Chapter 17: Basics of Agda

Zhenjiang Hu, Wei Zhang

School of Computer Science, PKU

November 11, 2026

北京大學 PEKING UNIVERSITY

What is Agda?

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



北京大學
PEKING UNIVERSITY

Installation

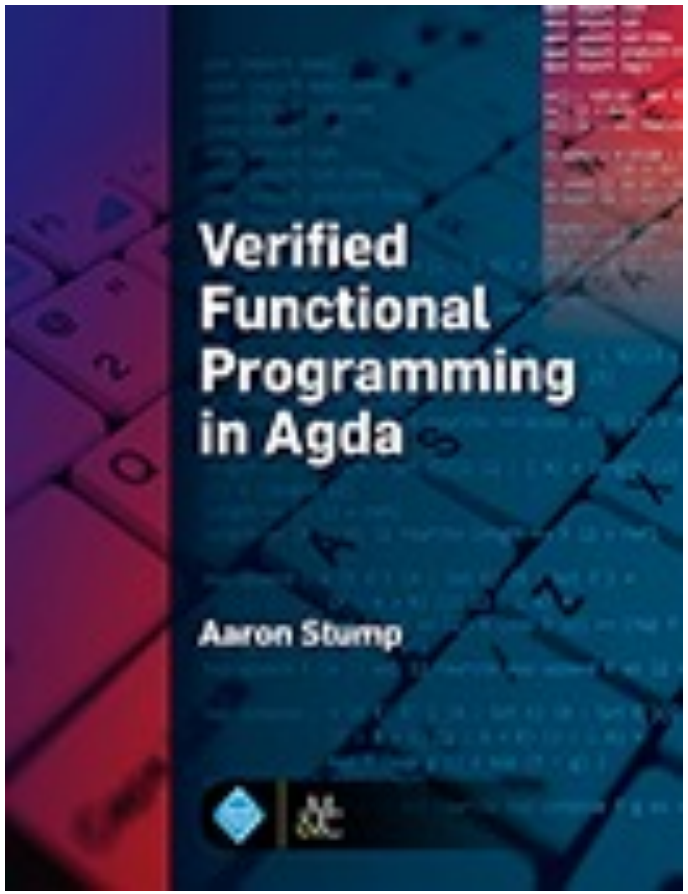
- There are several ways to install Agda:
  - Using a released source package from Hackage
  - Using a binary package prepared for your platform
  - Using the development version from the Git repository

- More information is available at

https://agda.readthedocs.io/en/latest/getting-started/installation.html

北京大學 PEKING UNIVERSITY

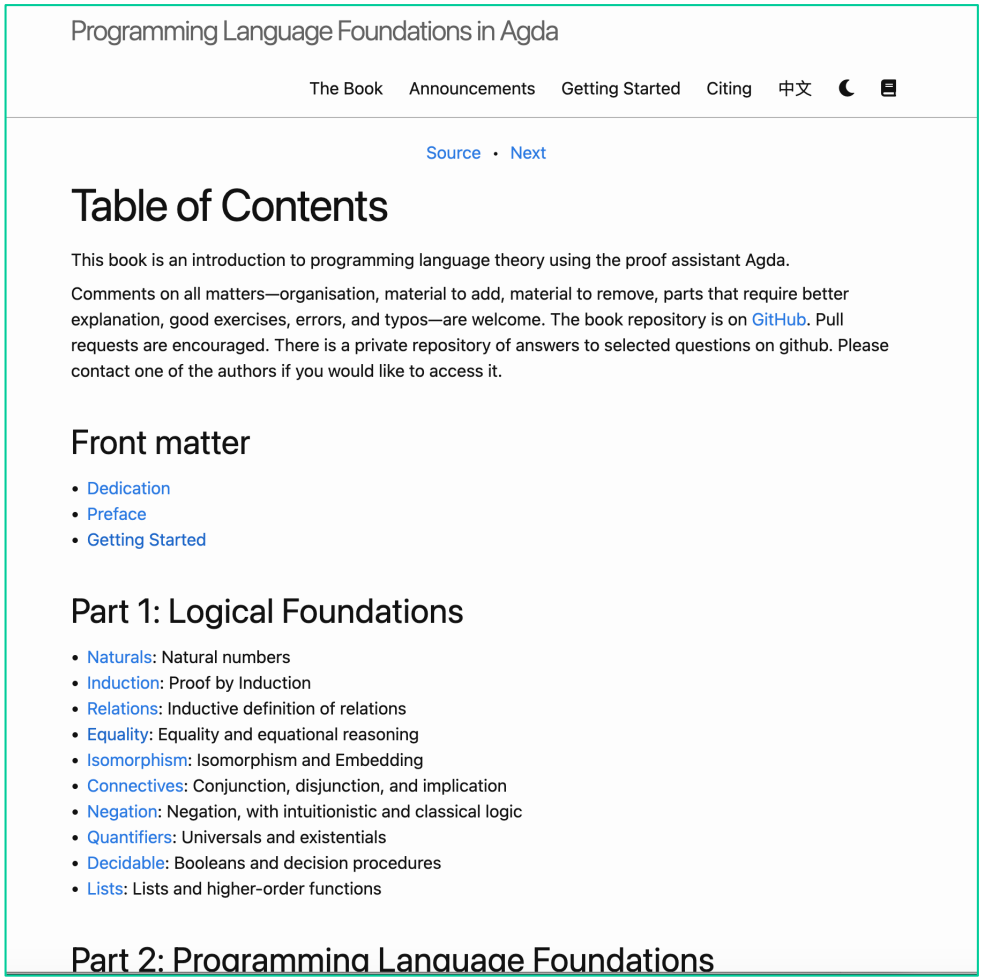
Textbook



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

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

Reference



Programming Language Foundations in Agda

The Book Announcements Getting Started Citing 中文

Source • Next

Table of Contents

This book is an introduction to programming language theory using the proof assistant Agda.

Comments on all matters—organisation, material to add, material to remove, parts that require better explanation, good exercises, errors, and typos—are welcome. The book repository is on GitHub. Pull requests are encouraged. There is a private repository of answers to selected questions on github. Please contact one of the authors if you would like to access it.

Front matter

- Dedication
- Preface
- Getting Started

Part 1: Logical Foundations

- Naturals: Natural numbers
- Induction: Proof by Induction
- Relations: Inductive definition of relations
- Equality: Equality and equational reasoning
- Isomorphism: Isomorphism and Embedding
- Connectives: Conjunction, disjunction, and implication
- Negation: Negation, with intuitionistic and classical logic
- Quantifiers: Universals and existentials
- Decidable: Booleans and decision procedures
- Lists: Lists and higher-order functions

Part 2: Programming Language Foundations

https://plfa.github.io/

PEKING UNIVERSITY 北京大學

17.1 Functional Programming with the Booleans

北京大學
PEKING UNIVERSITY

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

北京大學 PEKING UNIVERSITY

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)
```


北京大學 PEKING UNIVERSITY

17.2 Constructive Proof

北京大學
PEKING UNIVERSITY

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
```


北京大學 PEKING UNIVERSITY

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
```


北京大學 PEKING UNIVERSITY

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
```


北京大學 PEKING UNIVERSITY

Implicit Arguments

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

=

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


北京大學 PEKING UNIVERSITY

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
```


北京大學 PEKING UNIVERSITY

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

北京大學 PEKING UNIVERSITY

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

北京大學 PEKING UNIVERSITY

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 ()
```


北京大學 PEKING UNIVERSITY

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)
```


北京大學 PEKING UNIVERSITY