developing a tool in which users of the Agda proof assistant obtain support from (first-order logic) automatic theorem provers (e.g. Eprover, Equinox or Metis).

Reasoning about programs: the languages



What is a type?

- A type is a set of values (and operations on them).
- Types as ranges of significance of propositional functions.[1] In modern terminology, types are domains of predicates.

  Example. $P(0) \wedge (\forall x.P(x) \rightarrow P(succ(x))) \rightarrow \forall x.P(x)$ make sense only when P is a predicate over natural numbers (i.e. $P : \mathbb{N} \rightarrow \{True, False\}$)

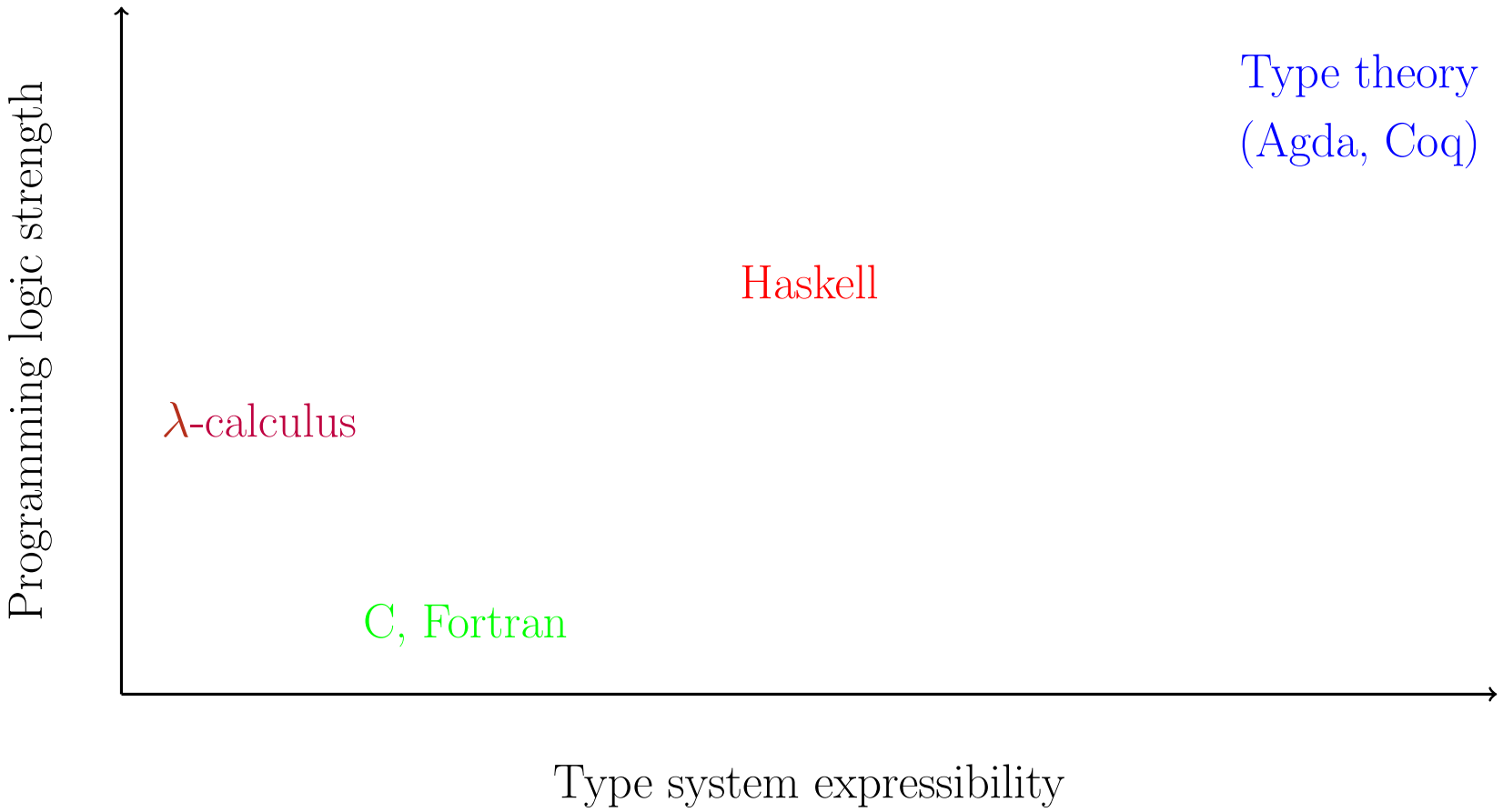
- A type system is a syntactic method for automatically checking the absence of certain erroneous behaviors by classifying program phrases according to the kinds of values they compute.[2]
- A type is an approximation of a dynamic behavior that can be derived from the form of an expression.[3]

[1] B. Russell. Mathematical logic as based on the theory of types. *American Journal of Mathematics*, 30(3):222–262, 1908.

[2] B. C. Pierce. *Types and programming languages*. The MIT Press, 2002.

[3] O. Kiselyov and C. chieh Shan. Interpreting types as abstract values. Formosan Summer School on Logic, Language and Computacion (FLOLAC '08), 2008.

Type system expressibility vs programming logic strength

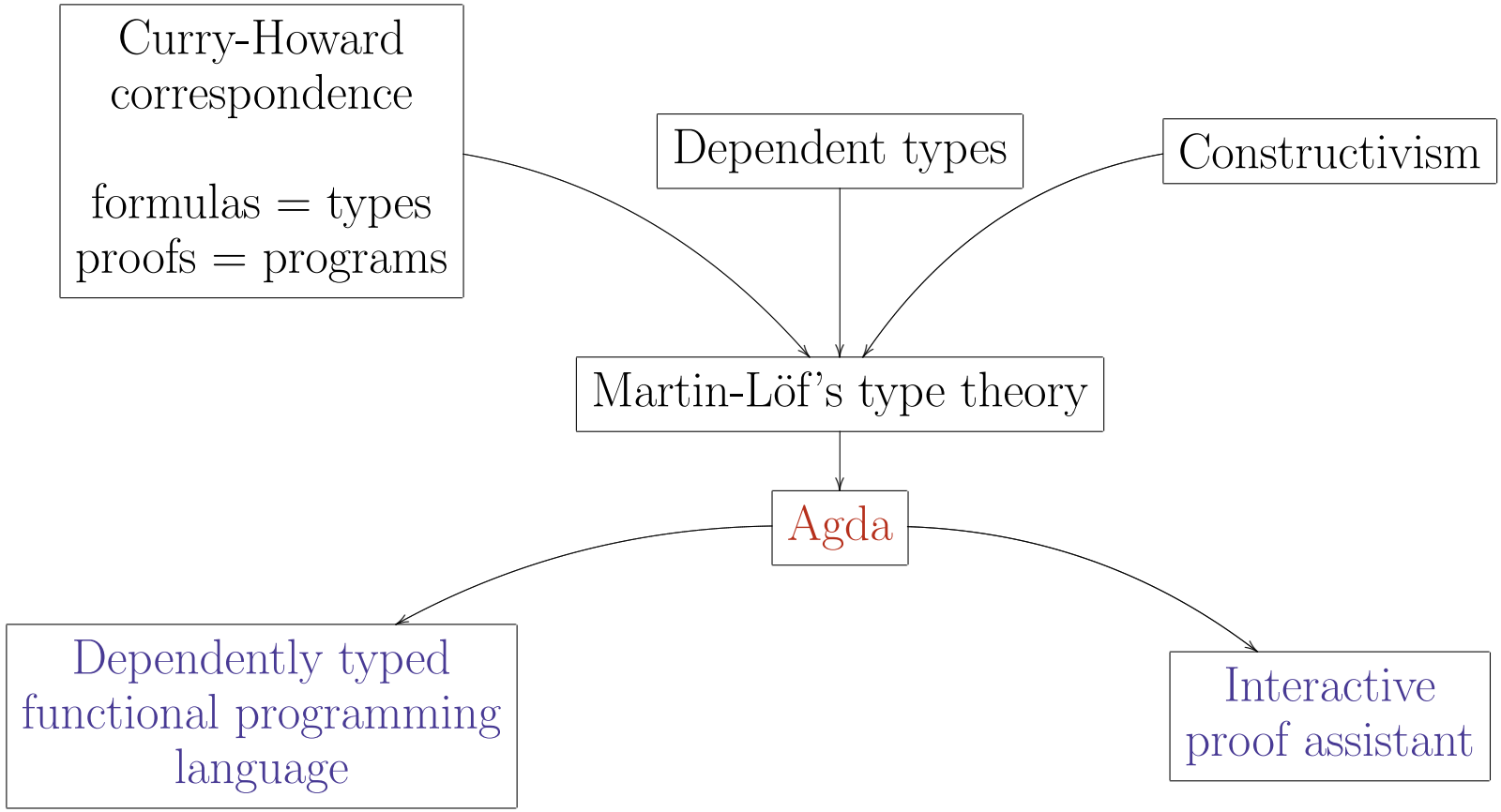


Martin-Löf's type theory: Types and terms

Per Martin-Löf. Swedish logician, philosopher, and mathematician.

Type A	Term $a : A$	
A is a set	a is an element of the set A	$A \neq \emptyset$
A is a proposition	a is a proof (construction) of the proposition A	A is true
A is a problem	a is a method of solving the problem A	A is solvable
A is a specification	a is a program than meets the specification A	A is satisfiable

The proof assistant Agda



Proof assistants and automatic theorem provers

Proof assistants (most)

- Higher order-logic
- Interactive (more user effort)
- Expressive types systems
- Complex developments
- Response from seconds to minutes

Automatic theorem provers (ATPs) (most)

- First-order logic
- Automatic
- Untyped
- One-shot problems
- Response from seconds to hours

Proof assistants and automatic theorem provers

Proof assistants (most)

- Higher order-logic
- Interactive (more user effort)
- Expressive types systems
- Complex developments
- Response from seconds to minutes

Automatic theorem provers (ATPs) (most)

- First-order logic
- Automatic
- Untyped
- One-shot problems
- Response from seconds to hours

$\Rightarrow$ Combination of automatic and interactive theorem proving

Proof assistants and automatic theorem provers

Proof assistants (most)

- Higher order-logic
- Interactive (more user effort)
- Expressive types systems
- Complex developments
- Response from seconds to minutes

Automatic theorem provers (ATPs) (most)

- First-order logic
- Automatic
- Untyped
- One-shot problems
- Response from seconds to hours

$\Rightarrow$ Combination of automatic and interactive theorem proving

Example (Agda and ATPs proofs). We will see proofs by induction, pattern matching and using equational reasoning.

Overview: the agda2atp tool

Agda users obtain support from first-order ATPs

Features:

- Agda: The high level proofs steps (introduction of hypothesis, case analysis, induction steps, etc.)
- ATPs: The "trivial" proofs steps
- The ATPs are called on users' marked conjectures
- The ATPs are called after the Agda type-checking is finished

Overview: the agda2atp tool

Agda users obtain support from first-order ATPs

Features:

- Agda: The high level proofs steps (introduction of hypothesis, case analysis, induction steps, etc.)
- ATPs: The "trivial" proofs steps
- The ATPs are called on users' marked conjectures
- The ATPs are called after the Agda type-checking is finished

What we did?

1. To modify Agda to accept the users' marked conjectures
2. To translate the required Agda internal types to FOL formulas
3. To translate the FOL formulas to ATPs' inputs

Users' marked conjectures

We added a new built-in pragma to Agda:

```
{-# ATP axiom myAxiom #-}
{-# ATP definition myDefinition #-}
{-# ATP hint myHypothesis #-}
{-# ATP prove myPostulate h1 h2 ... hn #-}
```

Users' marked conjectures

We added a new built-in pragma to Agda:

```
{-# ATP axiom myAxiom #-}
{-# ATP definition myDefinition #-}
{-# ATP hint myHypothesis #-}
{-# ATP prove myPostulate h1 h2 ... hn #-}
```

Example (Agda and ATPs proofs). We will see the previous proofs using the ATP pragma.

Implementation

Modification of the development version of Agda:

- Obvious modifications (lexer, parser, errors, etc.)
- To change the Agda internal signature

Implementation

Modification of the development version of Agda:

- Obvious modifications (lexer, parser, errors, etc.)
- To change the Agda internal signature

The external tool `agda2atp`:

- Agda has a lot features (implicit arguments, η-conversion rules, where clauses, etc.)
- Using Agda as an Haskell library (Agda has not a stable API)
- Source: Agda interface files (*.agdai)
- Target: TPTP
- ATPs supported: Eprover, Equinox, and Metis

Related work

External, internal or mix approach

Related work

External, internal or mix approach

- Andreas Abel, Thierry Coquand and Ulf Norell (2005)

  FOL plug-in to the Gandalf system for a previous and experimental version of Agda called AgdaLight (external approach)

- Makoto Takeyama (2009)

  Integration of Agda with external tools using Agda capability to generate an executable Haskell program (mix approach)

- Anton Setzer and Karim Kanso (2010)

  Combination of automated and interactive theorem proving using a built-in pragma (mix approach)

Future work

- User interaction? (non interaction with the type-checking)
- Translate a bigger part of Agda? Which one?

The translation algorithm (bonus slides)

Source: Agda internal types (simplified)

$$
\begin{aligned}
\mathsf{Types} \ni T\ U &\quad ::= S\ t \\
\mathsf{Sorts} \ni S &\quad ::= \mathsf{Set}_0 \mid \mathsf{Set}_1 \mid \dots \\
\mathsf{Terms} \ni t &\quad ::= \mathsf{Var}\ x \mid \mathsf{Lam}\ \lambda x.t \mid \mathsf{Pi}\ T\ (\lambda x.U) \mid \mathsf{Fun}\ T\ U \\
&\qquad \mid \mathsf{Def}\ d\ t^* \mid \mathsf{Con}\ c\ t^* \mid \mathsf{Sort}\ S \mid \dots
\end{aligned}
$$

The translation algorithm (bonus slides)

Source: Agda internal types (simplified)

$$
\begin{aligned}
\mathsf{Types} \ni T\ U &\quad ::= S\ t \\
\mathsf{Sorts} \ni S &\quad ::= \mathsf{Set}_0 \mid \mathsf{Set}_1 \mid \dots \\
\mathsf{Terms} \ni t &\quad ::= \mathsf{Var}\ x \mid \mathsf{Lam}\ \lambda x.t \mid \mathsf{Pi}\ T\ (\lambda x.U) \mid \mathsf{Fun}\ T\ U \\
&\qquad \mid \mathsf{Def}\ d\ t^* \mid \mathsf{Con}\ c\ t^* \mid \mathsf{Sort}\ S \mid \dots
\end{aligned}
$$

Target: First-order predicate logic with equality

$$
\begin{aligned}
\mathsf{Terms} \quad & t ::= \mathsf{FOLVar}\ x \mid \mathsf{FOLFun}\ f\ t^* \\
\mathsf{Formulas} \quad & F ::= \top \mid \bot \mid \neg F \mid F \wedge F \mid F \vee F \mid F \Rightarrow F \mid F \Leftrightarrow F \\
& \qquad \mid \forall x.F \mid \exists x.F \mid \mathsf{Predicate}\ p\ t^* \mid t \equiv t
\end{aligned}
$$

The translation algorithm (cont.)

Algorithm 0.1: typeToFormula($\Gamma :: \mathsf{Env}, T :: \mathsf{Type}$)

$$\textbf{case } T \textbf{ of } \begin{cases} (\mathsf{Set}_0, t) \rightarrow \text{termToFormula}(\Gamma, t) \\ (\mathsf{Set}_1, t) \rightarrow \text{termToFormula}(\Gamma, t) \\ \textbf{others} \rightarrow \textbf{fail} \end{cases}$$

The translation algorithm (cont.)

Algorithm 0.2: $\textsc{termToFormula}(\Gamma :: \mathsf{Env}, t :: \mathsf{Term})$

$$
\begin{array}{l}
\textbf{case } t \\
\quad \textbf{of} \begin{cases}
\mathsf{Var}\ x \rightarrow \begin{cases} \textbf{if } x \in \Gamma \\ \quad \textbf{then return } (\mathsf{Predicate}\ x\ [\,]) \\ \quad \textbf{else fail} \end{cases} \\
\mathsf{Lam}\ \lambda x.t \rightarrow \begin{cases} x' \leftarrow \textsc{freshVar}(\Gamma) \\ f \leftarrow \textsc{termToFormula}(\Gamma \cup \{x'\}, t) \\ \textbf{return } (f) \end{cases}
\end{cases}
\end{array}
$$

Algorithm 0.3: termToFormula($\Gamma :: \mathsf{Env}, t :: \mathsf{Term}$)

$$
\begin{array}{l}
\textbf{case } t \\
\quad \textbf{of} \left\{ \begin{array}{l}
\mathsf{Pi}\ T\ (\lambda x.U) \rightarrow \left\{ \begin{array}{l}
x' \leftarrow \textsc{freshVar}(\Gamma) \\
f_2 \leftarrow \textsc{typeToFormula}(\Gamma \cup \{x'\}, U) \\
\textbf{case } T \\
\quad \textbf{of} \left\{ \begin{array}{l}
(\mathsf{Set}_0, \mathsf{Def}\ d) \rightarrow \left\{ \begin{array}{l} \text{-- } x :: \mathsf{Set}_0 \\ \textbf{return } (\forall x'.f_2) \end{array} \right. \\
(\mathsf{Set}_0, \mathsf{Def}\ d\ t_1, \ldots, t_n) \rightarrow \left\{ \begin{array}{l} \text{-- The variable } x \text{ is a pr} \\ f_1 \leftarrow \textsc{typeToFormu} \\ \textbf{return } (f_1 \Rightarrow f_2) \end{array} \right. \\
(\mathsf{Set}_1, \mathsf{Sort}\ s) \rightarrow \left\{ \begin{array}{l} \text{-- } x :: \mathsf{Set}_1 \\ \textbf{return } (f_2) \end{array} \right. \\
\textbf{others} \rightarrow \textbf{fail}
\end{array} \right.
\end{array} \right.
\end{array} \right.
\end{array}
$$

Algorithm 0.4: termToFormula($\Gamma :: \mathsf{Env}, t :: \mathsf{Term}$)

$$
\begin{array}{l}
\textbf{case } t \\
\quad \textbf{of} \begin{cases}
\mathsf{Fun}\ T\ U \rightarrow \begin{cases} f_1 \leftarrow \textsc{typeToFormula}(\Gamma, T) \\ f_2 \leftarrow \textsc{typeToFormula}(\Gamma, U) \\ \textbf{return } (f_1 \Rightarrow f_2) \end{cases} \\
\mathsf{Def}\ d\ [\,] \rightarrow \begin{cases} \textbf{if } d \in \{\top, \bot\} \\ \quad \textbf{then return } (d) \\ \quad \textbf{else return } (\mathsf{Predicate}\ d\ [\,]) \end{cases} \\
\mathsf{Def}\ d\ [t] \rightarrow \begin{cases} \textbf{if } (d == \neg) \\ \quad \textbf{then } f \leftarrow \textsc{termToFormula}(\Gamma, t);\ \textbf{return } (\neg\ f) \\ \quad \textbf{else} \begin{cases} \textbf{if } d \in \{\forall D, \exists D\} \\ \quad \textbf{then} \begin{cases} f \leftarrow \textsc{termToFormula}(\Gamma, t) \\ x \leftarrow \textsc{freshVar}(\Gamma); \\ \textbf{return } ((\forall/\exists)\ x.f) \end{cases} \\ \quad \textbf{else} \begin{cases} a \leftarrow \textsc{termToFOLTerm}(\Gamma, t) \\ \textbf{return } (\mathsf{Predicate}\ d\ [a]) \end{cases} \end{cases} \end{cases}
\end{cases}
\end{array}
$$

Algorithm 0.5: termToFormula($\Gamma :: \mathsf{Env}, t :: \mathsf{Term}$)

$$
\begin{array}{l}
\textbf{case } t \\
\quad \textbf{of} \begin{cases}
\mathsf{Def}\ d\ [t_1, t_2] \rightarrow \begin{cases}
\textbf{if } d \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow\} \\
\quad \textbf{then} \begin{cases} f1 \leftarrow \text{termToFormula}(\Gamma, t_1) \\ f2 \leftarrow \text{termToFormula}(\Gamma, t_2) \\ \textbf{return } (f_1\ d\ f_2) \end{cases} \\
\quad \textbf{else} \begin{cases} a1 \leftarrow \text{termToFOLTerm}(\Gamma, t_1) \\ a2 \leftarrow \text{termToFOLTerm}(\Gamma, t_2) \\ \textbf{if } (d == \equiv) \\ \quad \textbf{then return } (a_1 \equiv a_2) \\ \quad \textbf{else return } (\mathsf{Predicate}\ d\ [a_1, a_2]) \end{cases}
\end{cases} \\
\mathsf{Def}\ d\ [t_1, \ldots, t_n] \rightarrow \begin{cases} a_i \leftarrow \text{termToFormula}(\Gamma, t_i) \\ \textbf{return } (\mathsf{Predicate}\ d\ [a_1, \ldots, a_n]) \end{cases} \\
\textbf{others} \rightarrow \textbf{fail}
\end{cases}
\end{array}
$$

The translation algorithm (cont.)

Algorithm 0.6: termToFOLTerm($\Gamma :: \mathsf{Env}, t :: \mathsf{Term}$)

$$
\begin{array}{l}
\textbf{case } t \\
\quad \textbf{of} \begin{cases}
\mathsf{Var}\ x \rightarrow \begin{cases} \textbf{if } x \in \Gamma \\ \quad \textbf{then return } (\mathsf{FOLVar}\ x) \\ \quad \textbf{else fail} \end{cases} \\
\\
\mathsf{Con}\ c\ [t_1, \ldots, t_n] \text{ or } \mathsf{Def}\ d\ [t_1, \ldots, t_n] \rightarrow \textsc{appArgs}(\Gamma, c/d, [t_1, \ldots, t_n]) \\
\textbf{others} \rightarrow \textbf{fail}
\end{cases}
\end{array}
$$

where

$\textsc{appArgs} :: \mathsf{Env} \rightarrow \mathsf{Name} \rightarrow [\mathsf{Term}] \rightarrow \mathsf{FOLTerm}$