Chapter 17: Basics of Agda

Zhenjiang Hu, Wei Zhang
School of Computer Science, PKU

November 12, 2025

What is Agda?

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

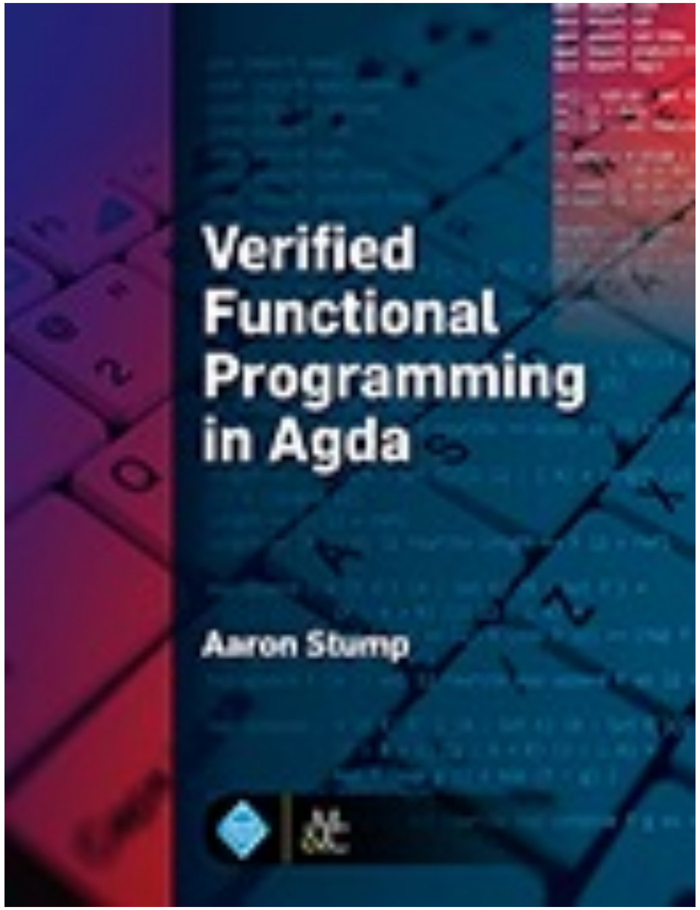
Installation

- 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
Type Check:	C-c C-d
Compute:	C-c C-n

Defining Boolean Operations

```
-- not
~_ : 𝔹 → 𝔹
~ tt = ff
~ ff = tt


-- and
_&&_ : 𝔹 → 𝔹 → 𝔹
tt && b = b
ff && b = ff

-- or
_||_ : 𝔹 → 𝔹 → 𝔹
tt || b = tt
ff || b = b
```

注：类型不可省略。

北京大學 PEKING UNIVERSITY

Defining Boolean Operations

```
if_then_else_ : ∀ {ℓ} {A : Set ℓ}
                   → 𝔹 → A → A → A
if tt then y else z = y
if ff then y else z = z

_xor_ : 𝔹 → 𝔹 → 𝔹
tt xor ff = tt
ff xor tt = tt
tt xor tt = ff
ff xor ff = ff

_nor_ : 𝔹 → 𝔹 → 𝔹
x nor y = ~ (x || 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 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
```

dot pattern

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 : ∀{ℓ}{A : Set ℓ} →
           ∀(b : 𝔹) (x : A) →
           (if b then x else x) ≡ x
ite-same tt x = refl
ite-same ff x = refl
```

```
𝔹-contra : ff ≡ tt → ∀{ℓ} {P : Set ℓ} → P
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)
```