On Extensions to Definitional Equality in Agda

Or: Making Agda See More

Andreas Abel

Department of Computer Science
Ludwig-Maximilians-University Munich

10th Agda Implementors' Meeting
Gothenburg, Sweden
15 September 2009

A Problem on the Agda List

- Conor `foo` type-checks, but `boo` does not.

```
data Unit : Set where
  unit : Unit

T : Unit -> Set
T unit = Nat

foo : (x : Unit) -> T x
foo unit = 0

boo : (x : Unit) -> T x
boo x = 0
```

- Agda does not see that `x = unit`.
- But this is so obvious that beginners are confused...

Eta-Expansion for the Unit Type

- Expand neutral terms of type `Unit` to `unit`.

$$\uparrow^{\texttt{Unit}} x = \texttt{unit}$$

```
boo : (x : Unit) -> T x
boo x = 0
```

- We need to check

$$\vdash x \mapsto 0 : (x : \texttt{Unit}) \to T\, x$$

- Introducing new variable x into the context

$$x : \texttt{Unit} \vdash 0 : T\,(\uparrow^{\texttt{Unit}} x)$$
$$x : \texttt{Unit} \vdash 0 : T\,\texttt{unit}$$
$$x : \texttt{Unit} \vdash 0 : \mathit{Nat}$$

Eta-Expansion for Data Types with One Constructor

- Data *types* with one constructor are non-dependent record types.

```
data Sigma (A : Set)(B : A -> Set) : Set where
  pair : (fst : A) -> (snd : B fst) -> Sigma A B
```

- Destructors `fst` and `snd` are generated:

```
fst : {A : Set}{B : A -> Set}(p : Sigma A B) -> A
snd : {A : Set}{B : A -> Set}(p : Sigma A B) -> B (fst{A}{B}p)
```

- Eta: `Sigma` is only inhabited by `pair`s.

$$\uparrow^{\mathtt{Sigma}\,A\,B} x \;=\; \mathtt{pair}\,\{A\}\,\{B\}\,(\uparrow^{A}(\mathtt{fst}\,\{A\}\,\{B\}\,x))\,(\uparrow^{B\,(\mathtt{fst}\,\{A\}\,\{B\}\,x)}(\mathtt{snd}\,\{A\}\,\{B\}\,x))$$

Eta-Expansion for the Empty Type

- Empty types have no inhabitants, so any two inhabitants of an empty type are equal.

```
data Empty : Set where
```

- Introduce an internal, inconstructible constant $*$.

$$\uparrow^{\mathtt{Empty}} x = *$$

- Does not compromise consistency, since there is already an $x : \mathtt{Empty}$. See [Abel/Coquand/Pagano, TLCA'09].

Propositional Equality

- Complication: Non-linearity.

```
data Id {A : Set}(a : A) : A -> Set where
  refl : Id a a

refl : {A : Set}(a : A) -> Id {A} a a
```

- $\mathtt{Id}\,\{T\}\,t\,t'$ is inhabited by $\mathtt{refl}\,\{T\}\,\{t\}$ if t definitionally equal to t', otherwise empty.

$$\uparrow^{\mathtt{Id}\,\{T\}\,t\,t'} = \begin{cases} \mathtt{refl}\,\{T\}\,\{t\} & \text{if } \vdash t = t' : T \\ * & \text{otherwise} \end{cases}$$

- Paper: [Abel, NBE'09].

Trust Me

- Subsumes

```
postulate
  trustMe : {A : Set}(a b : A) -> Id {A} a b
```

- Binds name `trustMe` to value

$$\uparrow^{\{A:\mathtt{Set}\}(a:A)(b:A)\to\mathtt{Id}\,\{A\}\,a\,b}\ \mathtt{trustMe}$$

- Behaves like a λ

$$(\uparrow^{(x:A)\to B\,x}\,r)\,s = \uparrow^{B\,s}(r\,s)$$

- `trustMe t t` evaluates to `refl`.

Non-Overlapping Pattern Inductive Types

- Vectors

```
data Vec (A : Set) : Nat -> Set where
  vnil  : Vec A zero
  vcons : {n : Nat}(hd : A)(tl : Vec A n) ->
          Vec A (suc n)
```

$$
\begin{aligned}
\uparrow^{\mathtt{Vec}\,A\,\mathtt{zero}} x &= \mathtt{vnil} \\
\uparrow^{\mathtt{Vec}\,A\,(\mathtt{suc}\,n)} x &= \mathtt{vcons}\,(\uparrow^{A}(\mathtt{hd}\,x)) \\
&\qquad (\uparrow^{\mathtt{Vec}\,A\,n}(\mathtt{tl}\,x))
\end{aligned}
$$

- Need to termination check data declaration!

```
data REmpty : Set where
  bla : REmpty -> REmpty
```

Smart Case Example: The Filinski Test

```
tripleF : (f : Bool -> Bool)(x : Bool) ->
  f (f (f x)) == f x
tripleF f true with f true
... | true = ?
... | false with f false
  ... | true = ?
  ... | false = ?
tripleF f false with f false
... | false = ?
... | true with f true
  ... | true = ?
  ... | false = ?
```

Sorted Lists

Another problem from the Agda list:

```
data SList : Nat -> Set where
  snil   : SList zero
  scons  : {n : Nat}(m : Nat) -> True (n <= m) ->
           SList n -> SList m

max : Nat -> Nat -> Nat
max n m with n <= m
... | true  = m
... | false = n
```

Sorted Lists, Insertion

- Naive try:

```
insert : {n : Nat}(m : Nat) -> SList n ->
         SList (max n m)
insert m snil = scons m tt snil
insert m (scons n p l) with n <= m
insert m (scons n p l) | true  =
   scons m ? (scons n p l)
insert m (scons n p l) | false =
   scons n (maxLemma p (notLe ?)) (insert m l)
```

- Intuitively, the holes have trivial proofs.
- In practice, Agda does not know `n <= m = true`.

Smart Case in MiniAgda and PiSigma-1.0

- When checking `e` in `case v of p -> e`, add rewrite $v \longrightarrow p$ to evaluator.
- Make sure v is in normal form, also w.r.t. to rewrites.
- Detect inconsistencies, like $v \longrightarrow \mathsf{true}$, $v \longrightarrow \mathsf{false}$.
- When adding new rewrite, apply it to old rewrites.
- Implemented prototypically in MiniAgda and old PiSigma (Thorsten Altenkirch, Nicolas Oury).
- Termination? Completeness?

Discussion

- Time is ripe for eta!
- Smart case should be thoroughly researched.
- Prototypical Agda implementation!?
- More rewriting?