

Chapter 17: Basics of Agda

Zhenjiang Hu, Wei Zhang

School of Computer Science, PKU

November 8, 2023

What is Agda?

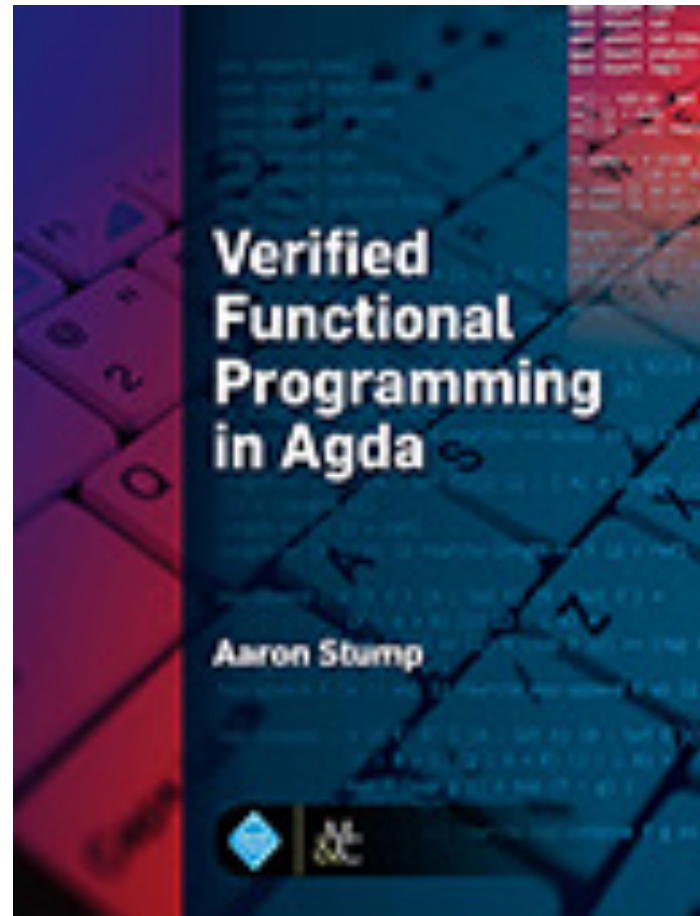
- A **dependently typed** programming language
 - Implemented in Haskell
- A **proof assistant**: propositions-as-types
 - Types: propositions
 - Terms (functional programs): proofs



Installation

- There are several ways to install Agda:
 - Using a released **source package** from Hackage
 - Using a **binary package** prepared for your platform
 - Using the development version from the **Git repository**
- More information is available at
<https://agda.readthedocs.io/en/v2.6.2.2/getting-started/installation.html>

Reference



<https://dl.acm.org/doi/book/10.1145/2841316>

Agda source: <http://svn.divms.uiowa.edu/repos/clc/projects/agda/ial-releases/1.2>

17.1 Functional Programming with the Booleans

Declaring the Datatype of Booleans

bool.agda

```
module bool where

-----

-- datatypes
-----

data B : Set where
  tt : B
  ff : B
```

基本命令

Load:	C-c C-l
Compile:	C-x C-c
Compute:	C-c C-n

Defining Boolean Operations

```
-- not
~_ :  $\mathbb{B} \rightarrow \mathbb{B}$ 
~ tt = ff
~ ff = tt

-- and
_&&_ :  $\mathbb{B} \rightarrow \mathbb{B} \rightarrow \mathbb{B}$ 
tt && b = b
ff && b = ff

-- or
_|_|_ :  $\mathbb{B} \rightarrow \mathbb{B} \rightarrow \mathbb{B}$ 
tt || b = tt
ff || b = b
```

注：类型不可省略。

Defining Boolean Operations

```
if_then_else_ :  $\forall \{l\} \{A : \text{Set } l\}$   
                $\rightarrow \mathbb{B} \rightarrow A \rightarrow A \rightarrow A$   
if tt then y else z = y  
if ff then y else z = z  
  
_xor_ :  $\mathbb{B} \rightarrow \mathbb{B} \rightarrow \mathbb{B}$   
tt xor ff = tt  
ff xor tt = tt  
tt xor tt = ff  
ff xor ff = ff  
  
_nor_ :  $\mathbb{B} \rightarrow \mathbb{B} \rightarrow \mathbb{B}$   
x nor y =  $\sim (x \mid\mid y)$ 
```


17.2 Constructive Proof

Curry-Howard Isomorphism

- Formulas (Properties) as Types

```
~ ~ tt ≡ tt
```

- Proof as Programs

```
~~tt : ~ ~ tt ≡ tt  
~~tt = refl  
  
~~ff : ~ ~ ff ≡ ff  
~~ff = refl
```

definitionally equal.

```
data _≡_ {ℓ} {A : Set ℓ} (x : A) : A → Set ℓ where  
  refl : x ≡ x
```

Proving Theorems using Pattern Matching

```
~~-elim : ∀ (b : ℬ) → ~ ~ b ≡ b  
~~-elim tt = ~ ~ tt  
~~-elim ff = ~ ~ ff
```

```
&&-idem : ∀ (b : ℬ) → b && b ≡ b  
&&-idem tt = refl  
&&-idem ff = refl
```

Implicit Arguments

```
&&-idem : ∀ (b : ℬ) → b && b ≡ b  
&&-idem tt = refl  
&&-idem ff = refl
```



自动推导 \mathbb{B}

```
&&-idem : ∀ (b) → b && b ≡ b  
&&-idem tt = refl  
&&-idem ff = refl
```



implicit argument b

```
&&-idem : ∀ {b} → b && b ≡ b  
&&-idem{tt} = refl  
&&-idem{ff} = refl
```

Implicit Arguments

```
&&-idem-tt : tt && tt ≡ tt  
&&-idem-tt = &&-idem
```

=

```
&&-idem-tt : tt && tt ≡ tt  
&&-idem-tt = &&-idem{tt}
```

Theorems with Hypotheses

```
||≡ff2 : ∀ {b1 b2} → b1 || b2 ≡ ff → b2 ≡ ff
||≡ff2 {tt} ()
||≡ff2 {ff}{tt} ()
||≡ff2 {ff}{ff} p = refl
```

absurd pattern (): the case being considered is impossible.

In-Class Exercise: Prove the following.

```
||≡ff1 : ∀ {b1 b2} → b1 || b2 ≡ ff → ff ≡ b1
```

Matching on Equality Proofs

```
||-cong1 : ∀ {b1 b1' b2} →  
             b1 ≡ b1' → b1 || b2 ≡ b1' || b2  
||-cong1 refl = refl
```

引入等式

定义性相等

=

```
||-cong1 : ∀ {b1 b1' b2} →  
             b1 ≡ b1' → b1 || b2 ≡ b1' || b2  
||-cong1 {b1} {.b1'} {b2} refl = refl
```

The rewrite Directive

```
||-cong₂ : ∀ {b1 b2 b2'} →  
           b2 ≡ b2' → b1 || b2 ≡ b1 || b2'  
||-cong₂ p rewrite p = refl
```

rewrite p: allow a more complicated equation proved by p

Other Examples

Polymorphic theorem

```
ite-same :  $\forall \{l\} \{A : \text{Set } l\} \rightarrow$   
            $\forall (b : \mathbb{B}) (x : A) \rightarrow$   
            $(\text{if } b \text{ then } x \text{ else } x) \equiv x$   
ite-same tt x = refl  
ite-same ff x = refl
```

```
 $\mathbb{B}$ -contra :  $\text{ff} \equiv \text{tt} \rightarrow \forall \{l\} \{P : \text{Set } l\} \rightarrow P$   
 $\mathbb{B}$ -contra ()
```

Homework

17.1. Define a datatype `day`, which is similar to the `B` datatype but has one constructor for each day of the week.

17.2. Using the `day` datatype from the previous problem, define a function `nextday` of type `day → day`, which given a day of the week will return the next day of the week.

17.3. Give a proof of the following formula:

```
ite-arg : ∀{ℓ ℓ'}{A : Set ℓ}{B : Set ℓ'} →  
          (f : A → B)(b : ℬ)(x y : A) →  
          (f (if b then x else y)) ≡ (if b then f x else f y)
```