Translating between Agda and Dedukti

Andreas Abel

Department of Computer Science and Engineering
Göteborgs Universitet

Dagstuhl Seminar 16421
Universality of Proofs
18 October 2016
Agda

- Dependent function types + (Co)inductive types + Universes
- Computation in types
- Functions defined by dependent pattern matching
- ...represented as rewrite rules
- Coverage & termination checker
Dedukti

- Logical framework + rewriting ($\lambda\Pi$ modulo)
- Dependently-typed constants + rewrite rules
- No coverage nor termination checking!
- Unlimited proof theoretical power!
- Conjecture: backend for any logic.
From Agda to Dedukti

- Agda’s definitions (function/data/record) → Dedukti’s constants
- Agda’s function clauses → Dedukti’s rewrite rules
- Agda’s universes/polymorphism → Tarski-style axiomatic universes
- Agda’s coinduction → just works in Dedukti!
Example: Hello World! in Agda

data Nat : Set where
    zero : Nat
    suc : Nat → Nat

plus : Nat → Nat → Nat
plus zero y = y
plus (suc x) y = suc (plus x y)

data Vec : Nat → Set where
    vnil : Vec zero
    vcons : ∀{n} → Nat → Vec n → Vec (suc n)

append : ∀{n m} → Vec n → Vec m → Vec (plus n m)
append vnil ys = ys
append (vcons x xs) ys = vcons x (append xs ys)
Hello World! translated to Dedukti

Nat : Type.
zero : Nat.
suc : Nat -> Nat.

def plus : Nat -> Nat -> Nat.
[ _y] plus zero _y --> _y.
[ _x, _y] plus (suc _x) _y --> suc (plus _x _y).

Vec : Nat -> Type.
vnil : Vec zero.
vcons : _n : Nat -> Nat -> Vec _n -> Vec (suc _n).

def append : _n : Nat -> _m : Nat ->
Vec _n -> Vec _m -> Vec (plus _n _m).
[ _m, _ys] append _ _m vnil _ys --> _ys.
[ _n, _m, _x, _xs, _ys] append _ _m (vcons _n _x _xs) _ys -->
vcons (plus _n _m) _x (append _n _m _xs _ys).
Example: Fibonacci Stream in Agda

record Stream : Set where  
  coinductive; constructor cons  
  field    head : Nat  
             tail : Stream  
open Stream public

plusS : (s t : Stream) → Stream  
head (plusS s t) = plus (head s) (head t)  
tail (plusS s t) = plusS (tail s) (tail t)

{-# TERMINATING #-}

fib : Stream  
( (head fib)) = 0  
(head (tail fib)) = 1  
(tail (tail fib)) = plusS fib (tail fib)
Fibonacci translated to Dedukti

Stream : Type.

def head : Stream -> Nat.
[_head, _tail] head (cons _head _tail) --> _head.

def tail : Stream -> Stream.
[_head, _tail] tail (cons _head _tail) --> _tail.

def plusS : Stream -> Stream -> Stream.
[_s, _t] head (plusS _s _t) --> plus (head _s) (head _t).
[_s, _t] tail (plusS _s _t) --> plusS (tail _s) (tail _t).

def fib : Stream.
[] head fib --> zero.
[] head (tail fib) --> suc zero.
[] tail (tail fib) --> plusS fib (tail fib).
From Dedukti to Agda

- Dedukti’s constructions are open.
- Can always add inhabitants to types.
- Can always add rewrite rules for function symbols.
- How to proceed?
Agda + rewriting

- Since 2.4, Agda has rewriting!
- Anything can be rewritten to anything.
- No confluence or termination check...
- Agda contains $\lambda\Pi$-modulo!
Example: Using Agda as LF modulo – part I

{-# OPTIONS –rewriting #-}

postulate
  _⇒_ : ∀{a}{{A : Set a}} (l r : A) → Set

{-# BUILTIN REWRITE _⇒_ #-}

postulate
  Unit Empty Bool U : Set
  unit empty bool : U

  triv : Unit
  true false : Bool
Example: Using Agda as LF modulo – part II

\( \text{El} : U \rightarrow \text{Set} \)

\( \text{r-unit} : \text{El unit} \Rightarrow \text{Unit} \)

\( \text{r-empty} : \text{El empty} \Rightarrow \text{Empty} \)

\( \text{r-bool} : \text{El bool} \Rightarrow \text{Bool} \)

\{-\# \text{REWRITE r-unit r-empty r-bool \#-}\}

\( \text{is-true} : \text{Bool} \rightarrow U \)

\( \text{r-true} : \text{is-true true} \Rightarrow \text{unit} \)

\( \text{r-false} : \text{is-true false} \Rightarrow \text{empty} \)

\{-\# \text{REWRITE r-true r-false \#-}\}
Future Work

- Quick-and-dirty prototype Agda $\rightarrow$ Dedukti (hacked in 2 days)
- TODO:
  - Universes and polymorphism
  - Proper name translation (Unicode, modules)
  - Imports and file handling
- Try also Dedukti $\rightarrow$ Agda.