Let's Write a Type Checker

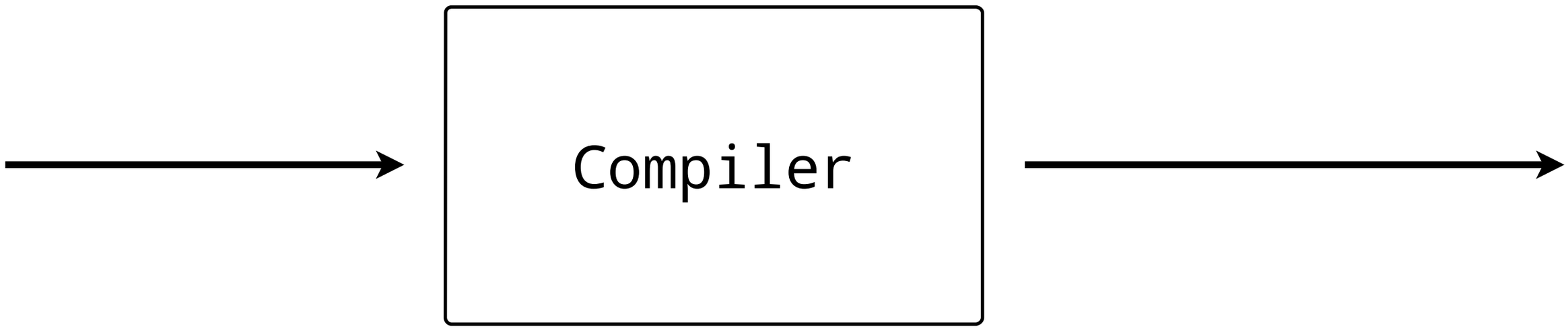
Ionuț G. Stan — I T.A.K.E. — May 2015

The Plan

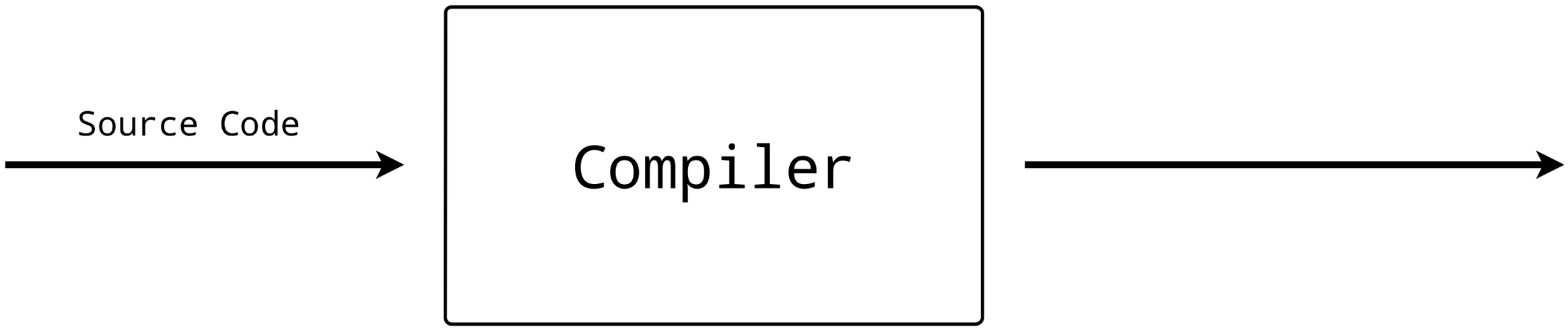
- Part 1
  - Compilers Overview
  - Type Checking vs Type Inference
  - Vehicle Language
  - Wand's Type Inference Algorithm
- Part 2
  - Live Demonstration

Compilers Overview

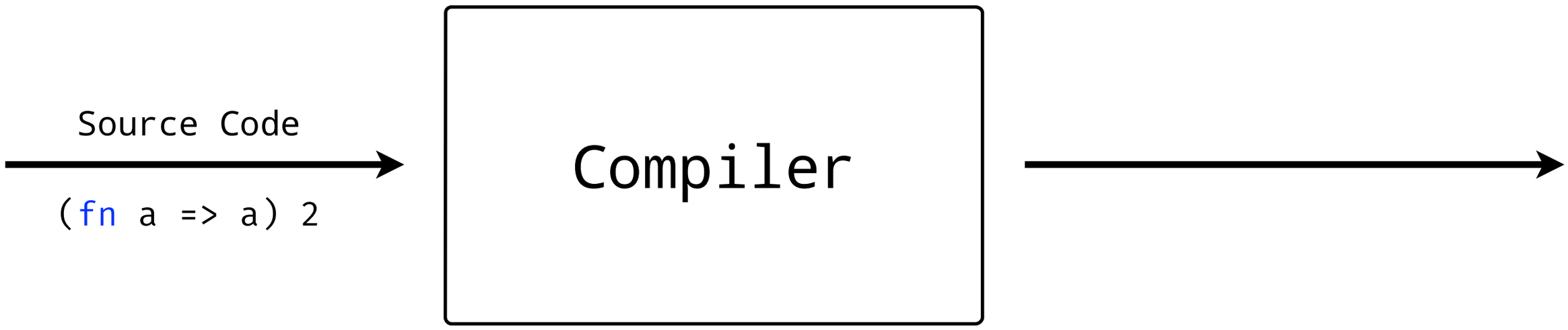
Compilers Overview



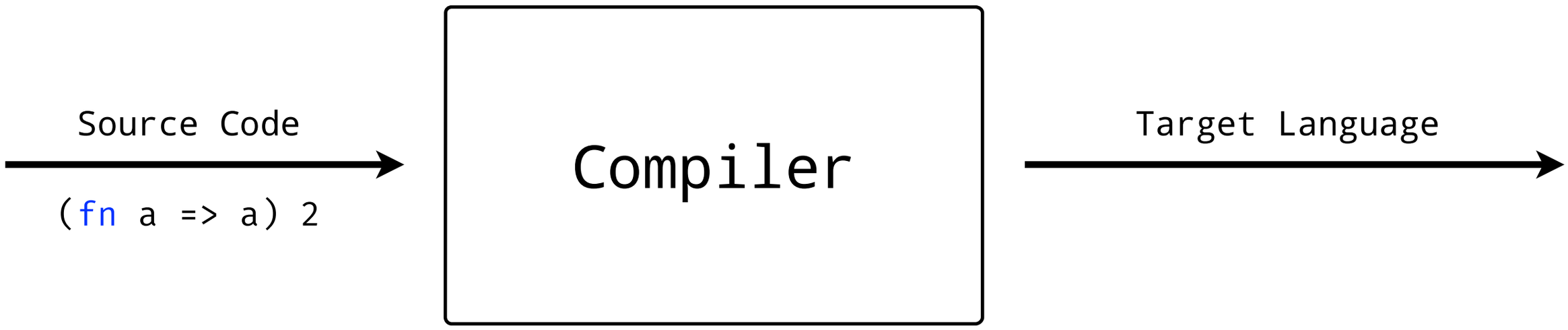
Compilers Overview



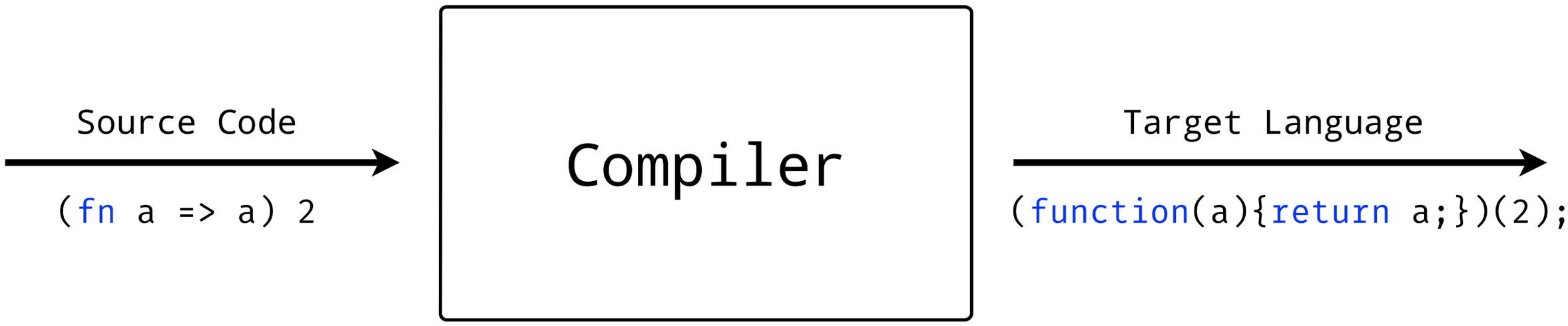
Compilers Overview



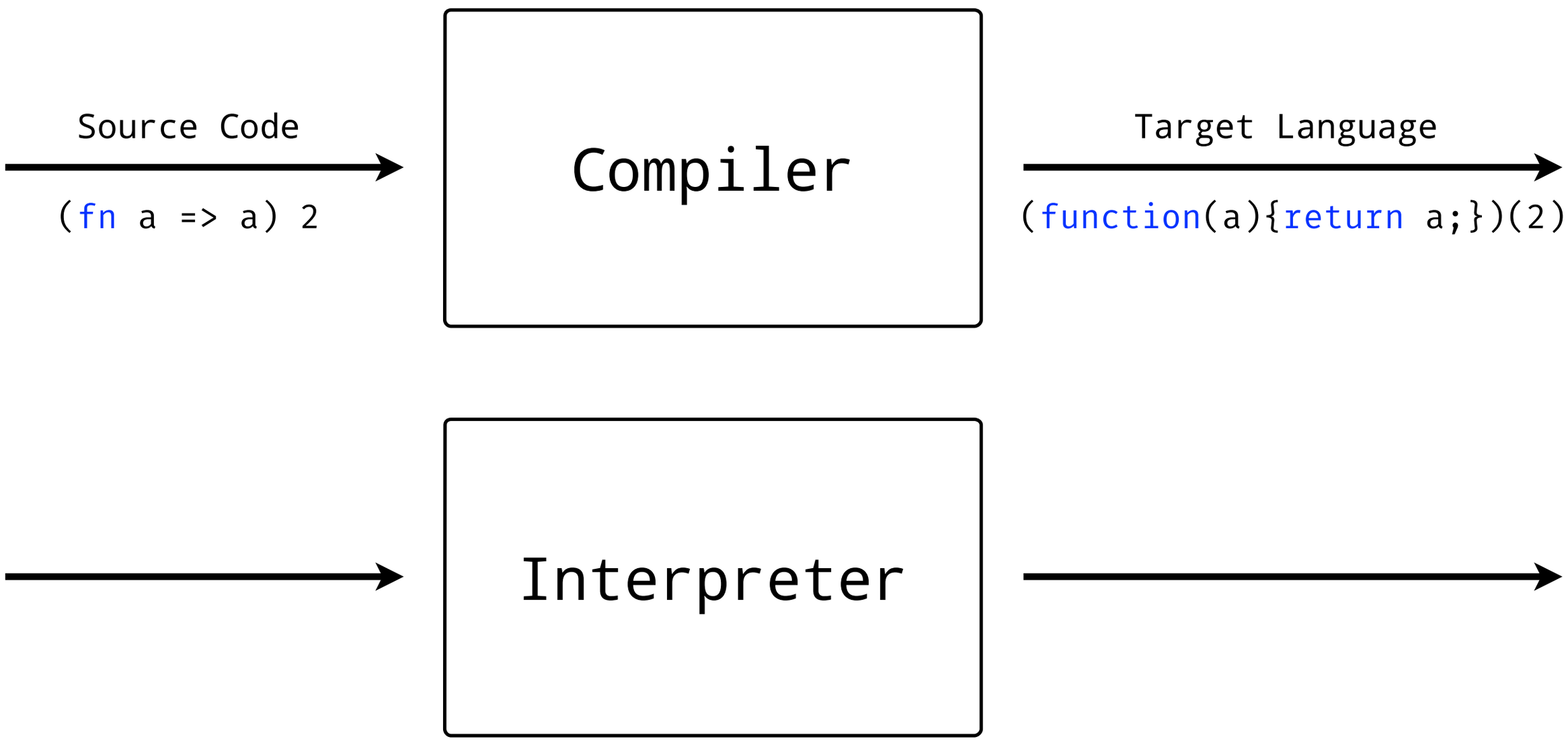
Compilers Overview



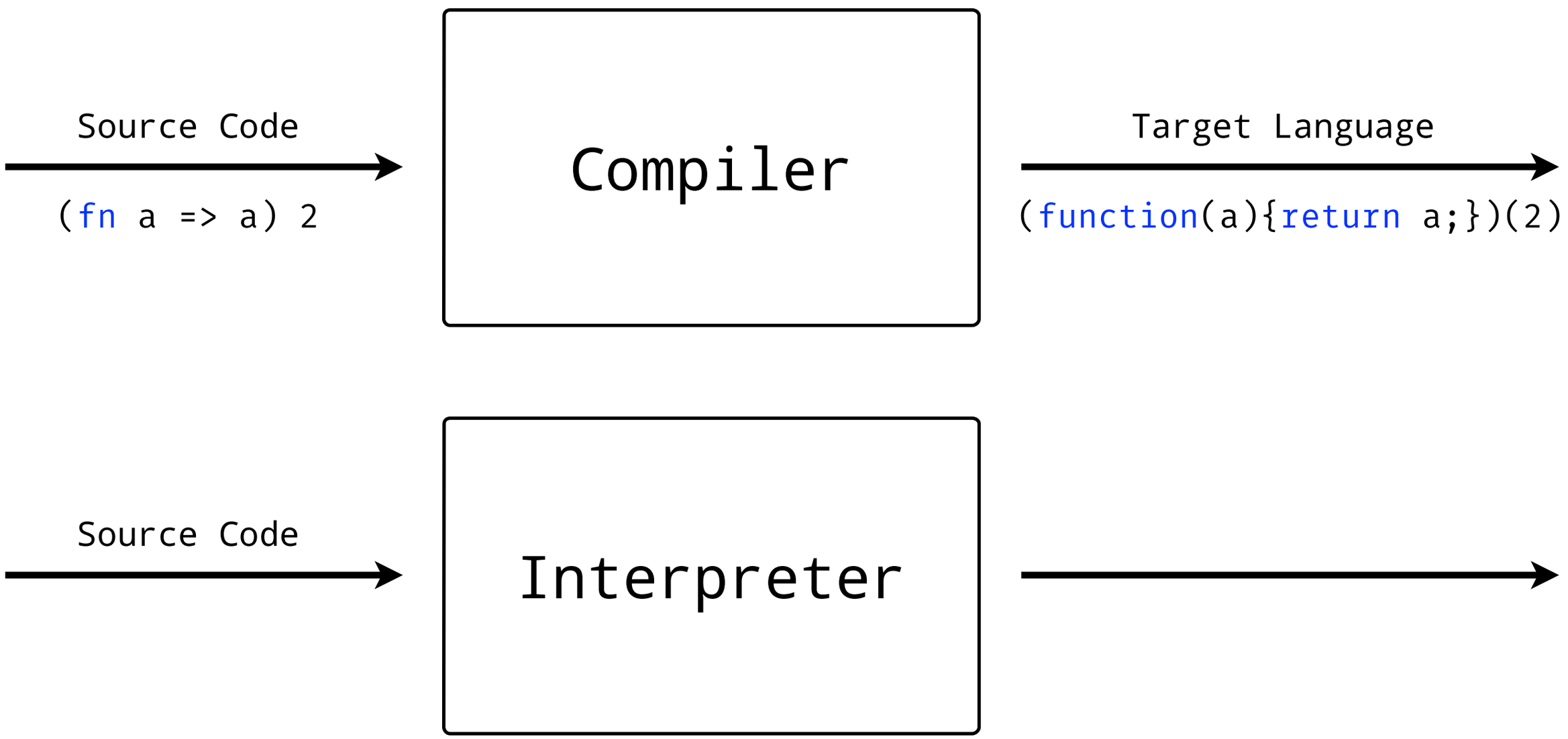
Compilers Overview



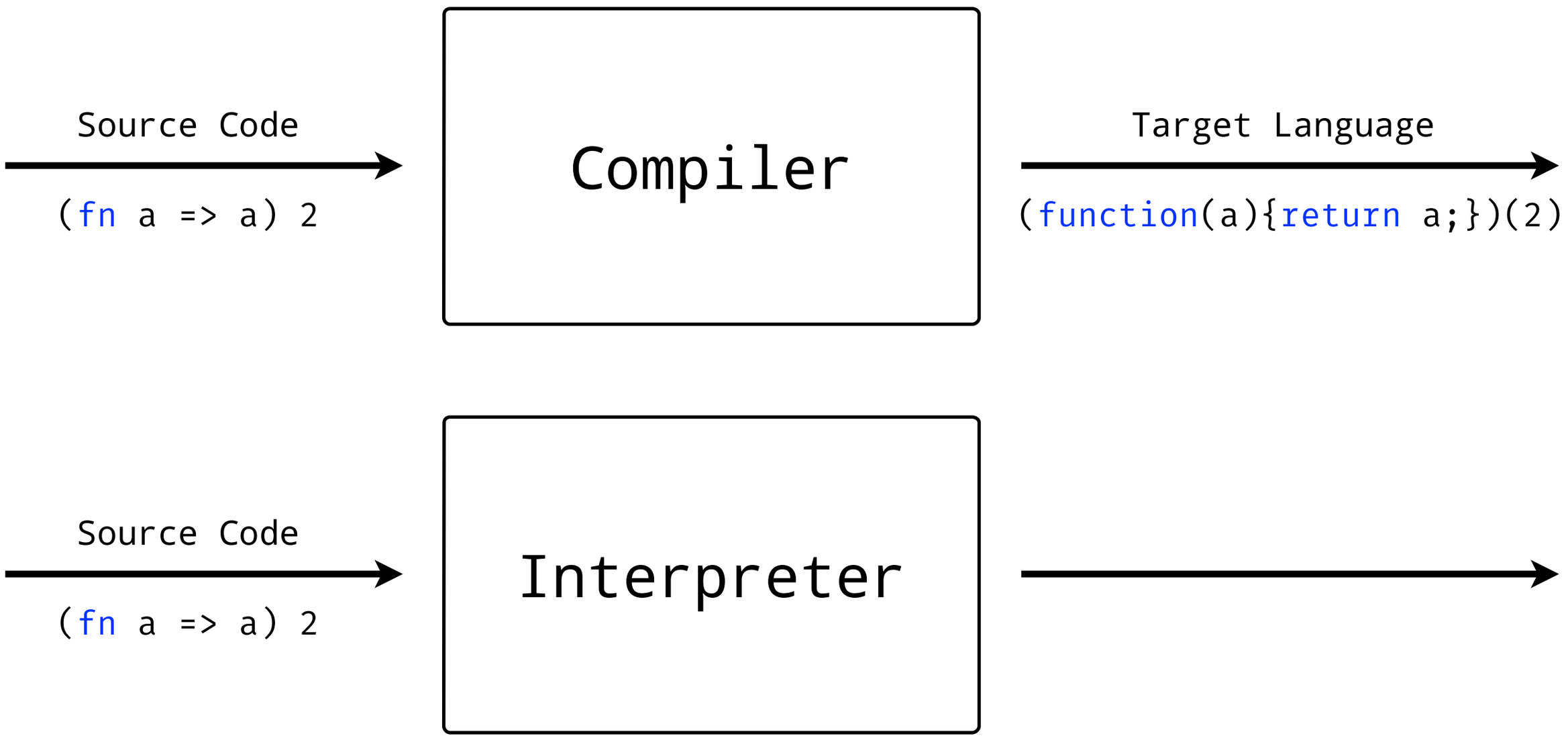
Compilers Overview



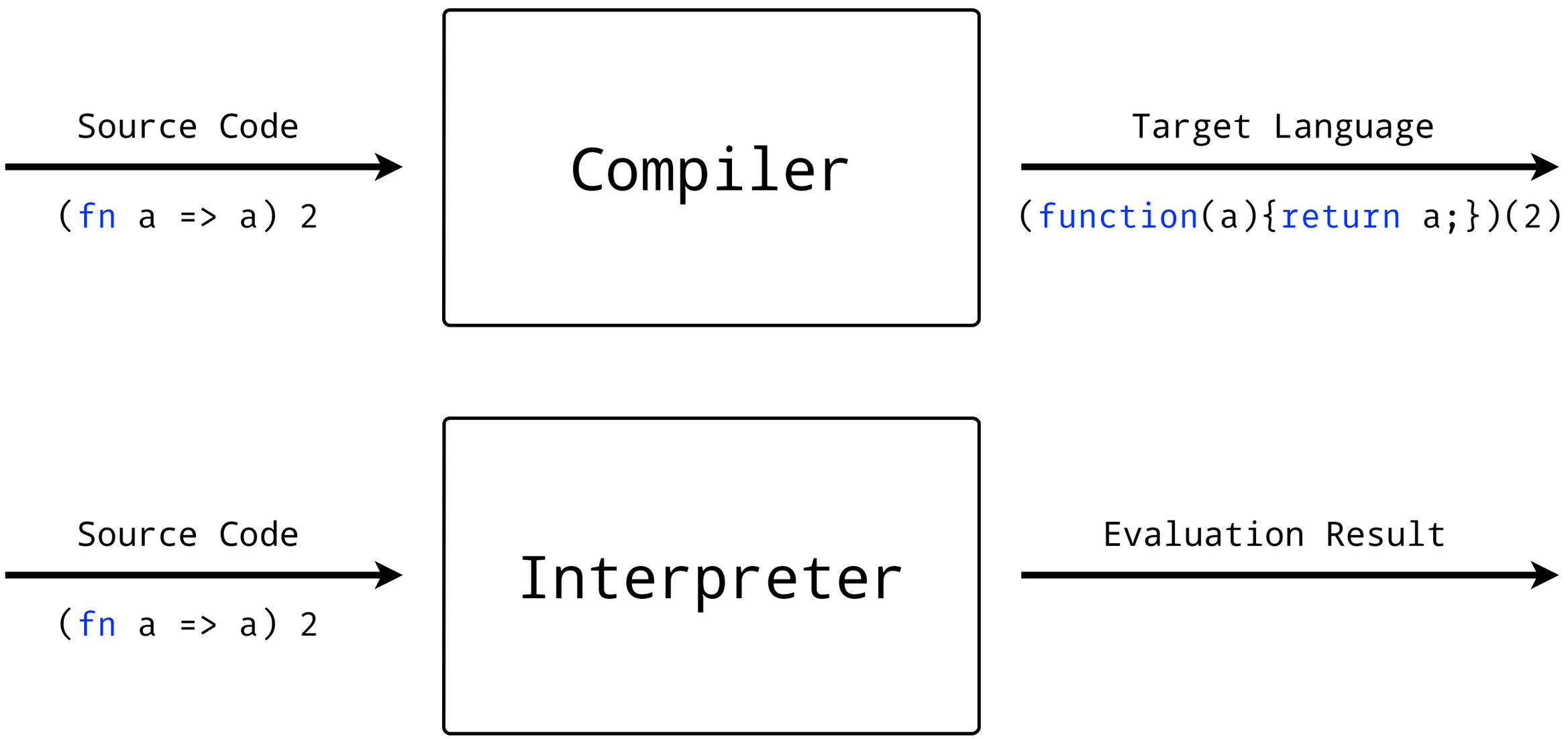
Compilers Overview



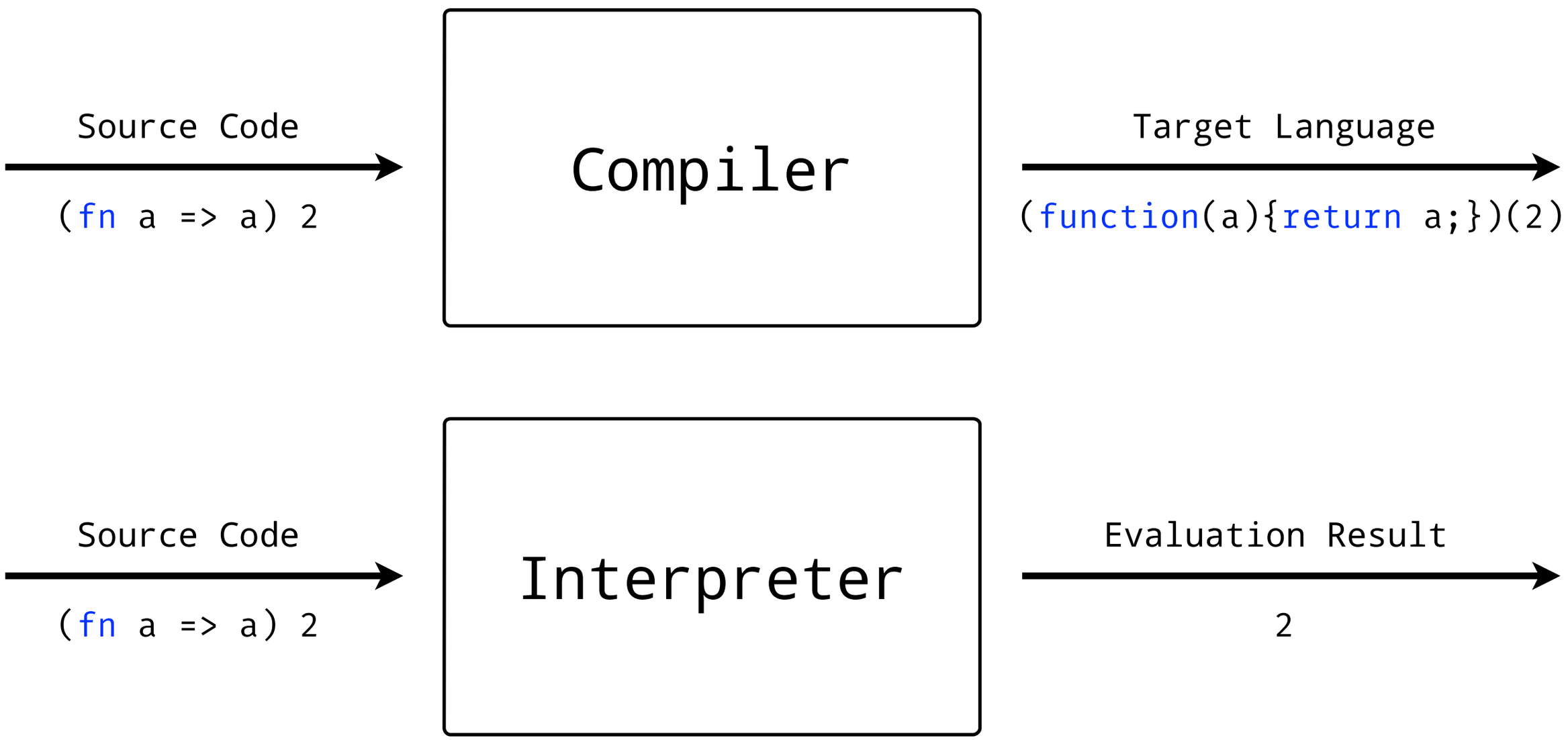
Compilers Overview



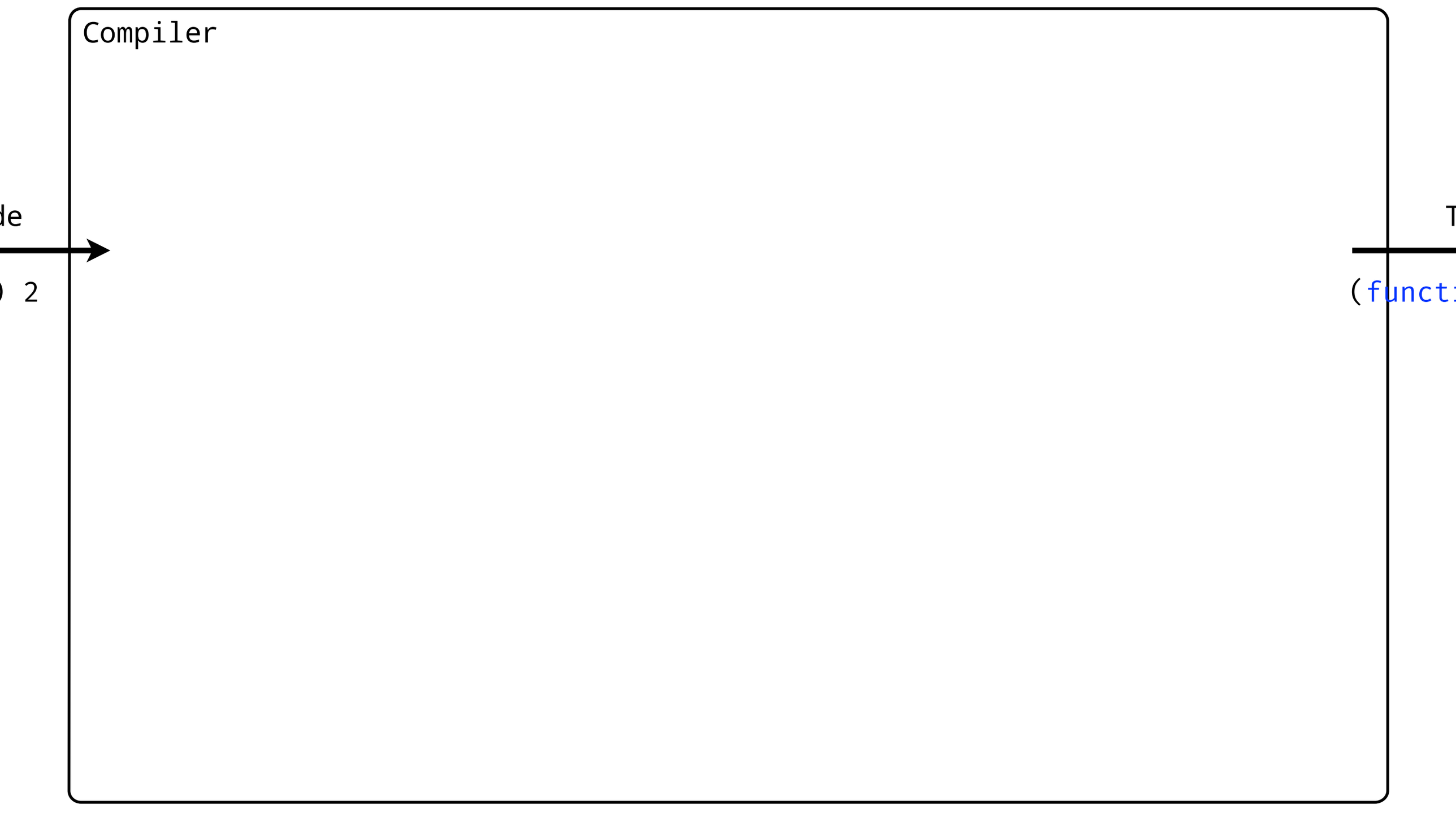
Compilers Overview



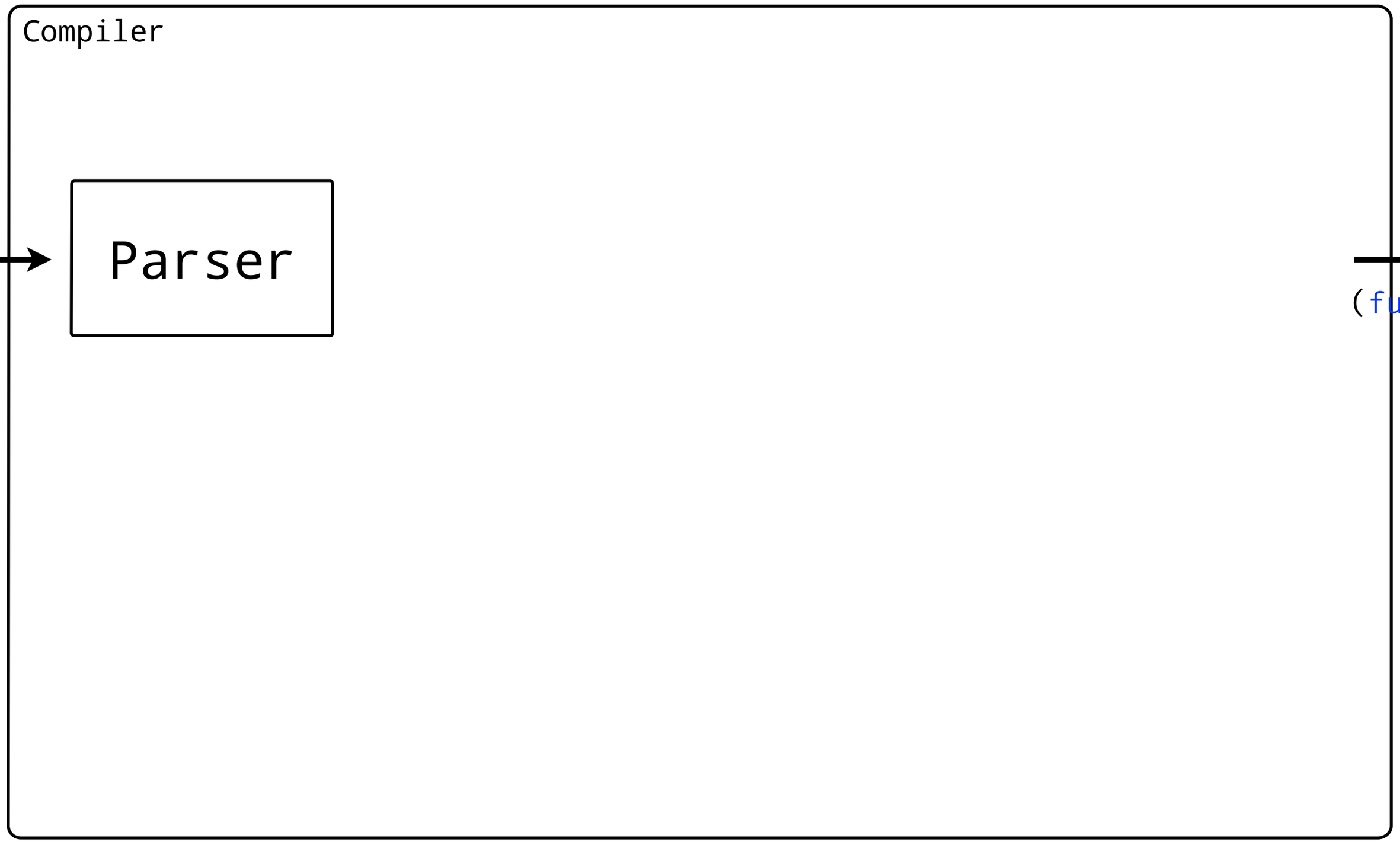
Compilers Overview



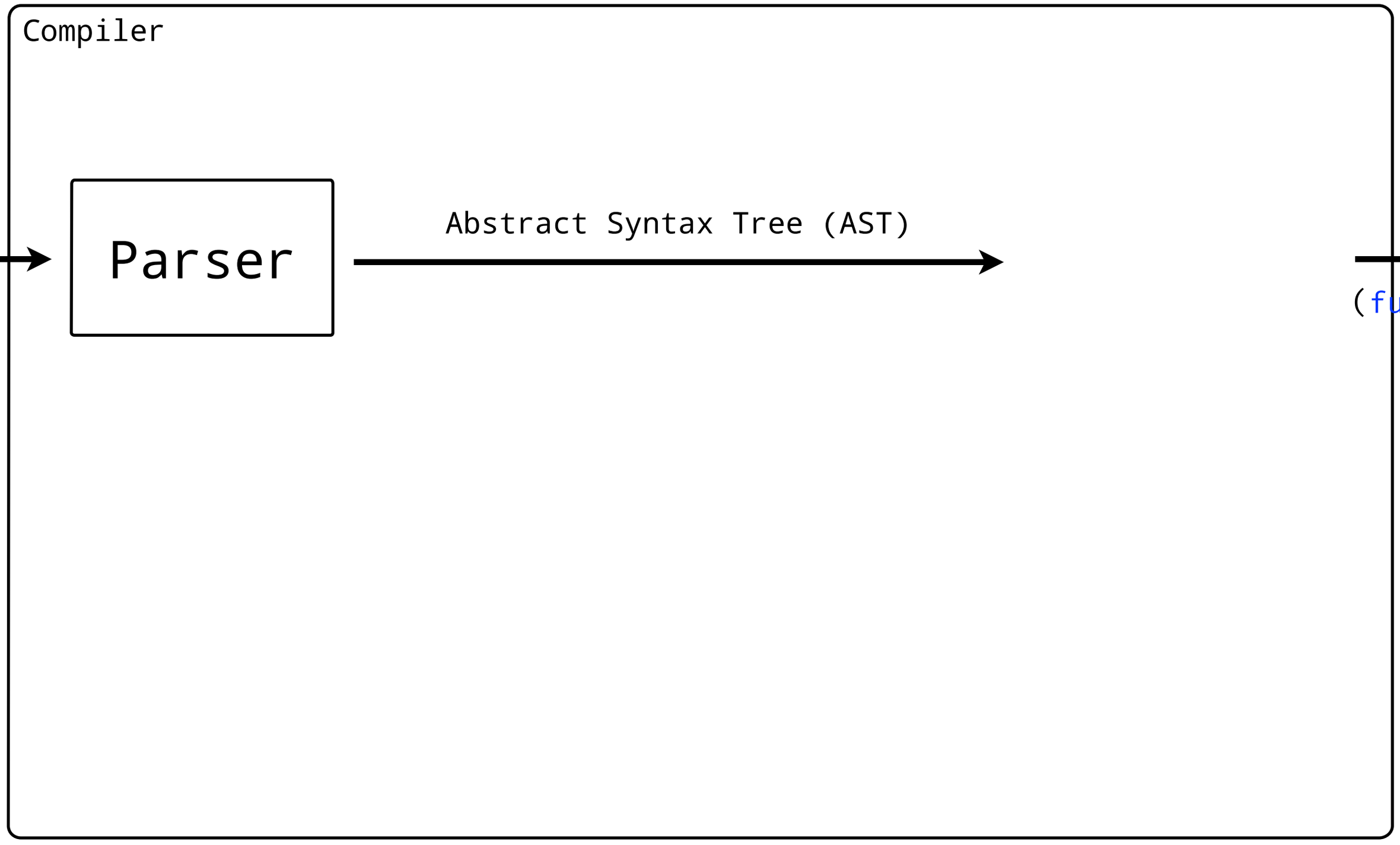
Compilers Overview



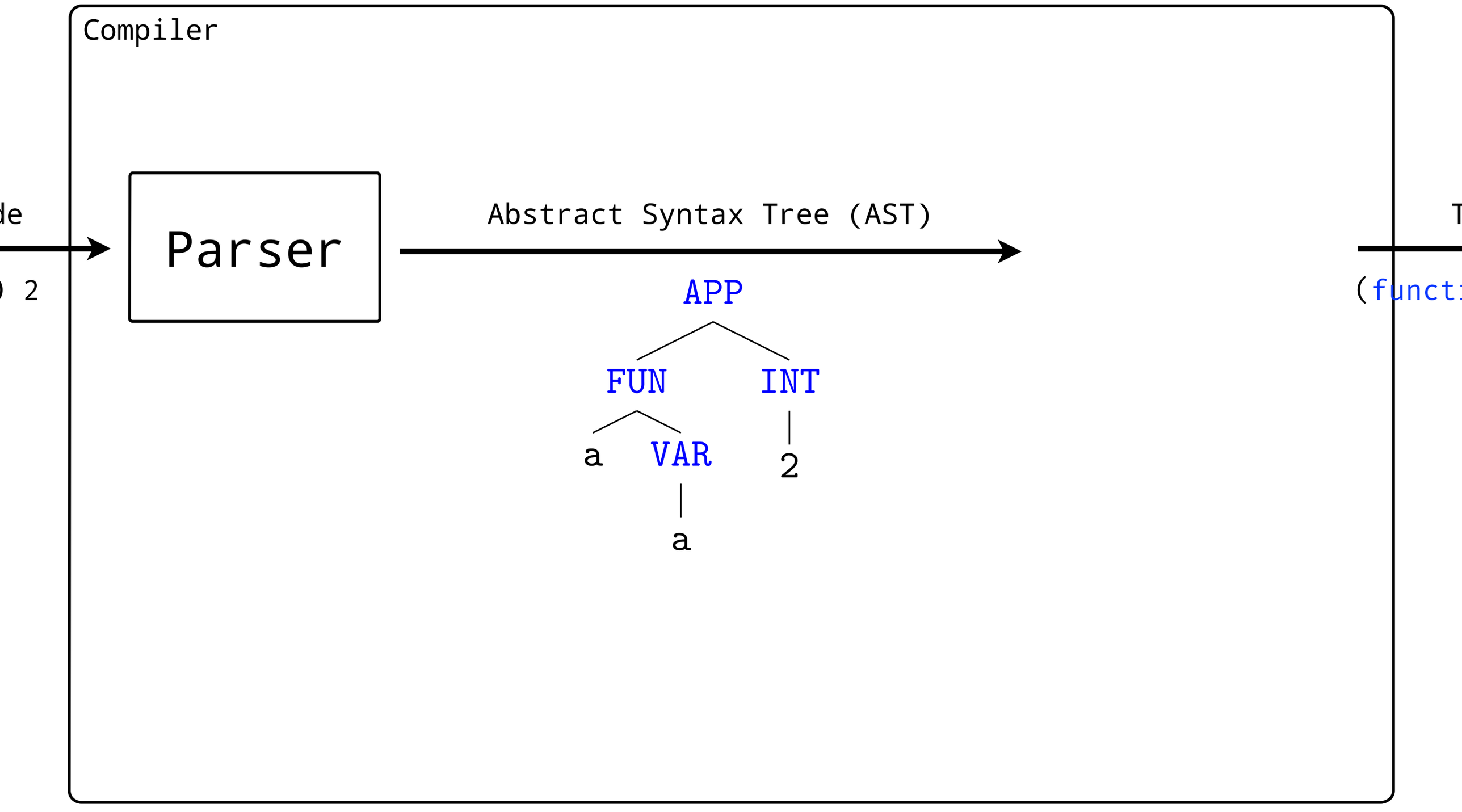
Parsing



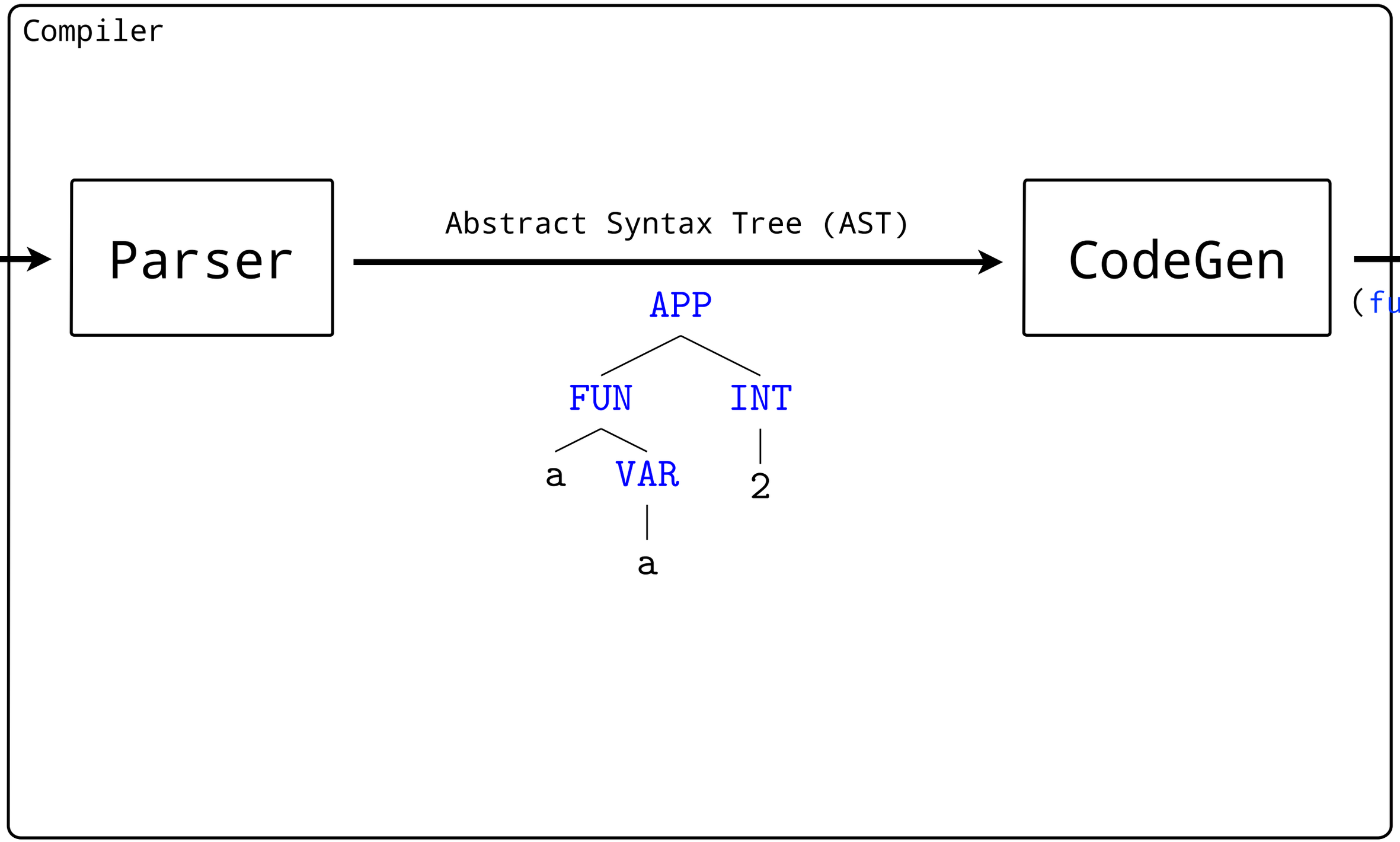
Abstract Syntax Tree



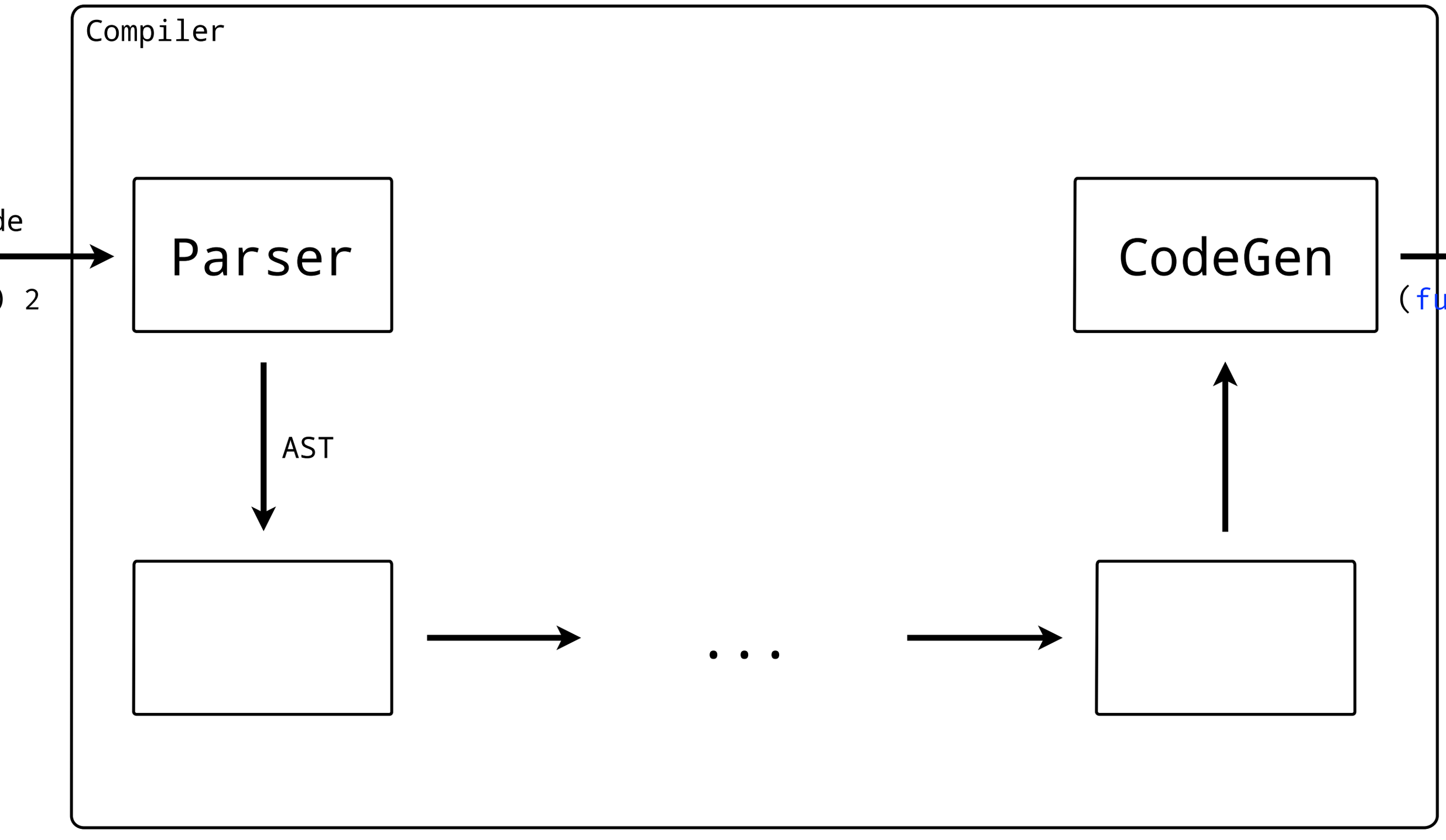
Abstract Syntax Tree



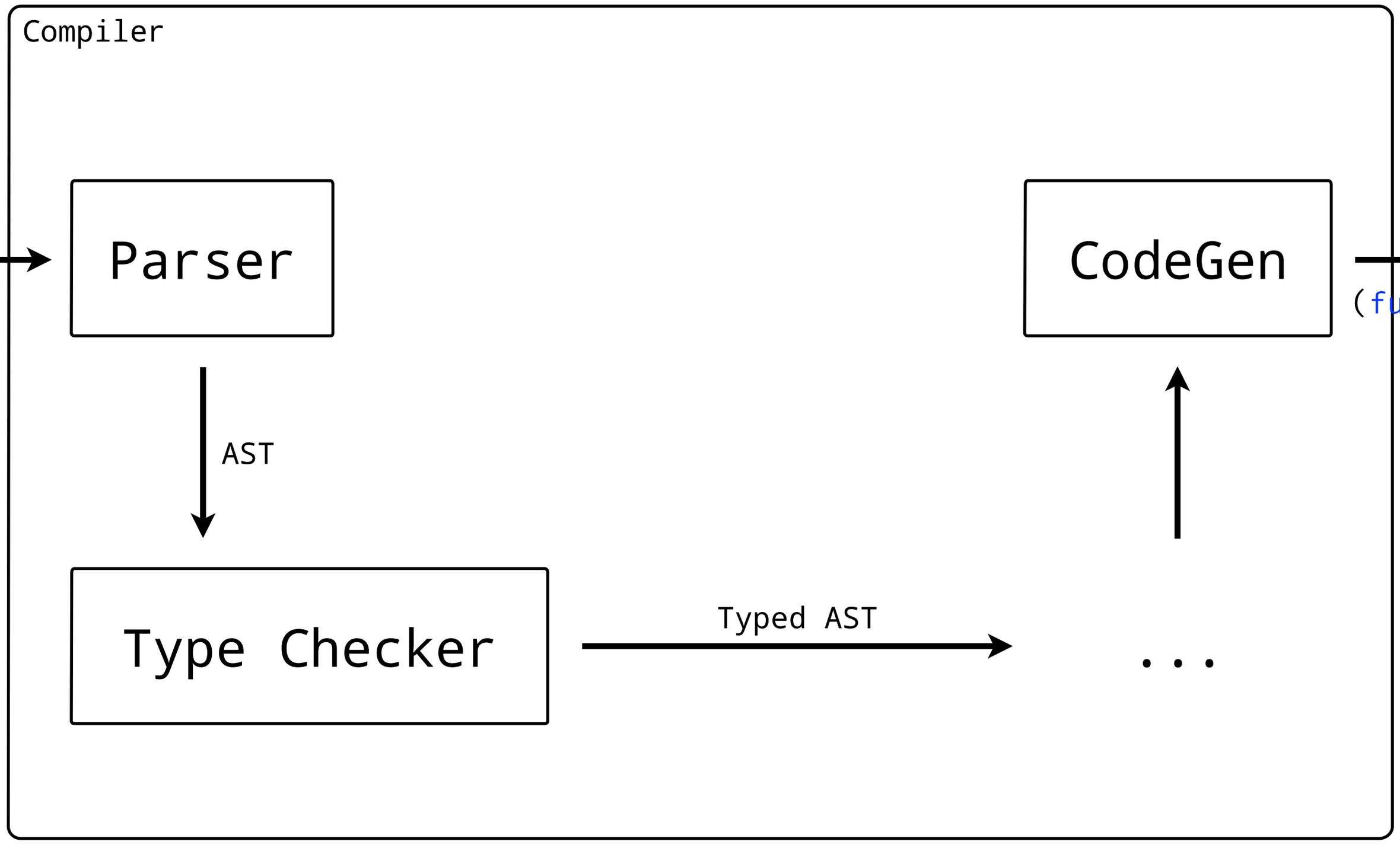
Code Generation



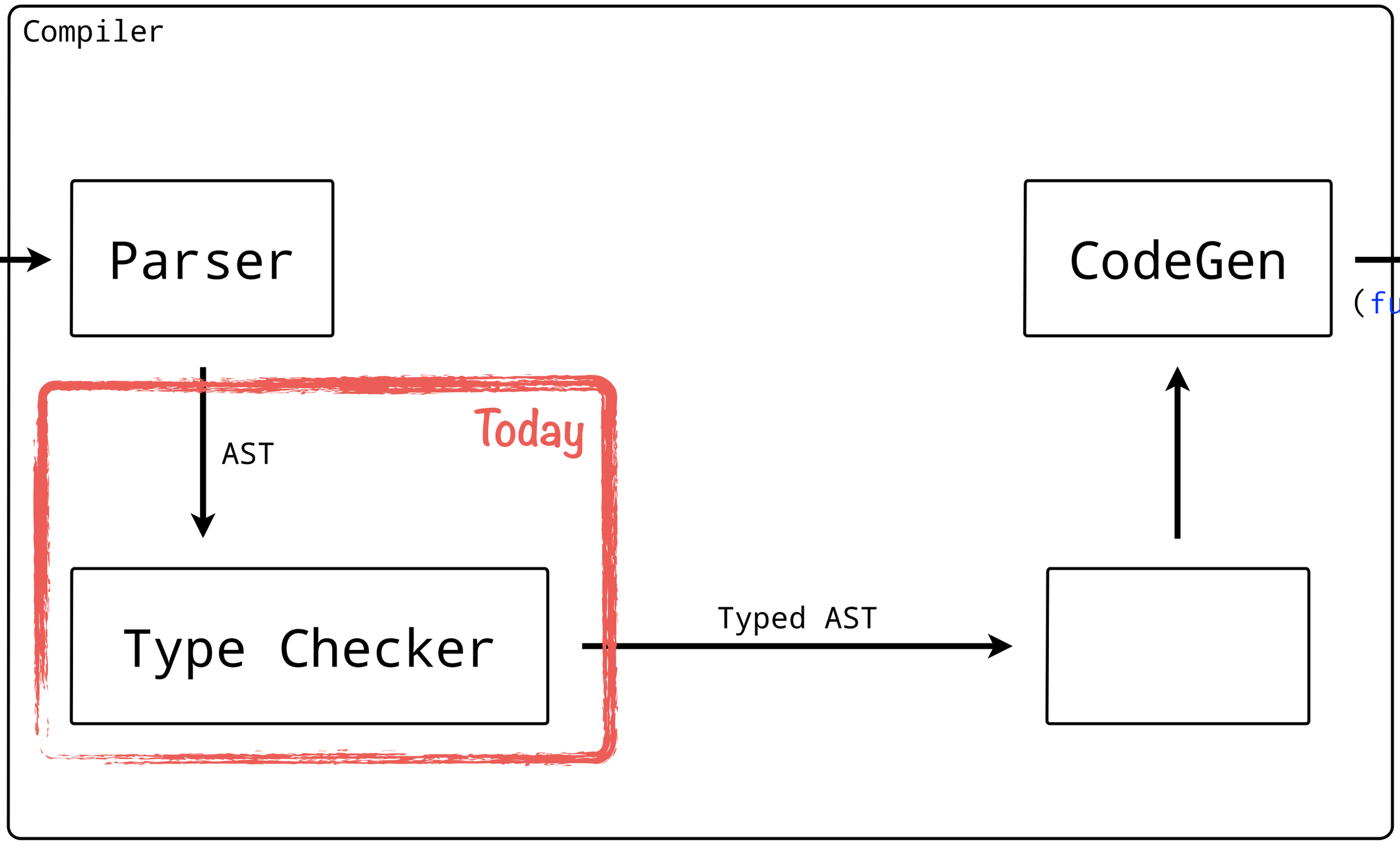
Many Intermediate Phases



Type Checking



Today's Talk


Compiler
de
) 2
Parser
CodeGen
T
(functi
AST
Today
Type Checker
Typed AST

Type Checking

vs

Type Inference

Type Checking vs Inference

- Type Checking
  - Ensures **declared** types are used consistently
  - All types must be declared
  - Traverse AST and compare def site with use site
- Type Inference
  - Ensures consistency as well
  - Types need not be declared, though; are **deduced**
  - Two main classes of algorithms
  - We'll see one instance today

Vehicle Language

Vehicle Language

- Surface Syntax
  - What's the concrete syntax of the language
- Type System
  - What types are supported by the language

Surface Syntax[1]

1. Numbers: 1, 2, 3, ...

1 — a subset of Standard ML

Surface Syntax[1]

1. Numbers: 1, 2, 3, ...
2. Booleans: `true` and `false`

1 — a subset of Standard ML

Surface Syntax[1]

1. Numbers: 1, 2, 3, ...
2. Booleans: `true` and `false`
3. Anonymous functions (lambdas): `fn a => a`

1 — a subset of Standard ML

Surface Syntax[1]

1. Numbers: `1`, `2`, `3`, ...
2. Booleans: `true` and `false`
3. Anonymous functions (lambdas): `fn a => a`
4. Function application: `inc 42`

1 — a subset of Standard ML

Surface Syntax[1]

1. Numbers: `1`, `2`, `3`, ...
2. Booleans: `true` and `false`
3. Anonymous functions (lambdas): `fn a => a`
4. Function application: `inc 42`
5. If expressions: `if cond then t else f`

1 — a subset of Standard ML

Surface Syntax[1]

6. Let blocks/expressions:

```
let
   val name = ...
in
   name
end
```

1 — a subset of Standard ML

Small Example

```
let
  val inc = fn a => a + 1
in
  inc 42
end
```

Language Types

1. Integer type: `int`

Language Types

1. Integer type: `int`
2. Boolean type: `bool`

Language Types

1. Integer type: `int`
2. Boolean type: `bool`
3. Function type: `int -> bool`

Language Types

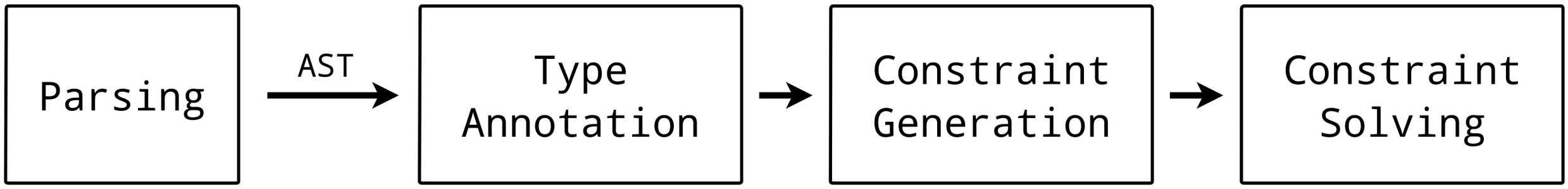
1. Integer type: `int`
2. Boolean type: `bool`
3. Function type: `int -> bool`
4. Generic type variables: `'a`, `'b`, `'c`, etc.

Today's Algorithm Overview

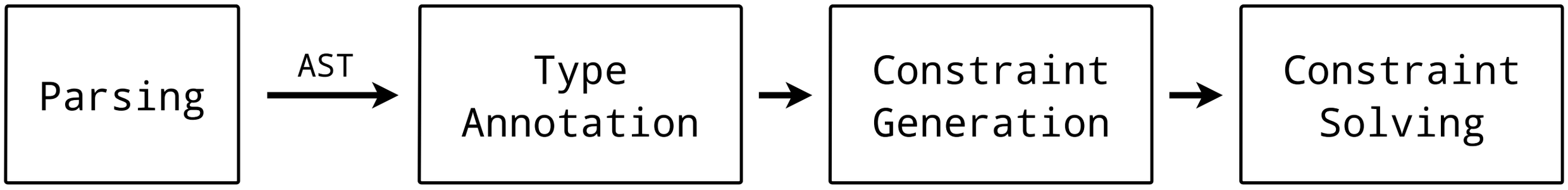
Wand's Algorithm

Overview

Wand's Algorithm Overview

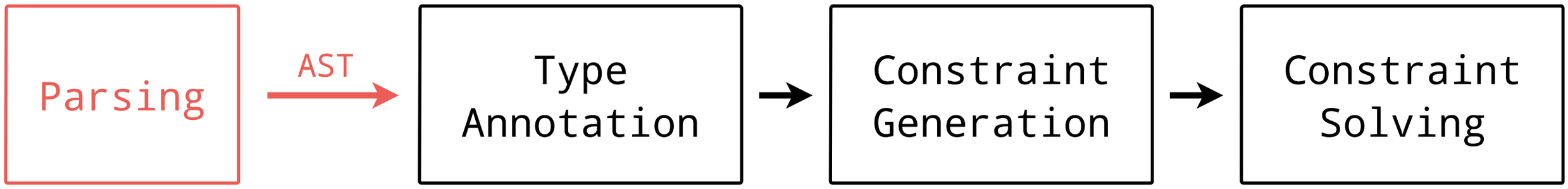


Wand's Algorithm Overview

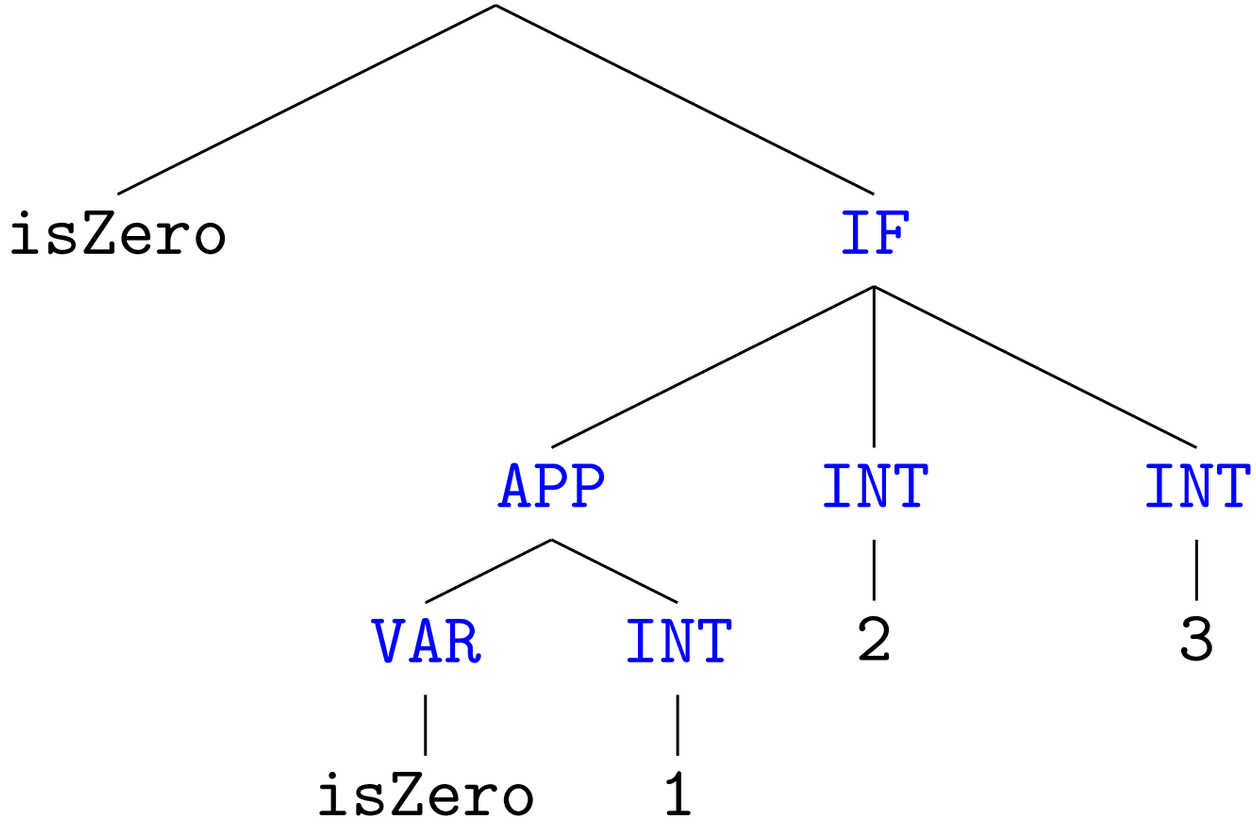



```
fn isZero =>
  if isZero 1
  then 2
  else 3
```

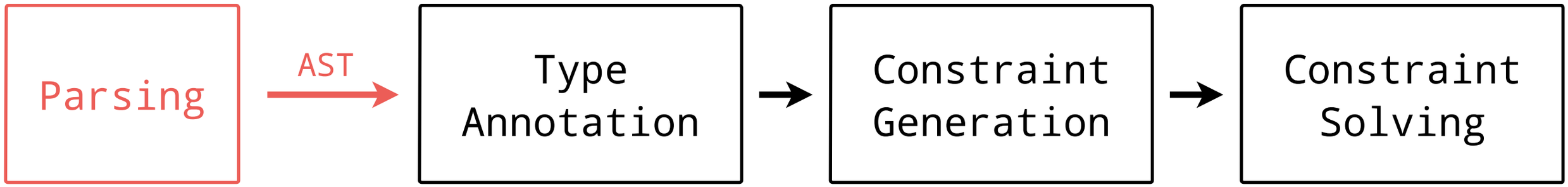
Wand's Algorithm Overview



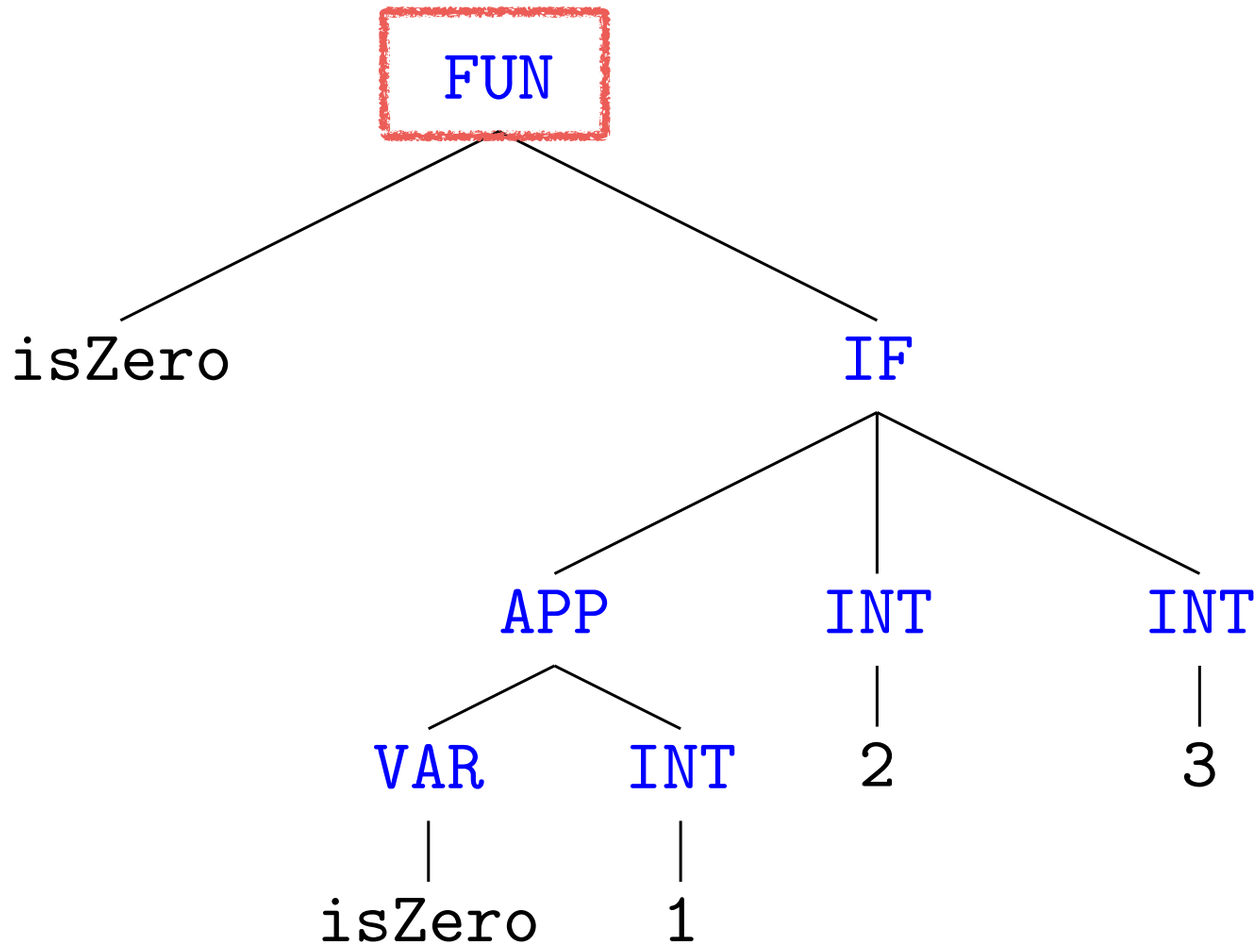

```
fn isZero =>
  if isZero 1
  then 2
  else 3
```



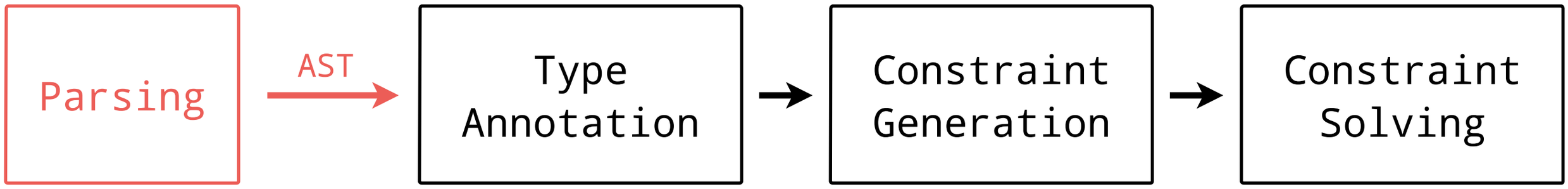
Wand's Algorithm Overview



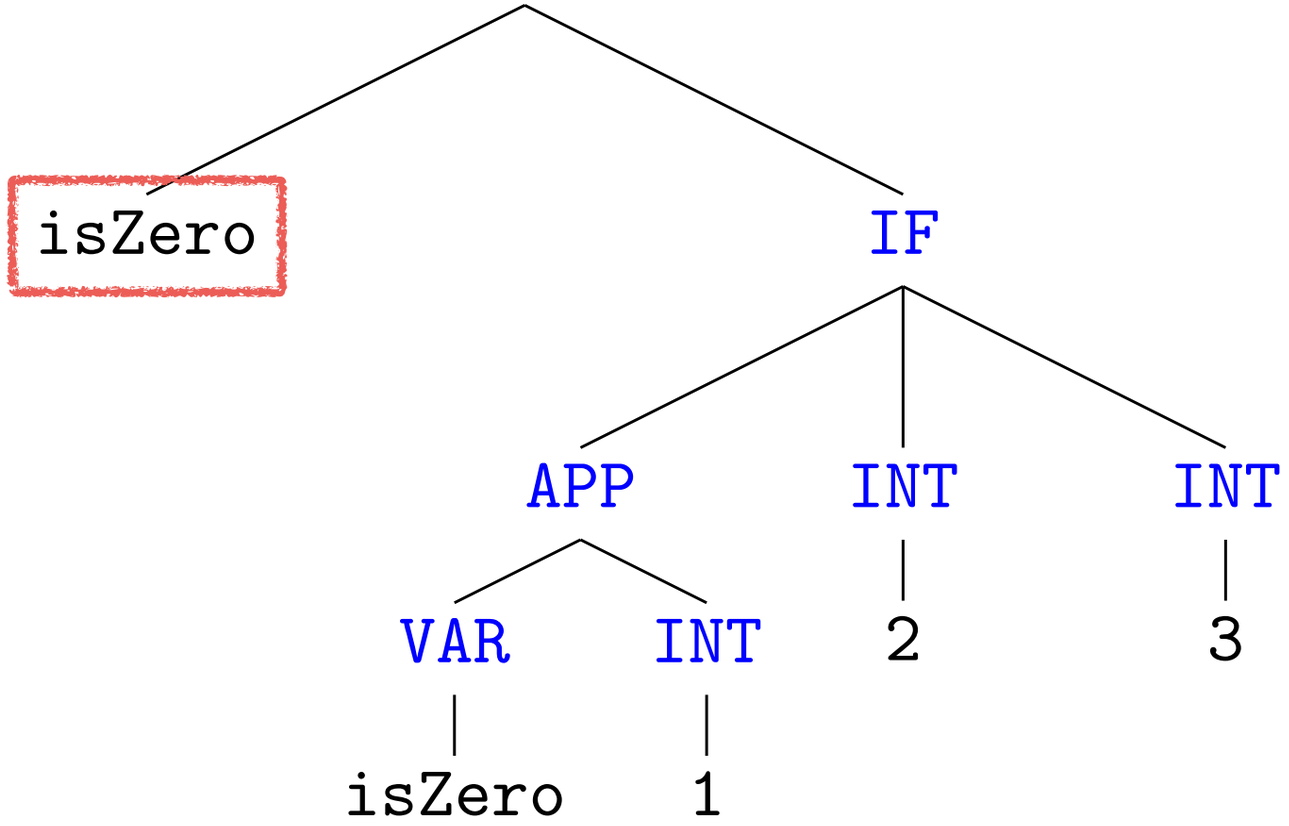

```
fn isZero =>
  if isZero 1
  then 2
  else 3
```



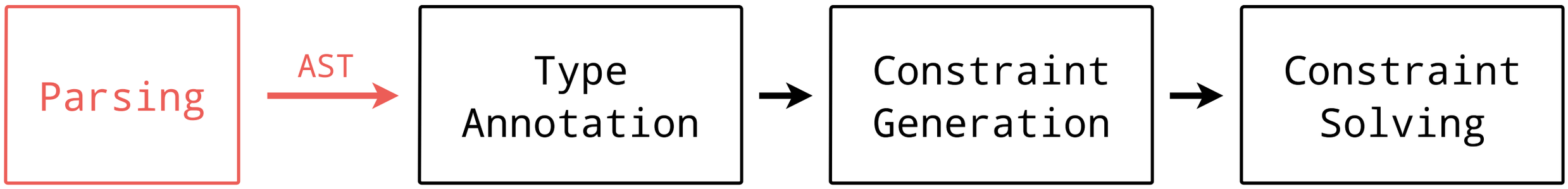
Wand's Algorithm Overview



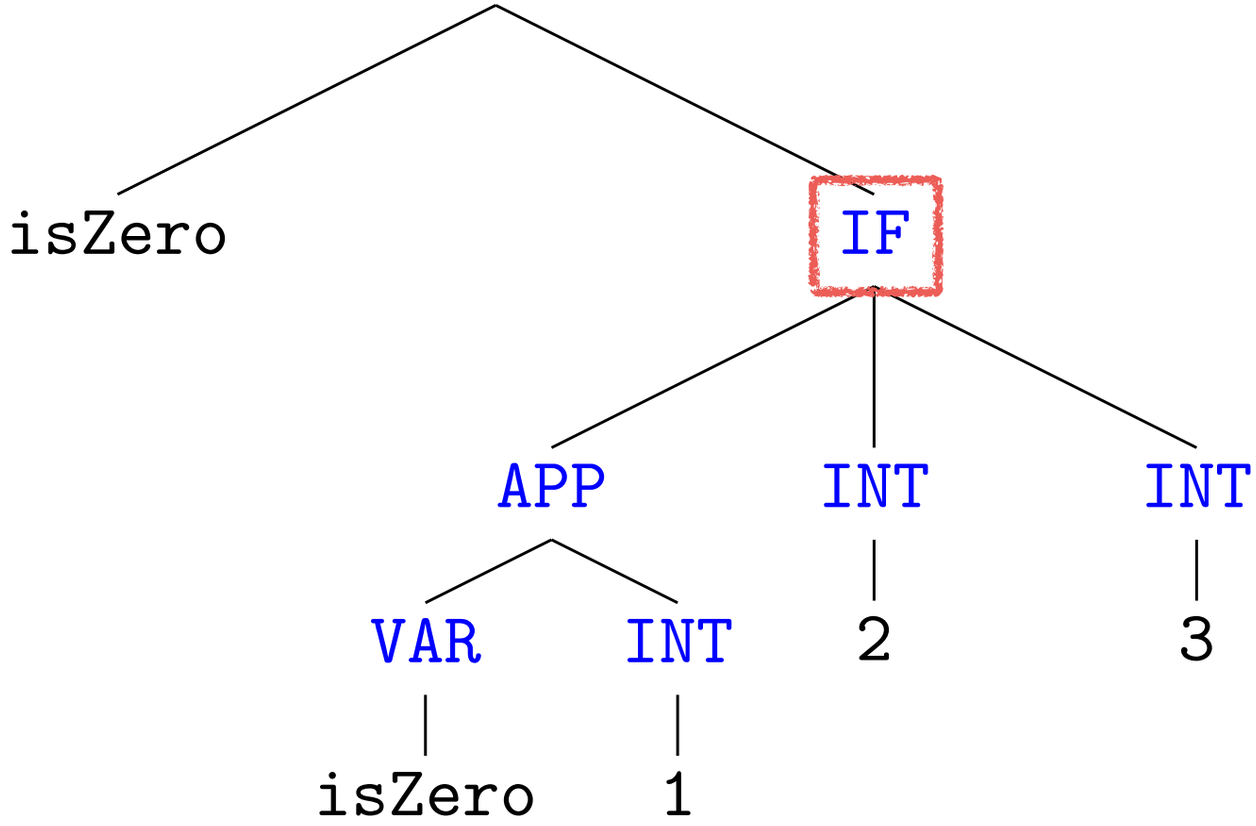

```
fn isZero =>
  if isZero 1
  then 2
  else 3
```



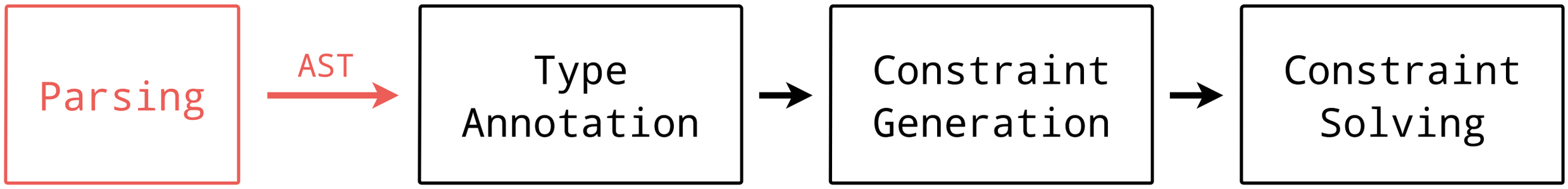
Wand's Algorithm Overview



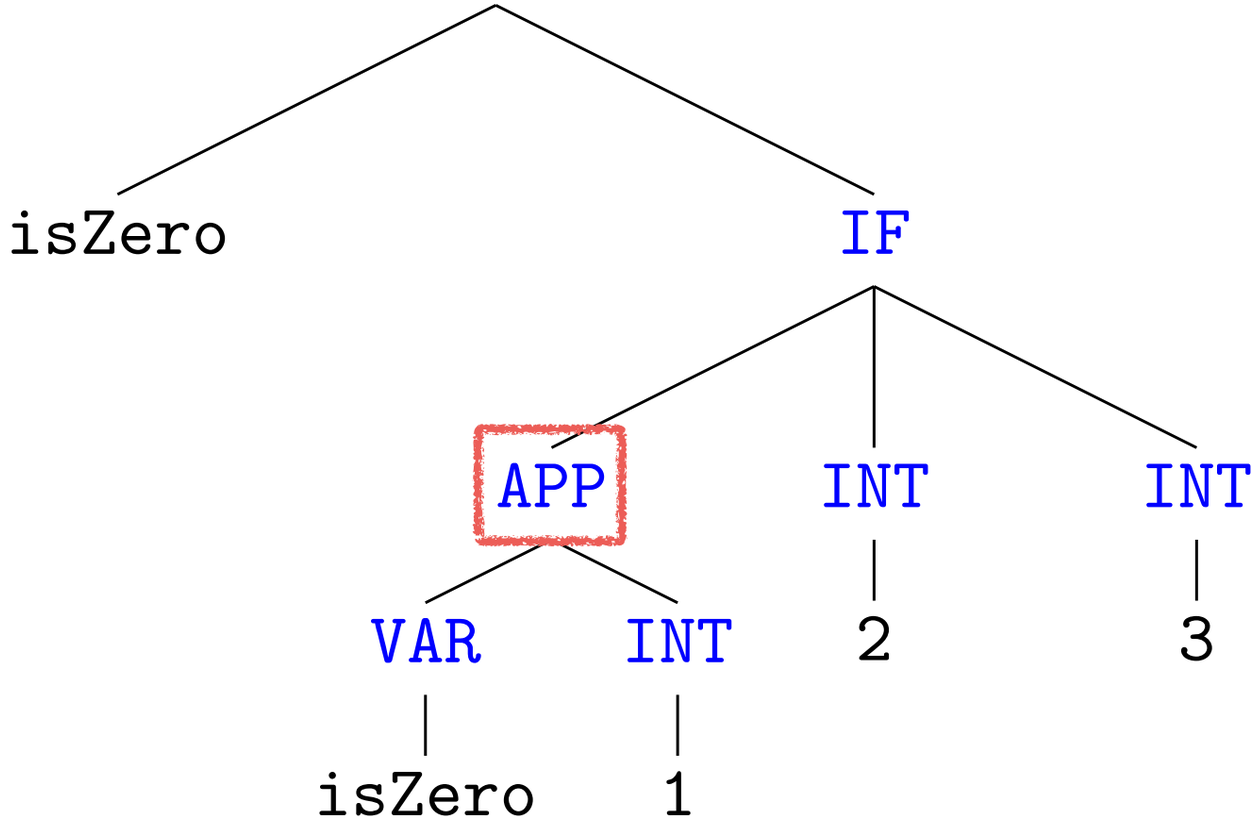

```
fn isZero =>
  if isZero 1
  then 2
  else 3
```



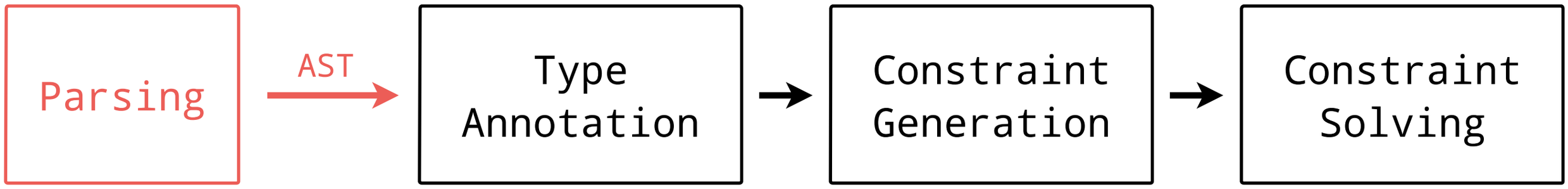
Wand's Algorithm Overview



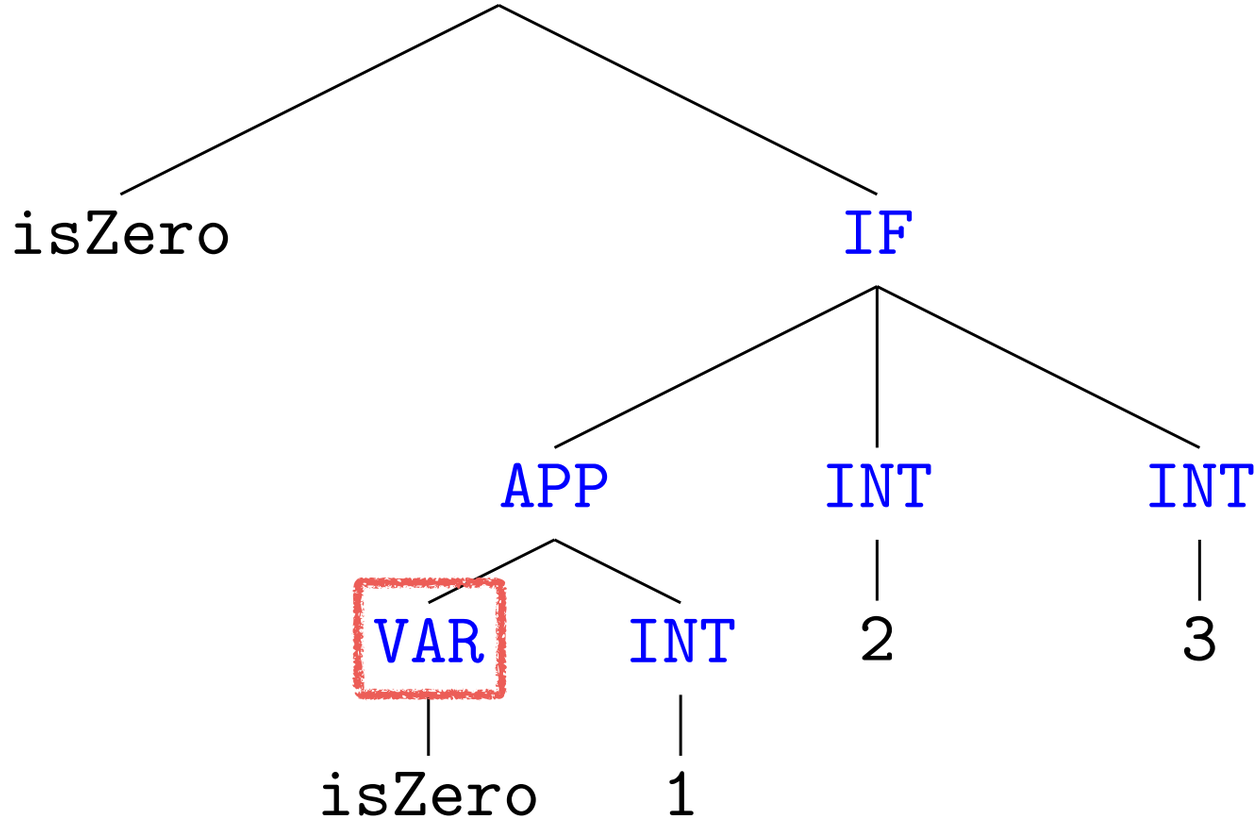

```
fn isZero =>
  if isZero 1
  then 2
  else 3
```



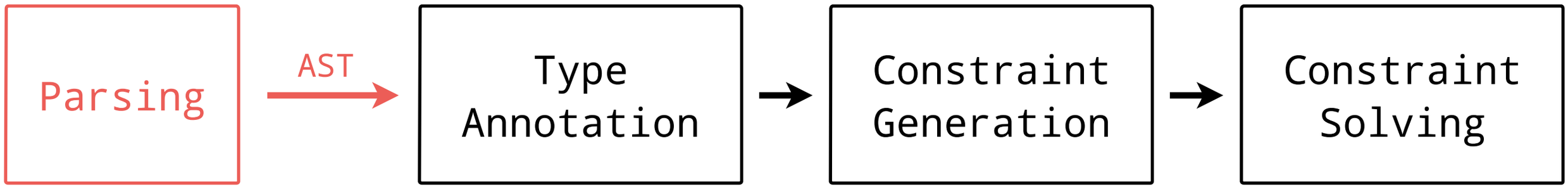
Wand's Algorithm Overview



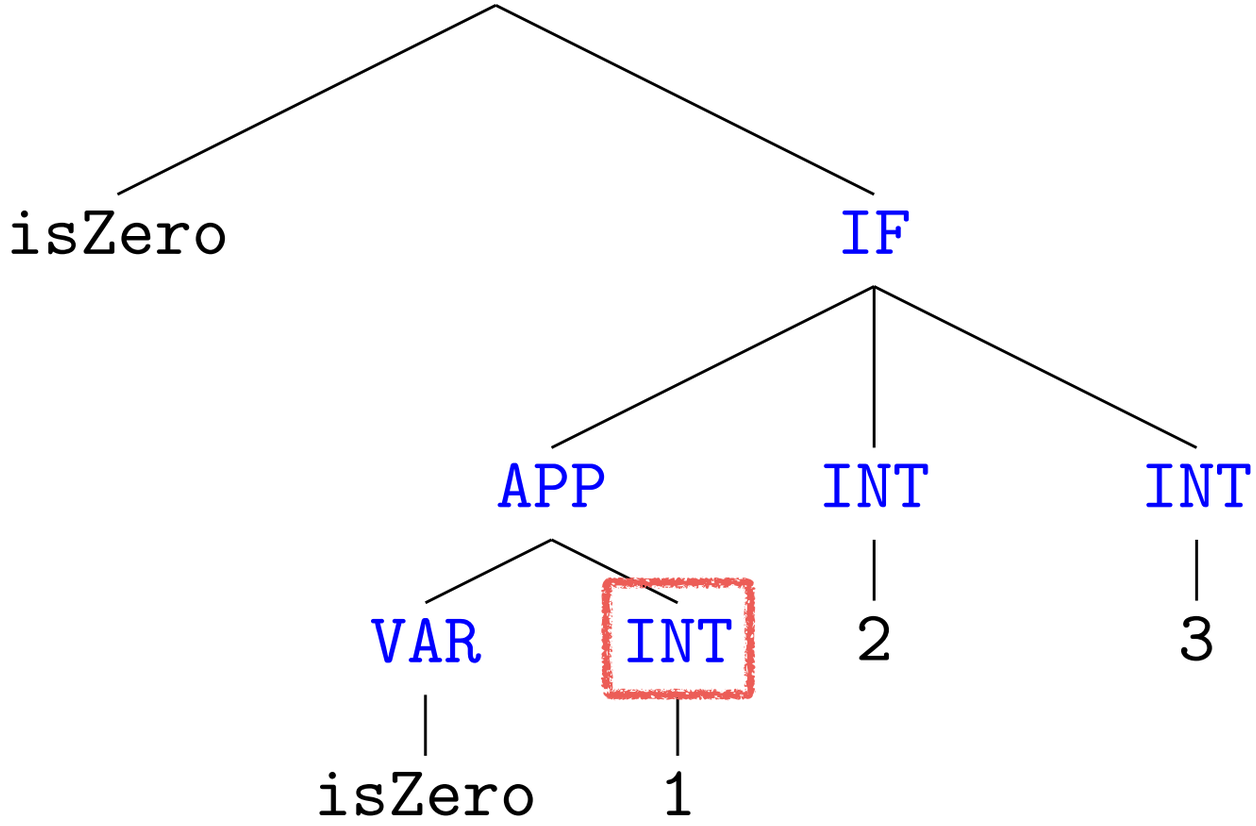

```
fn isZero =>
  if isZero 1
  then 2
  else 3
```



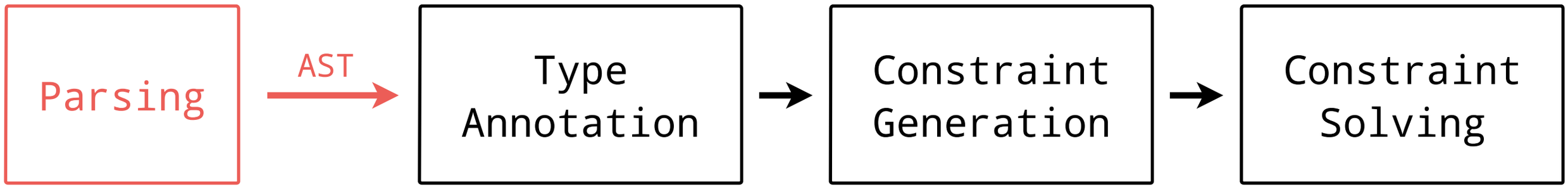
Wand's Algorithm Overview



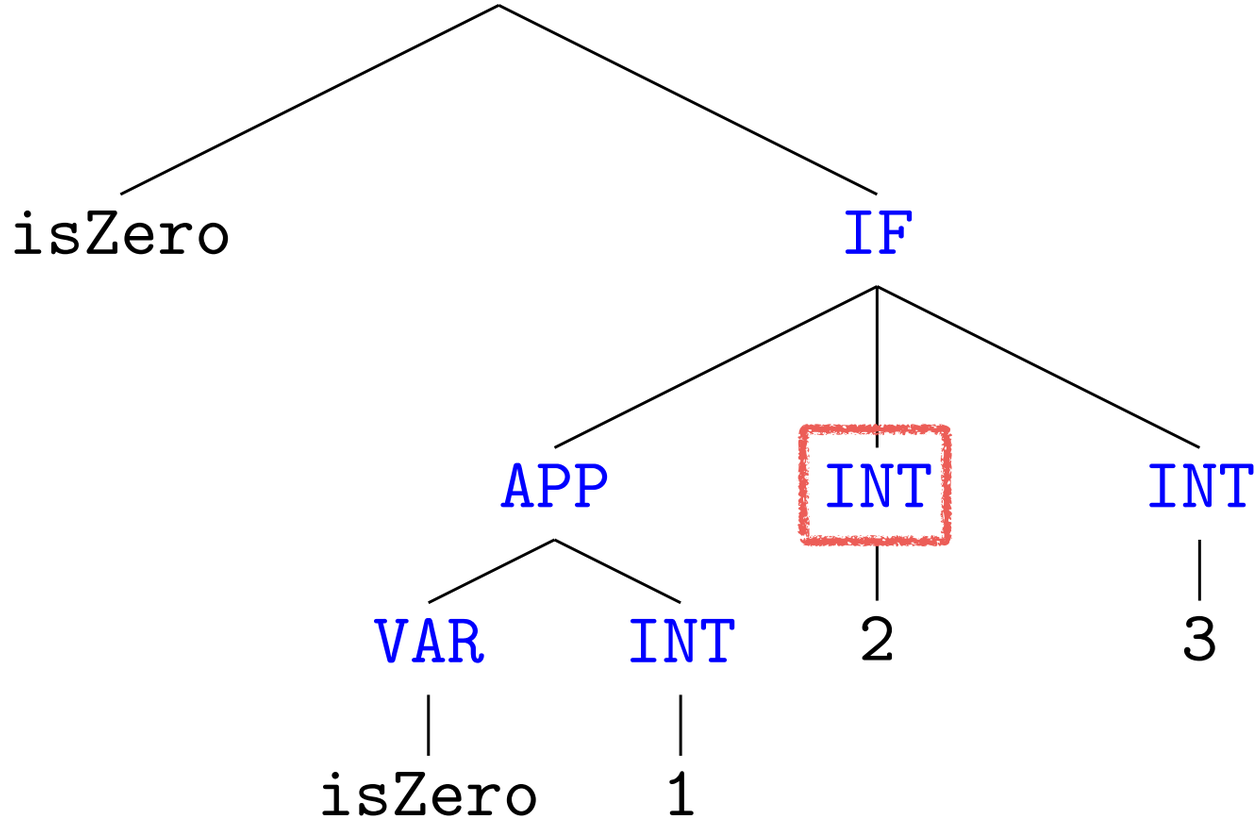

```
fn isZero =>
  if isZero 1
  then 2
  else 3
```



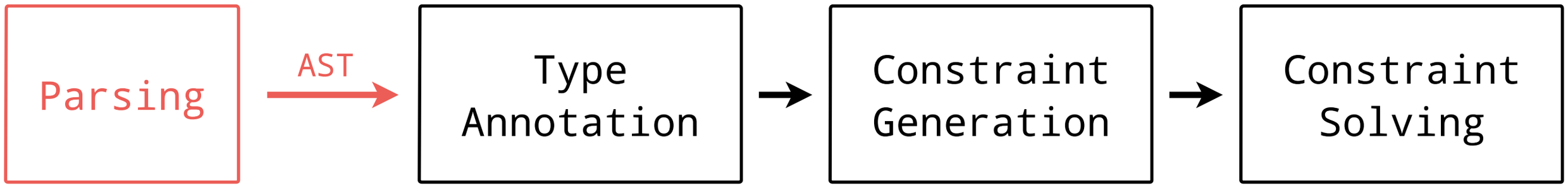
Wand's Algorithm Overview



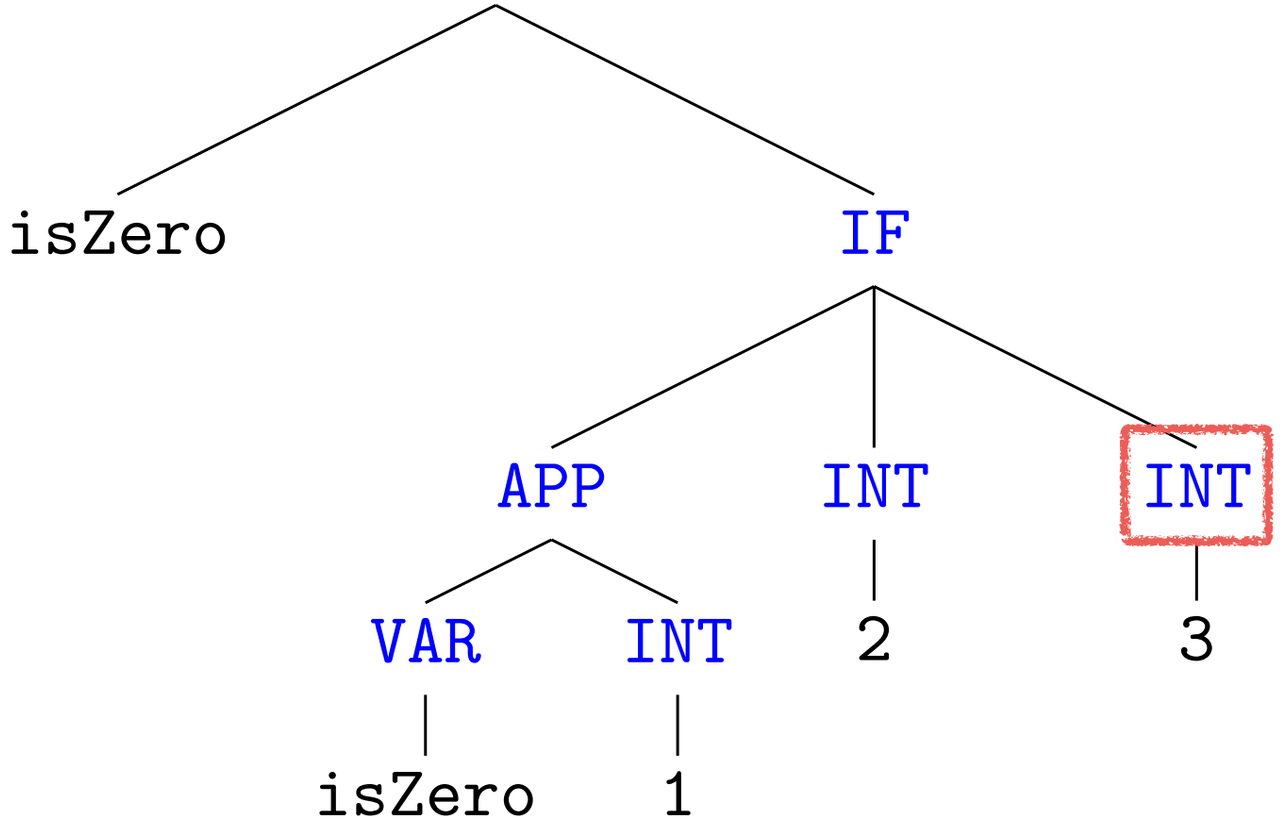

```
fn isZero =>
  if isZero 1
  then 2
  else 3
```



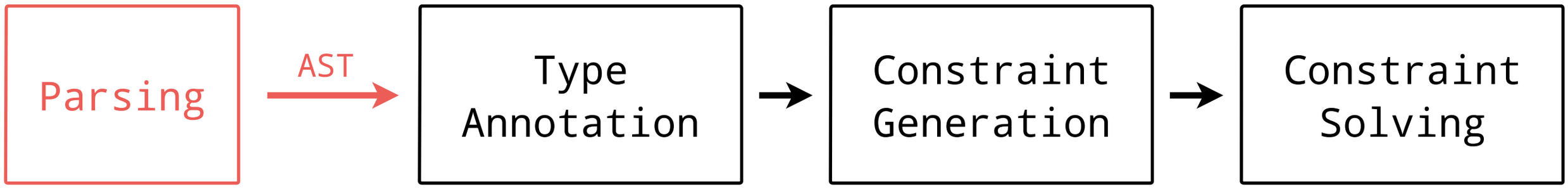
Wand's Algorithm Overview



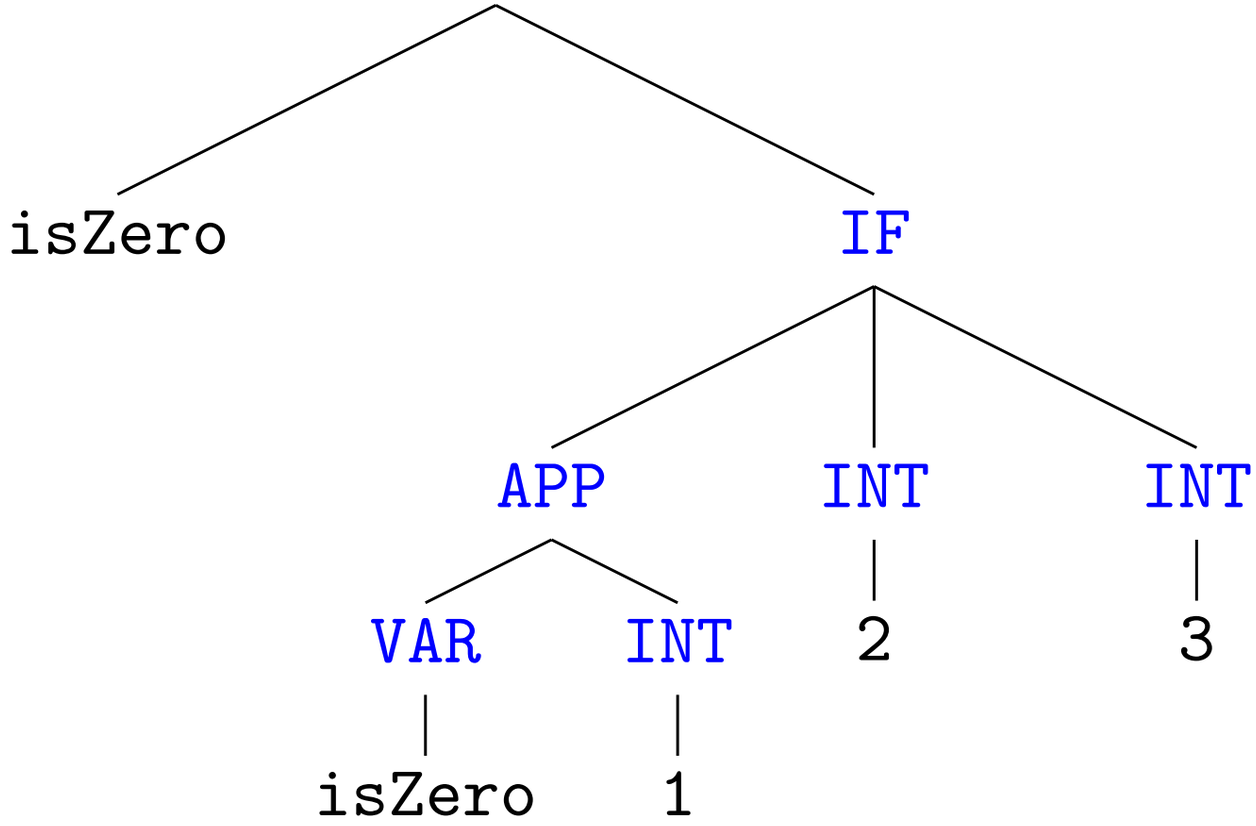

```
fn isZero =>
  if isZero 1
  then 2
  else 3
```



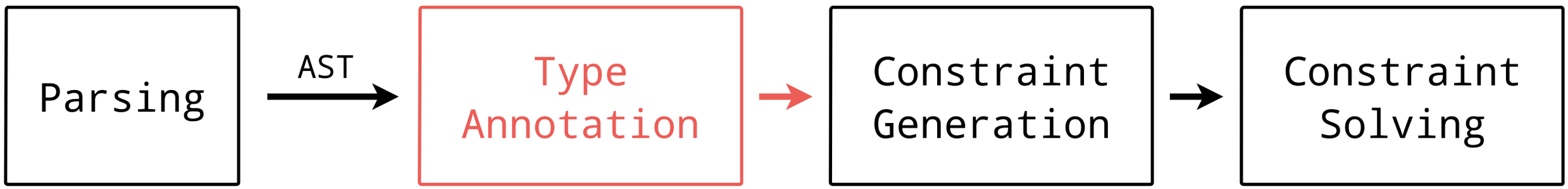
Wand's Algorithm Overview

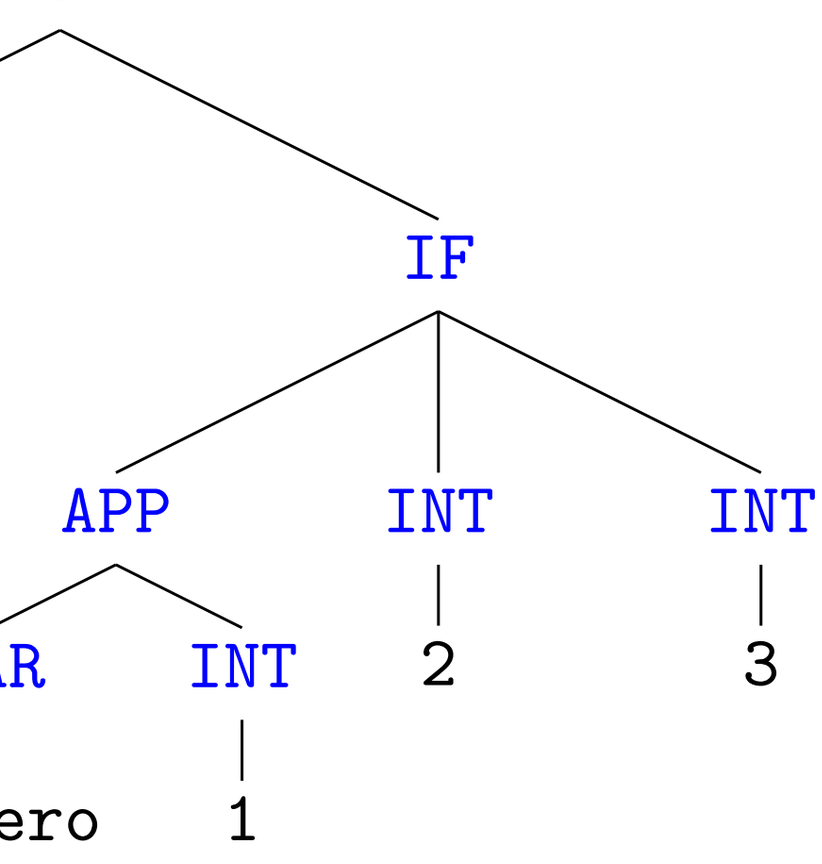


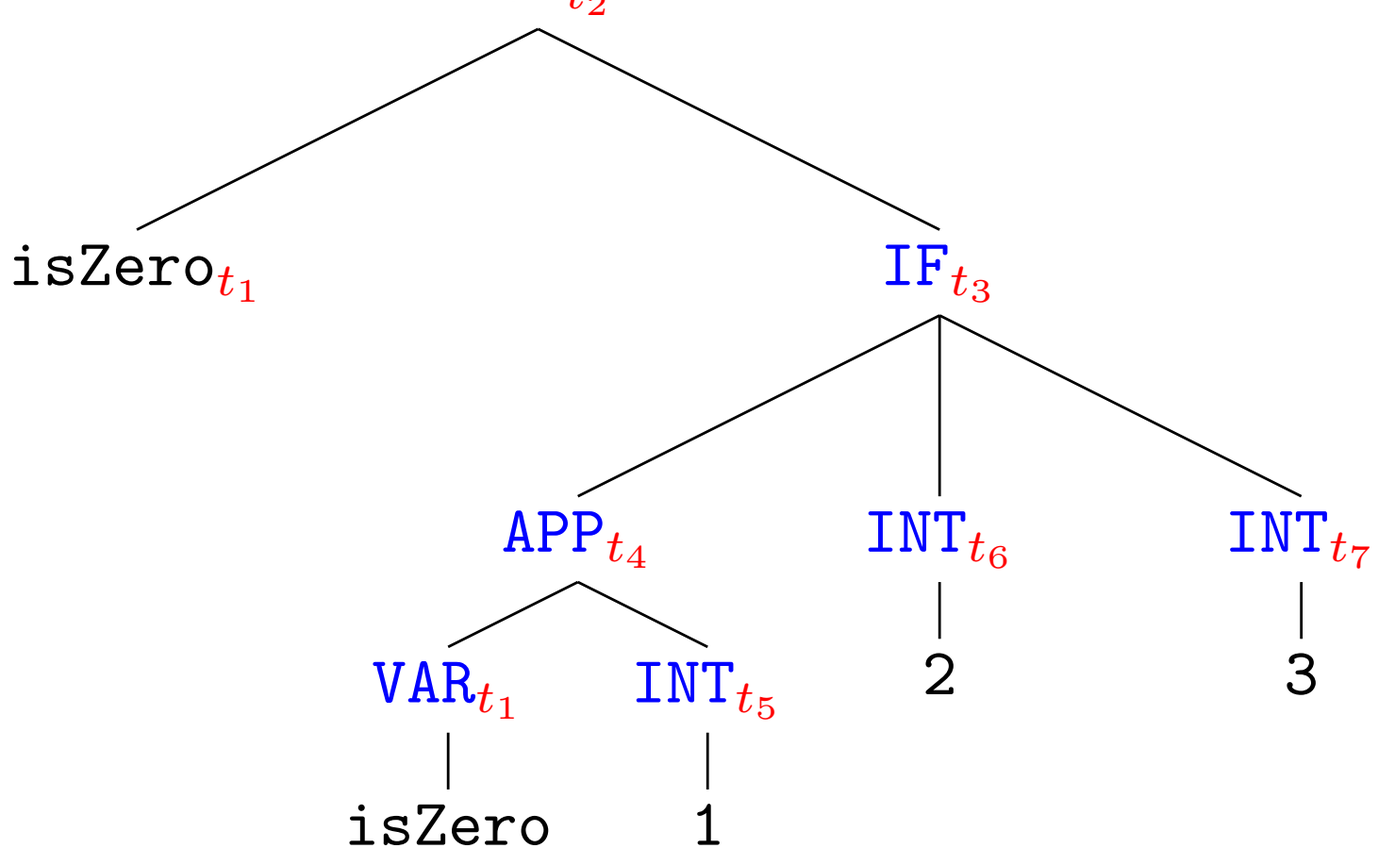

```
fn isZero =>
  if isZero 1
  then 2
  else 3
```



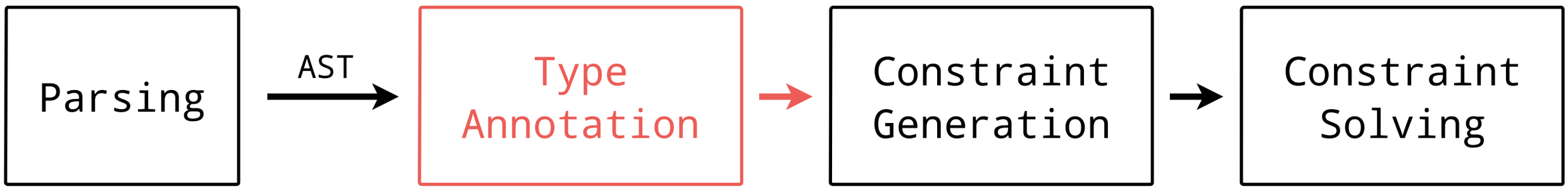
Wand's Algorithm Overview

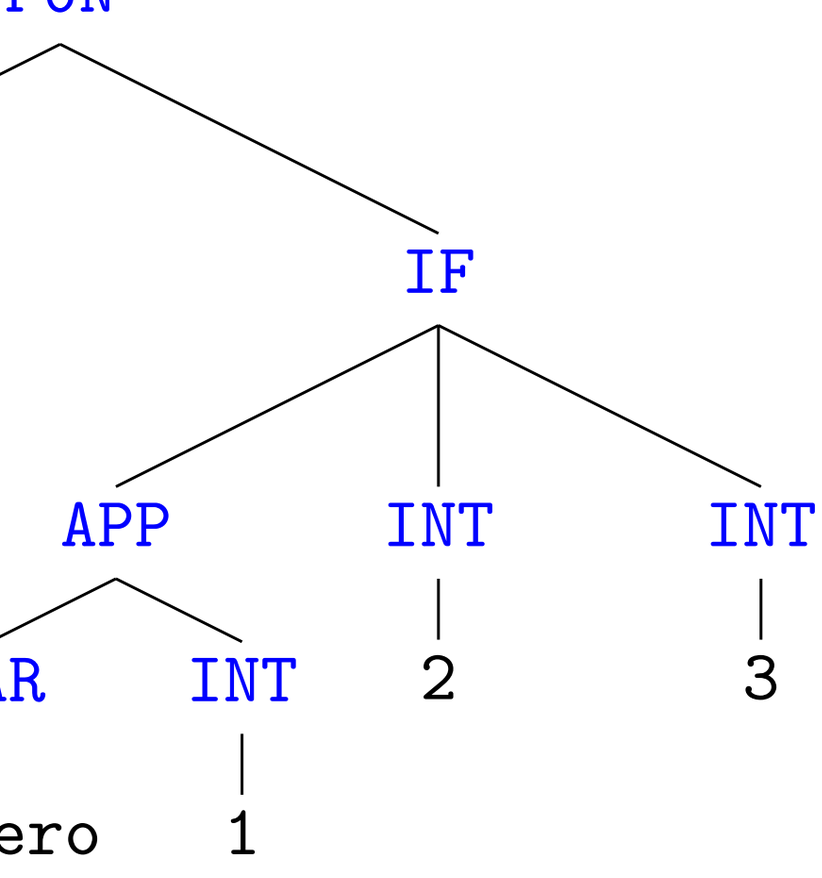


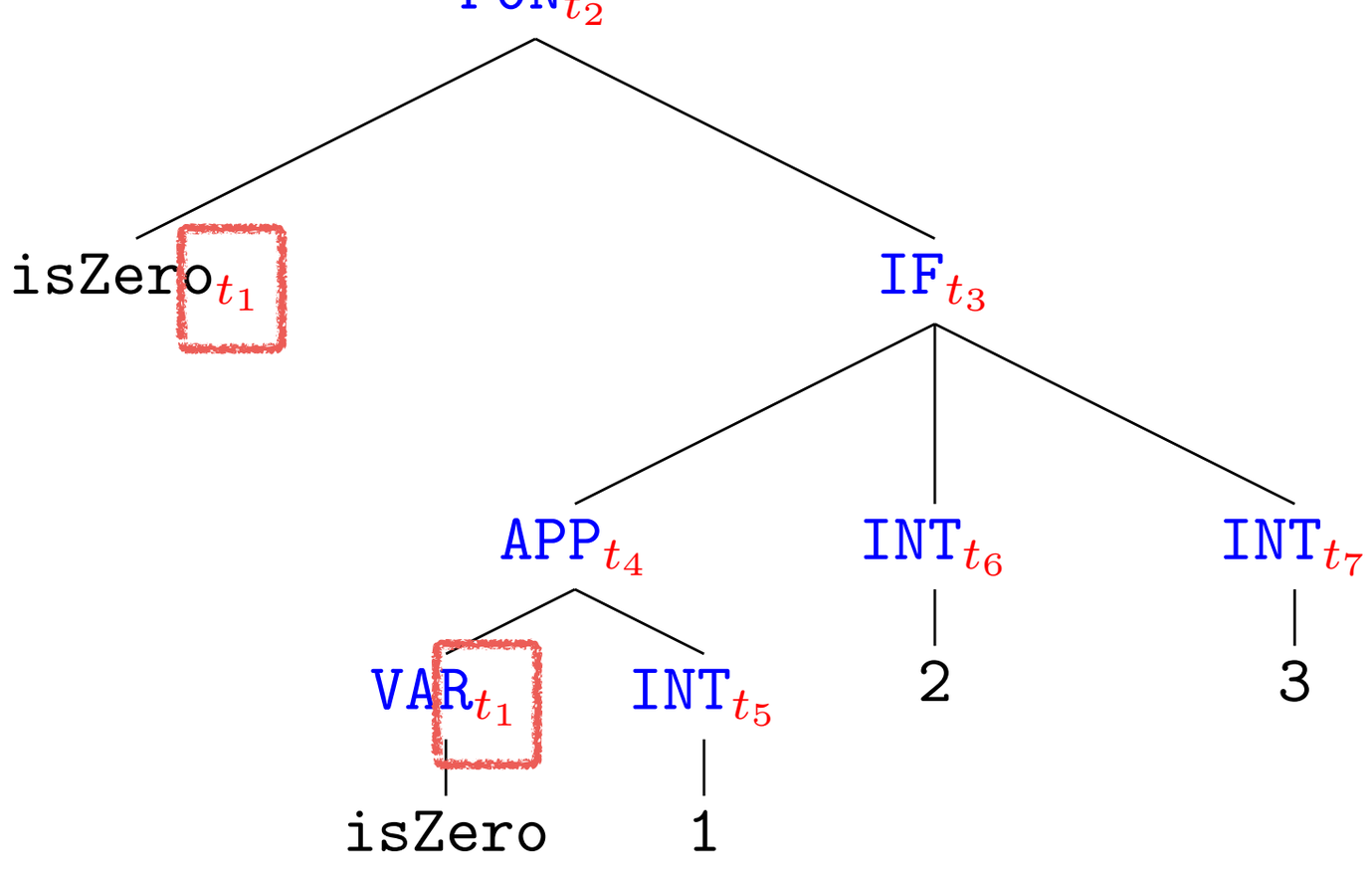




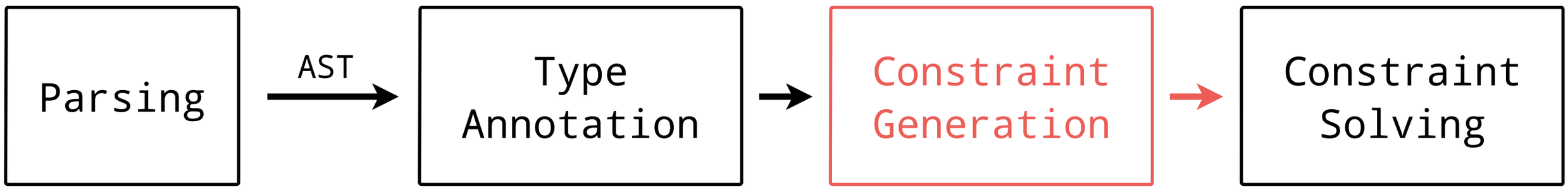
Wand's Algorithm Overview

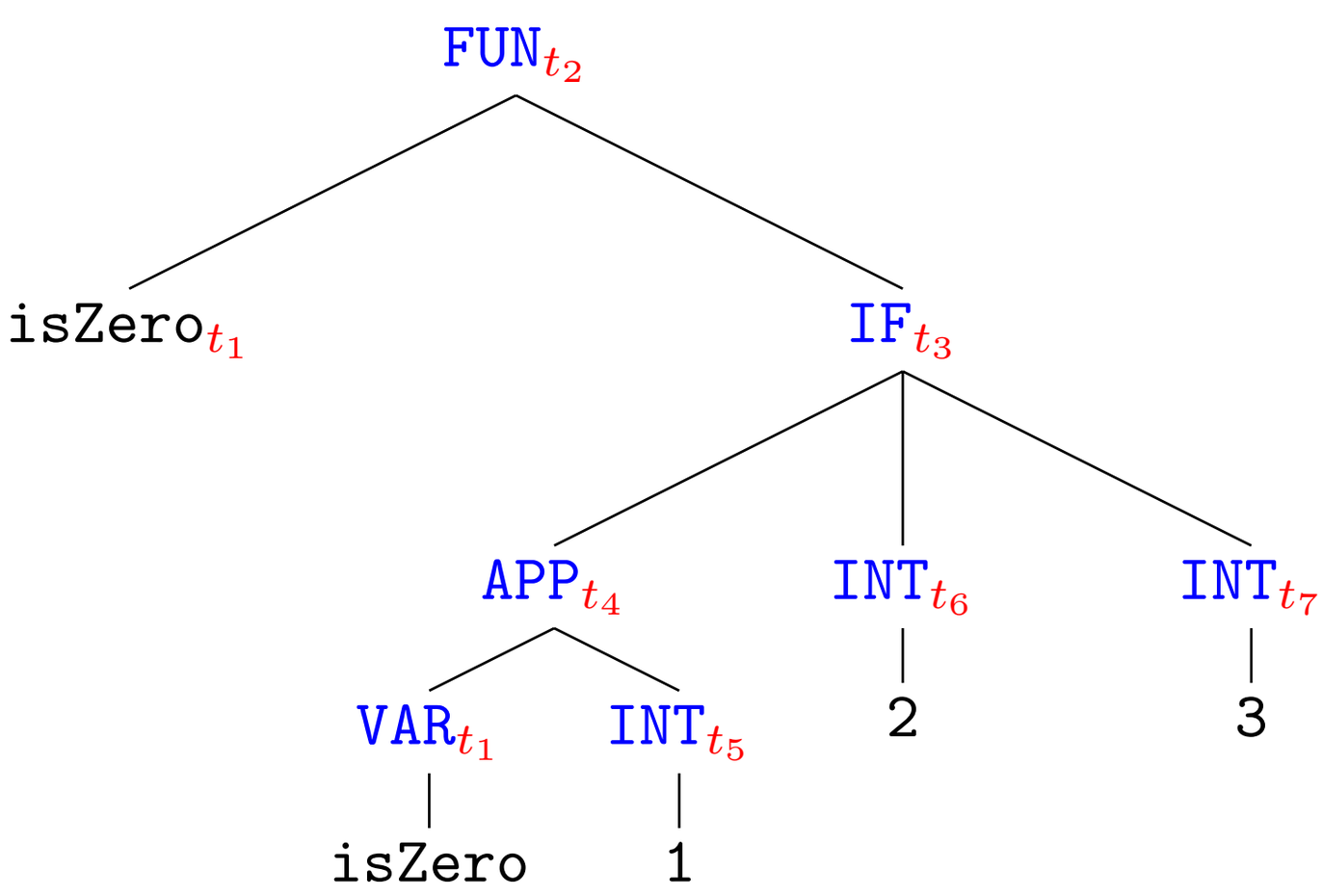






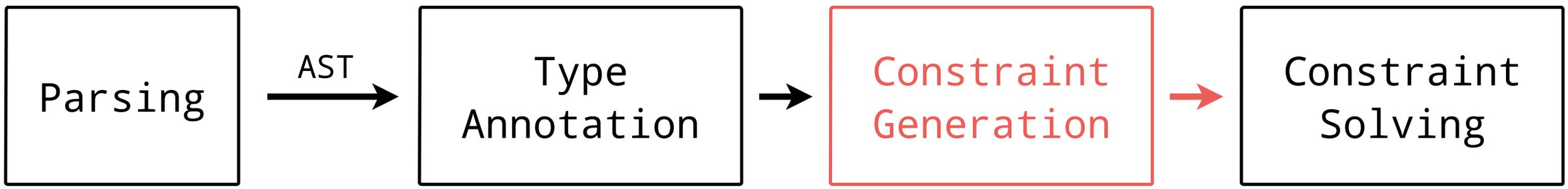
Wand's Algorithm Overview

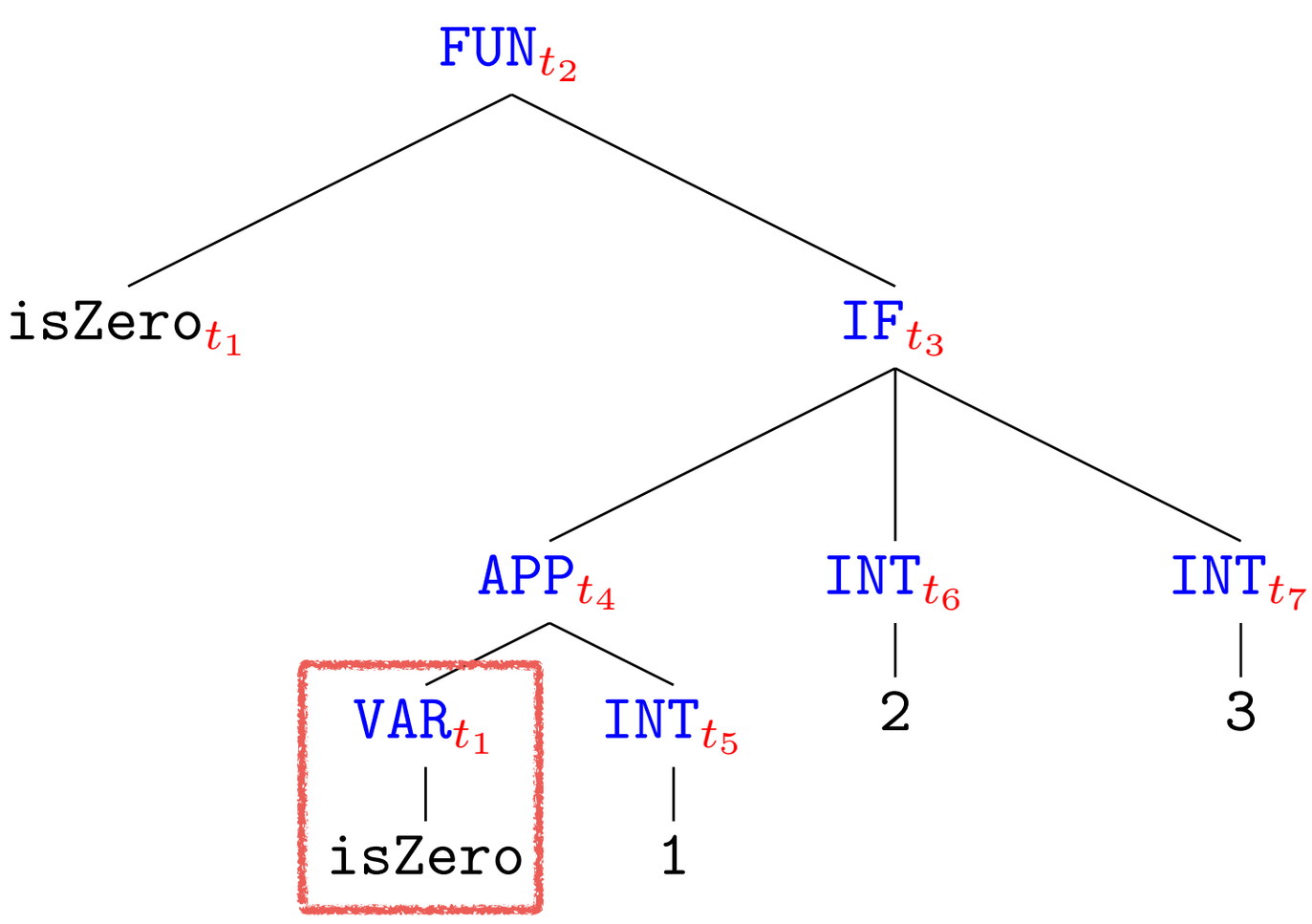




Constraint Set

Wand's Algorithm Overview

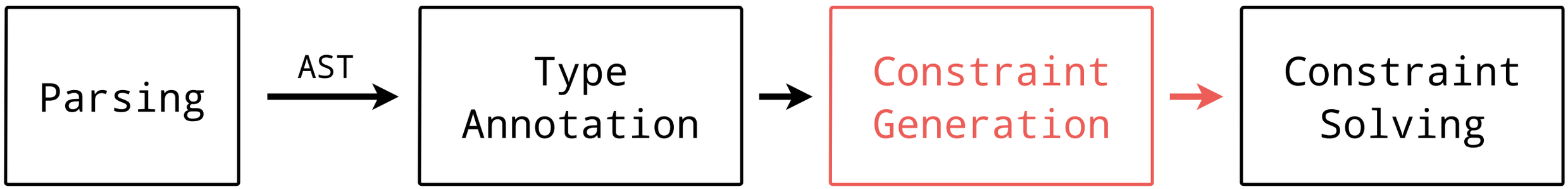


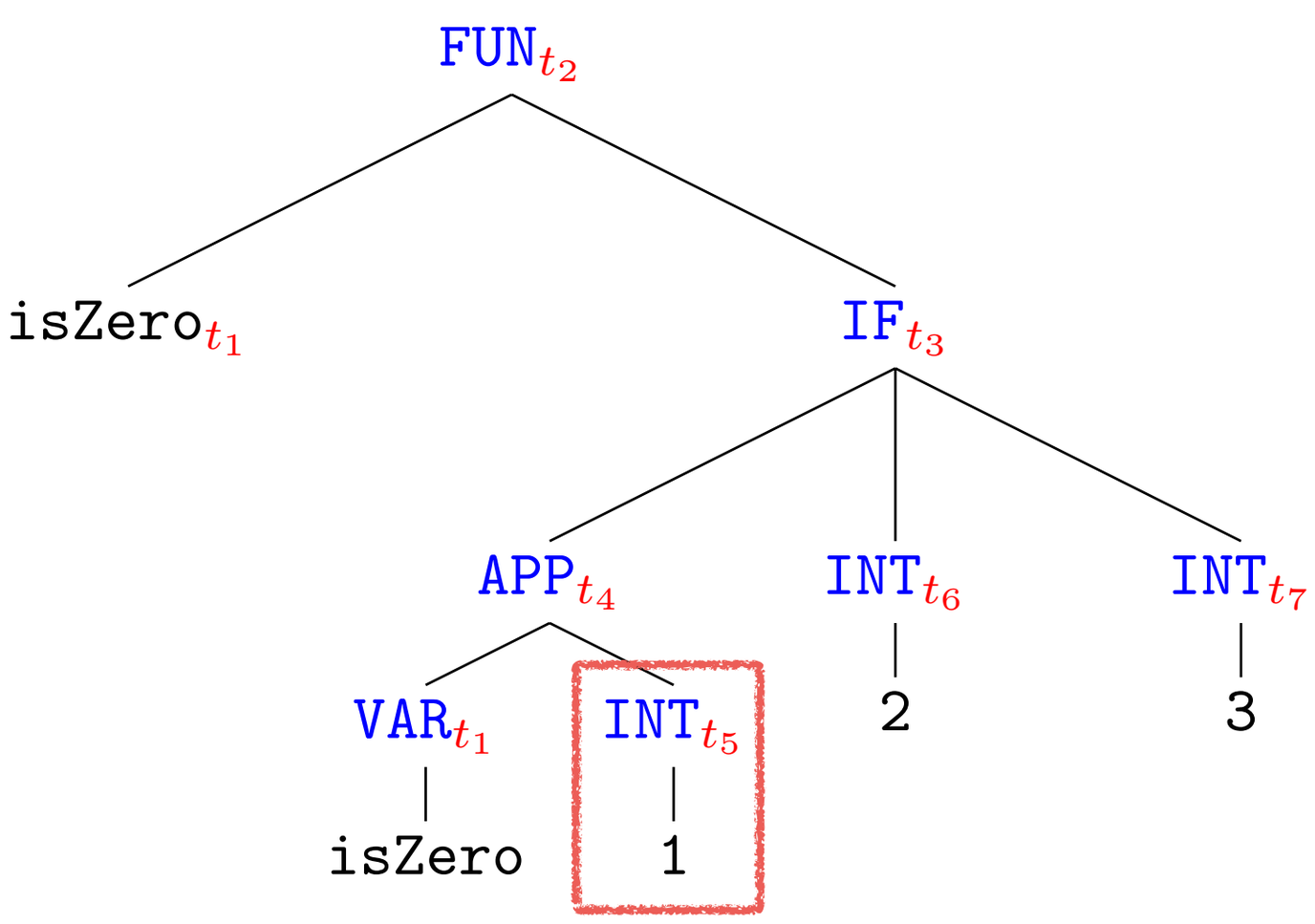


Constraint Set

```
t1 ≡ t5 → t4
```

Wand's Algorithm Overview

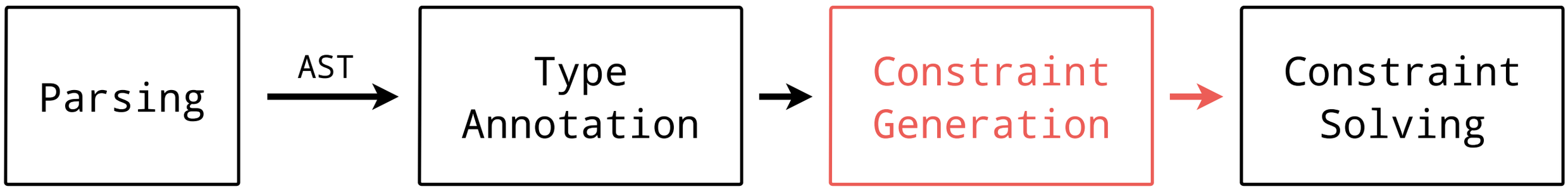


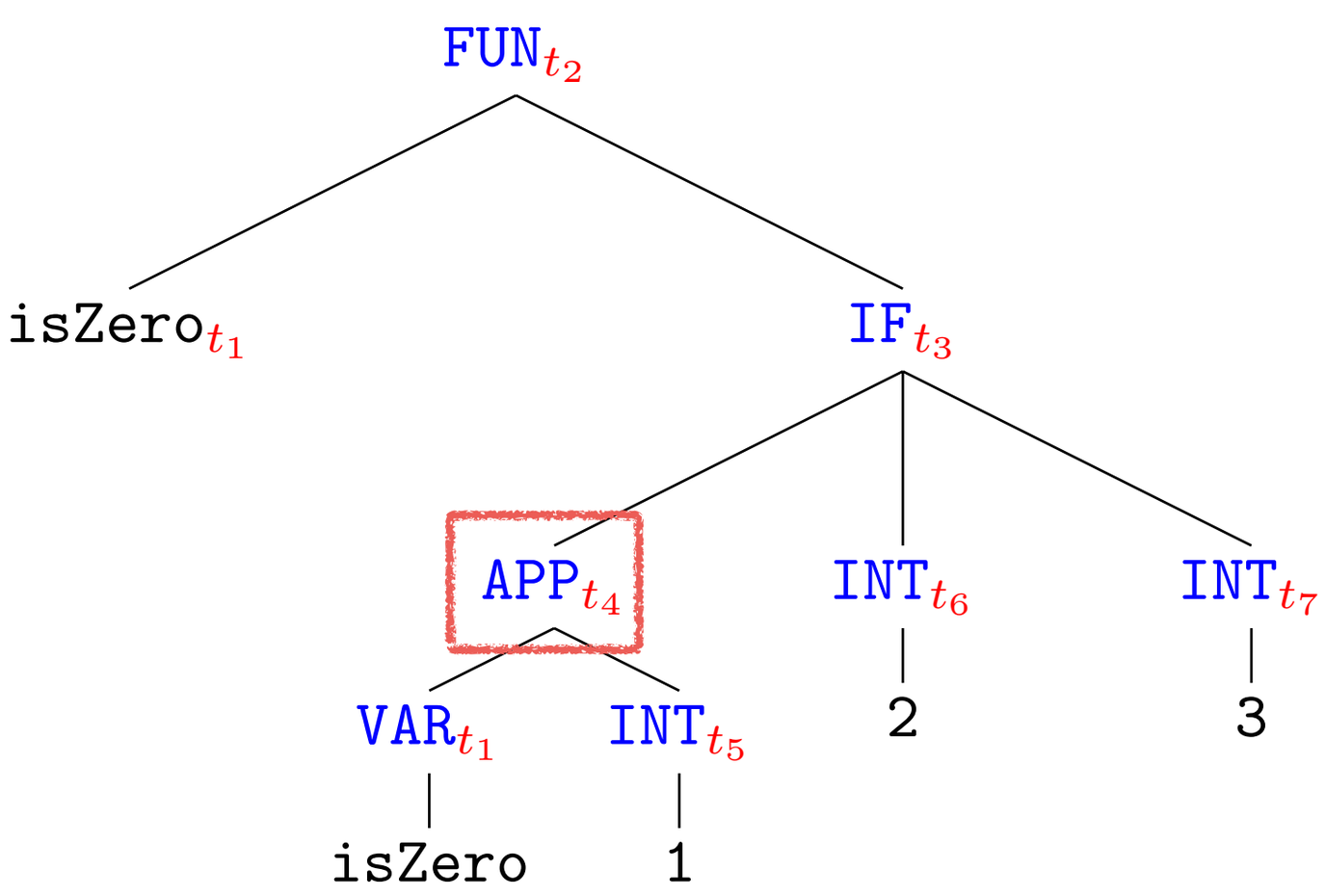


Constraint Set

```
t1 ≡ t5 → t4
t5 ≡ int
```

Wand's Algorithm Overview

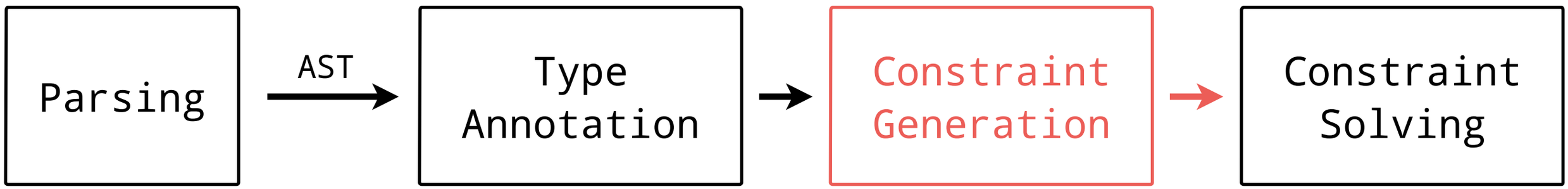


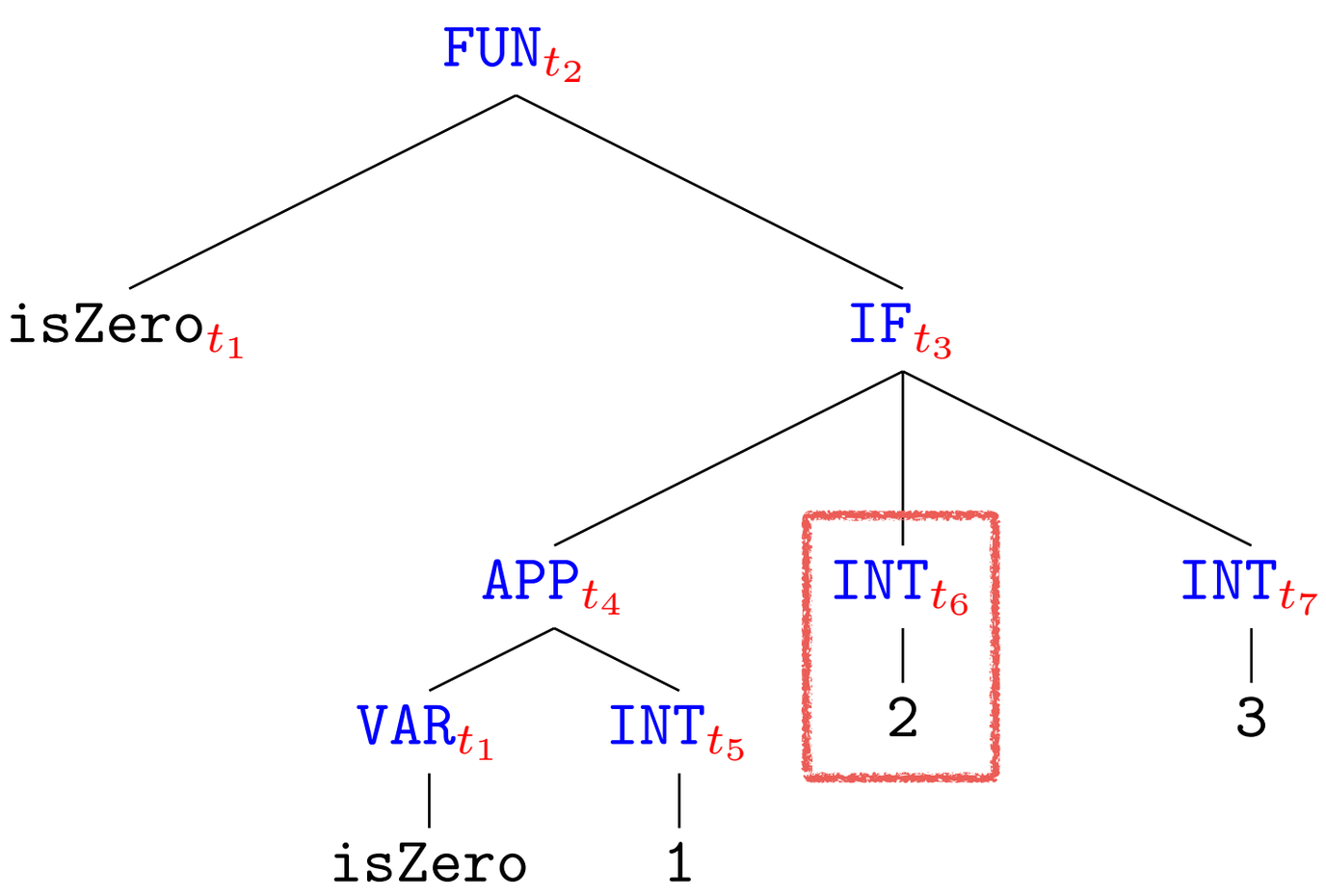


Constraint Set

```
t1 ≡ t5 → t4
t5 ≡ int
t4 ≡ bool
```

Wand's Algorithm Overview

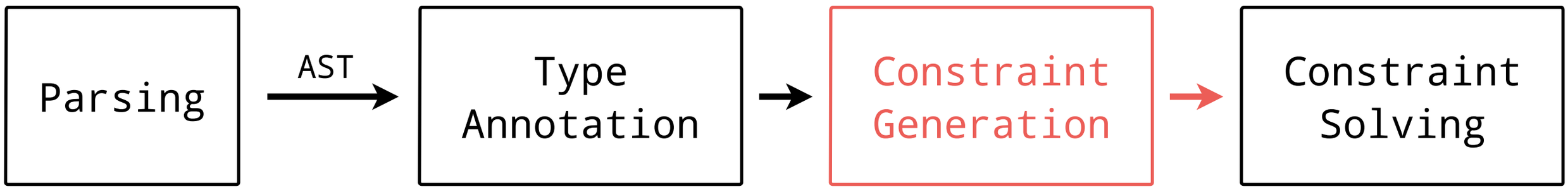


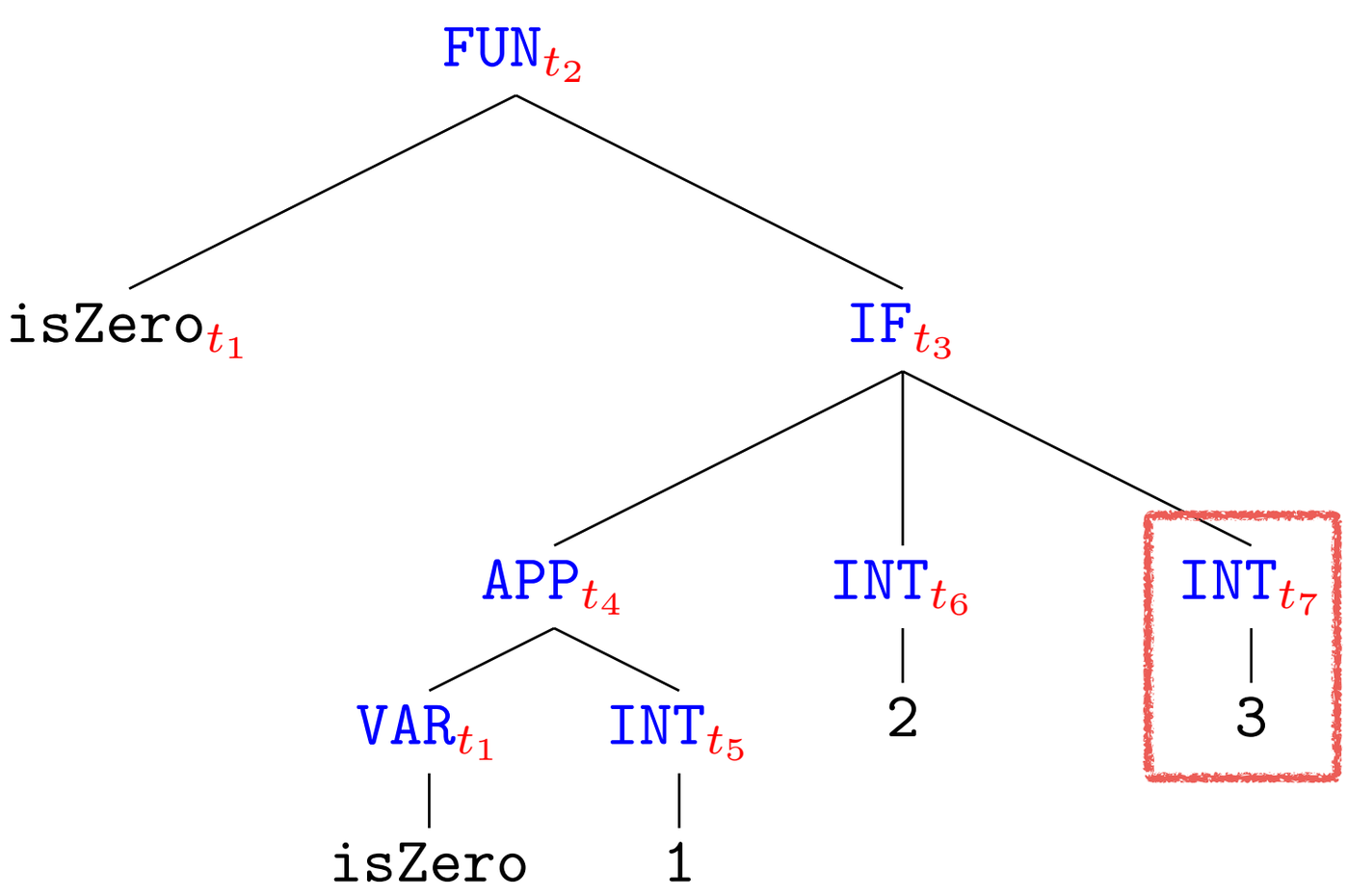


Constraint Set

t1 ≡ t5 → t4
t5 ≡ int
t4 ≡ bool
t6 ≡ int

Wand's Algorithm Overview

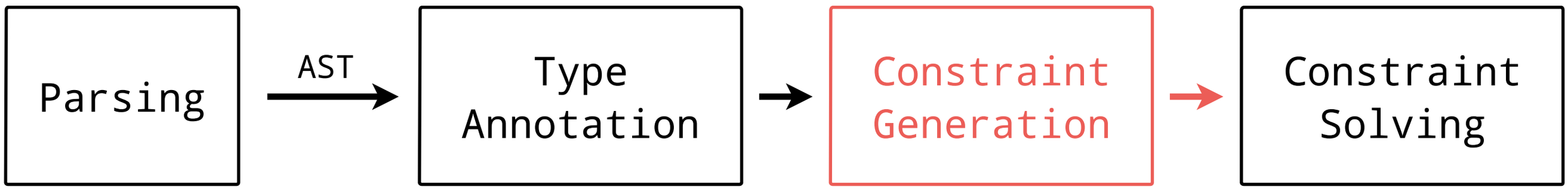


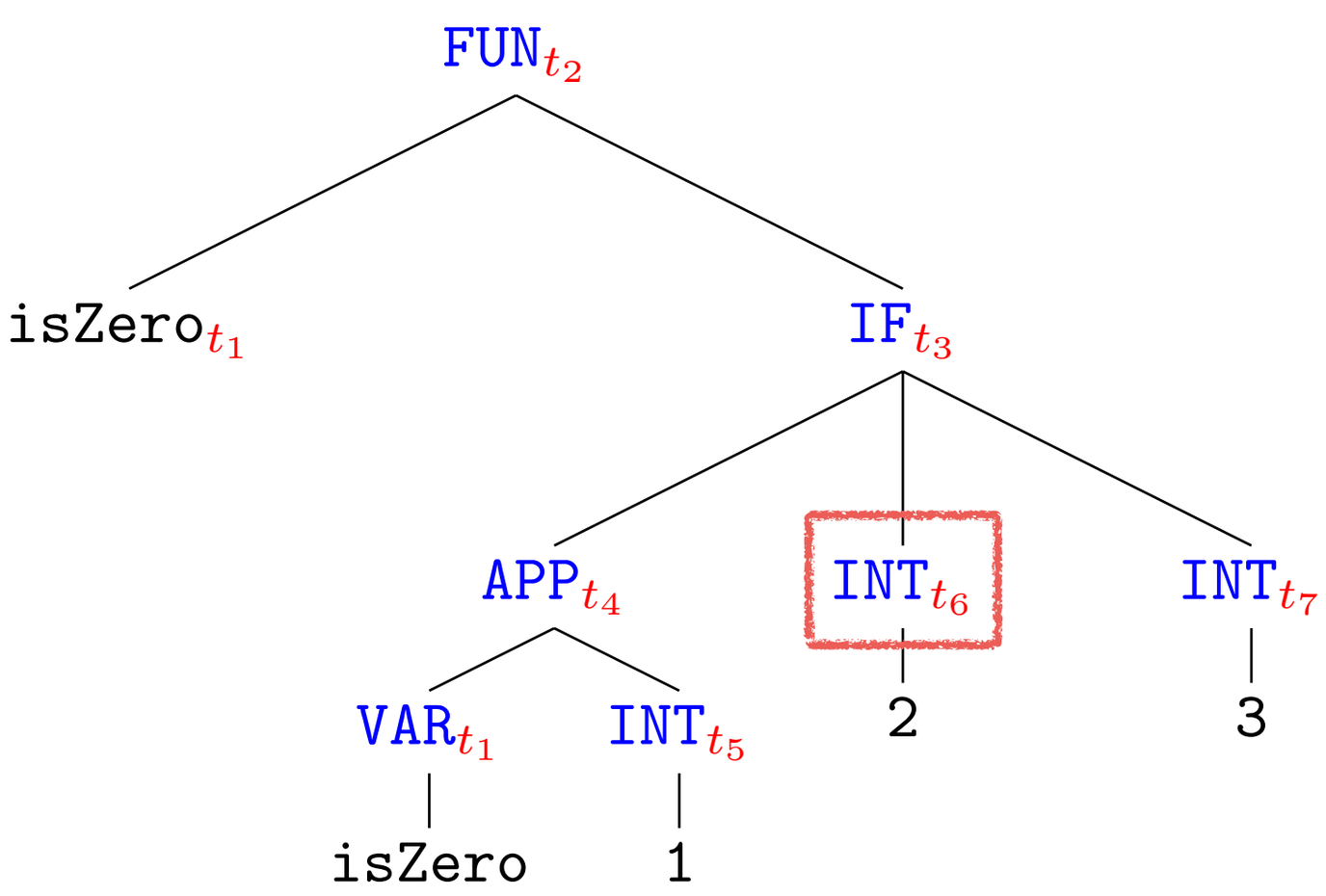


Constraint Set

```
t1 ≡ t5 → t4
t5 ≡ int
t4 ≡ bool
t6 ≡ int
t7 ≡ int
```

Wand's Algorithm Overview

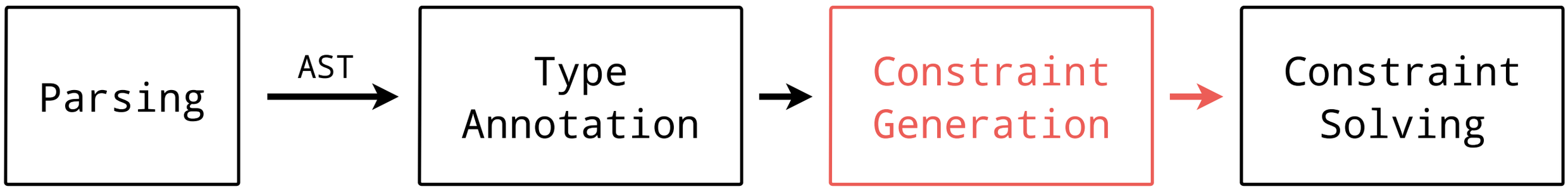


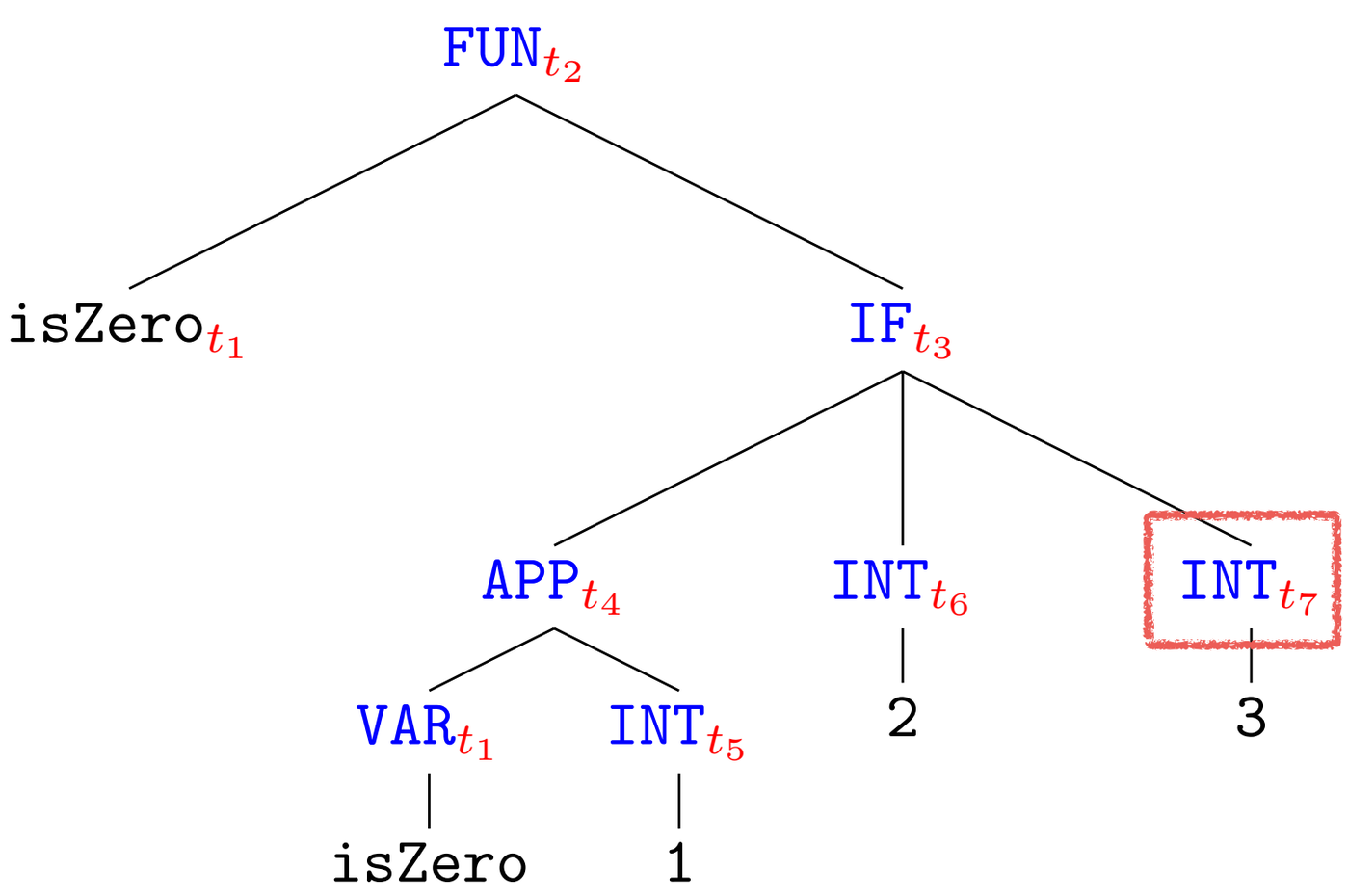


Constraint Set

```
t1 ≡ t5 → t4
t5 ≡ int
t4 ≡ bool
t6 ≡ int
t7 ≡ int
t6 ≡ t3
```

Wand's Algorithm Overview

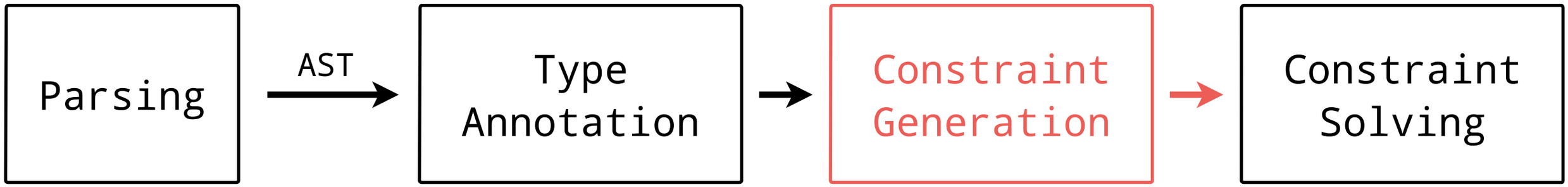


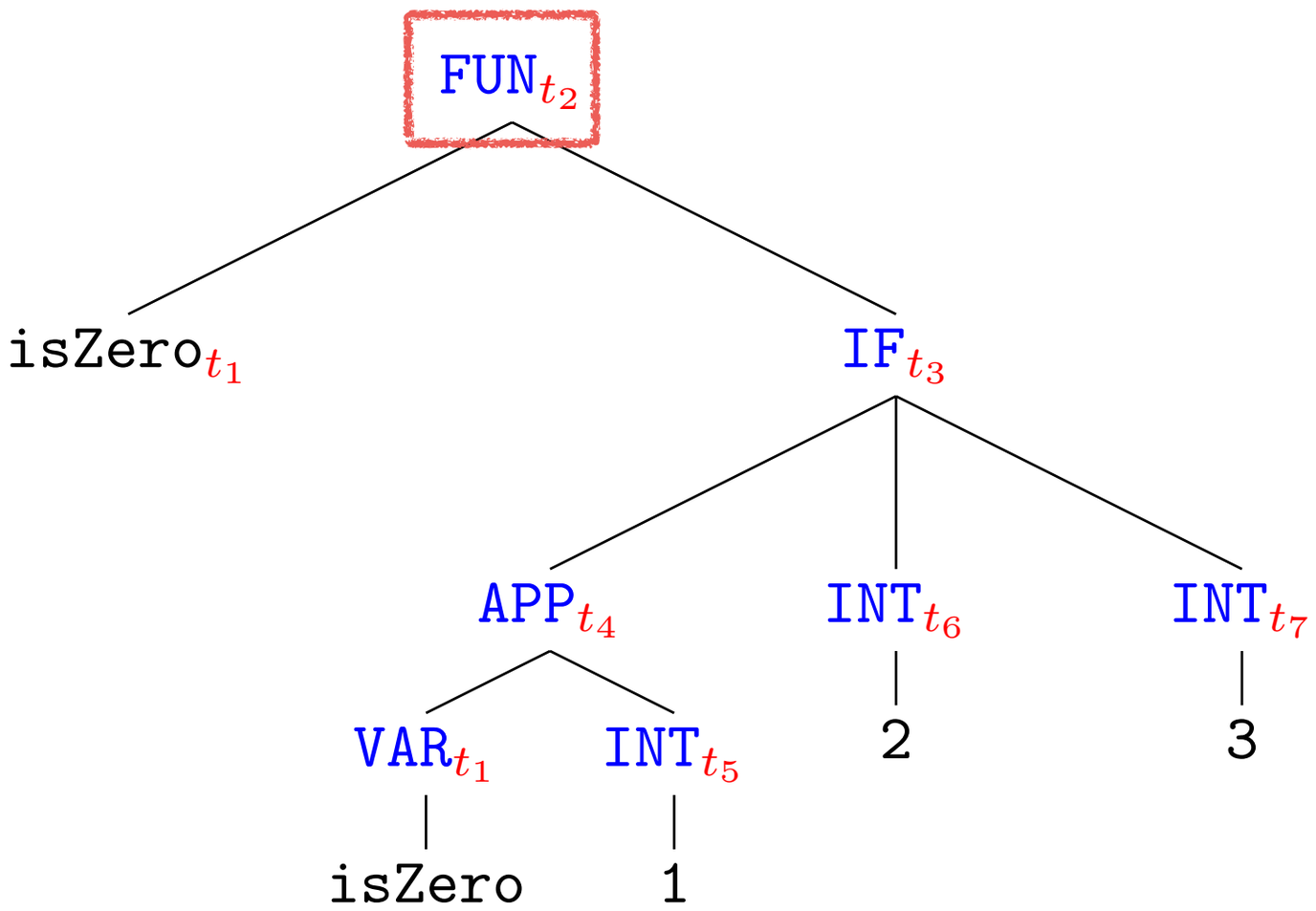


Constraint Set

```
t1 ≡ t5 → t4
t5 ≡ int
t4 ≡ bool
t6 ≡ int
t7 ≡ int
t6 ≡ t3
t7 ≡ t3
```

Wand's Algorithm Overview

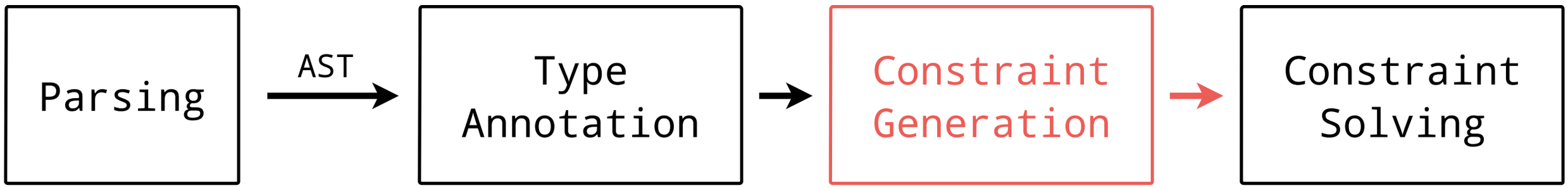


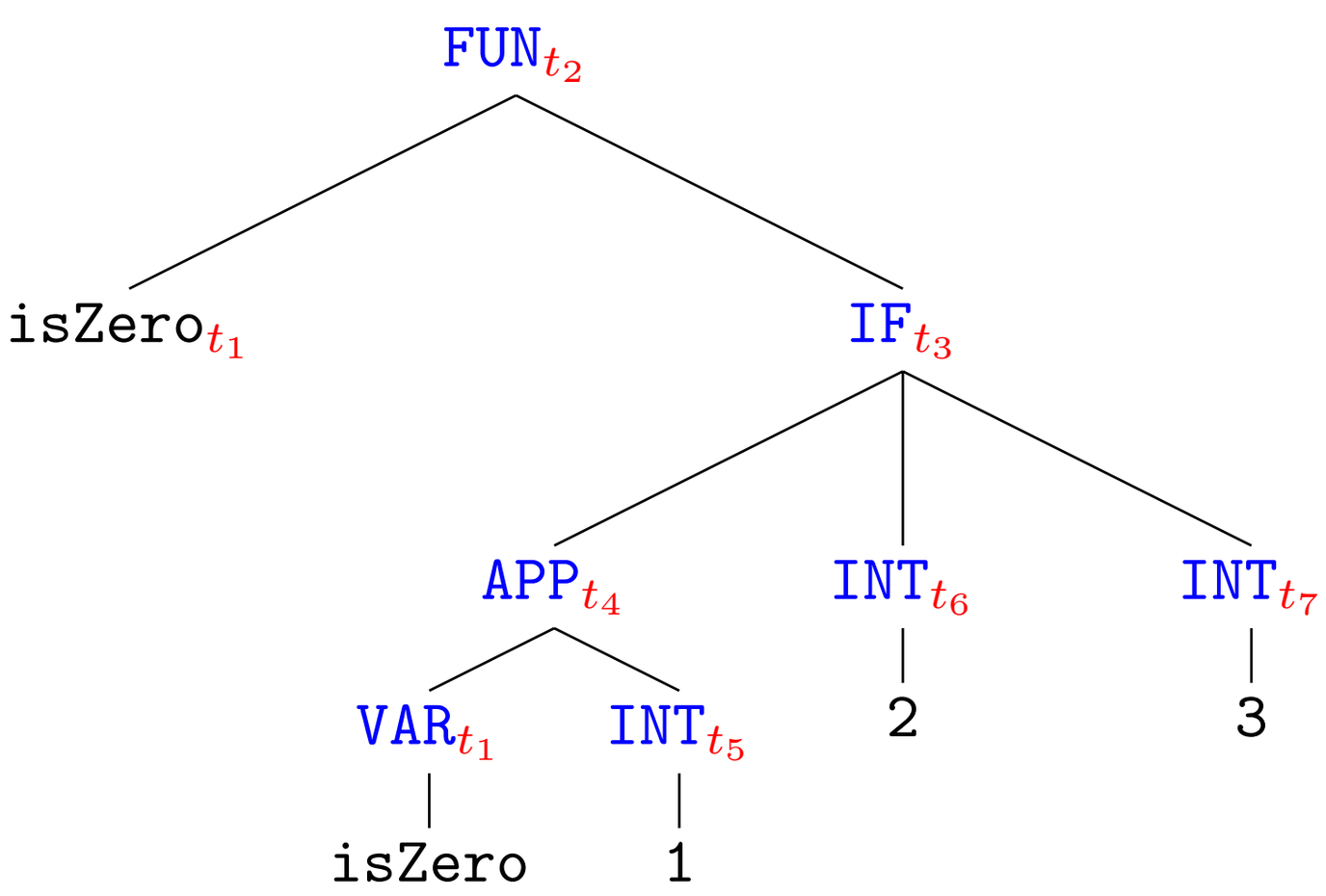


Constraint Set

```
t1 ≡ t5 → t4
t5 ≡ int
t4 ≡ bool
t6 ≡ int
t7 ≡ int
t6 ≡ t3
t7 ≡ t3
t2 ≡ t1 → t3
```

Wand's Algorithm Overview

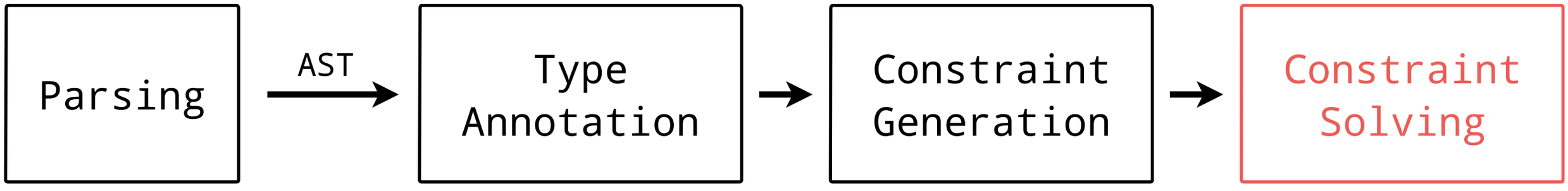




Constraint Set

```
t1 ≡ t5 → t4
t5 ≡ int
t4 ≡ bool
t6 ≡ int
t7 ≡ int
t6 ≡ t3
t7 ≡ t3
t2 ≡ t1 → t3
```

Wand's Algorithm Overview

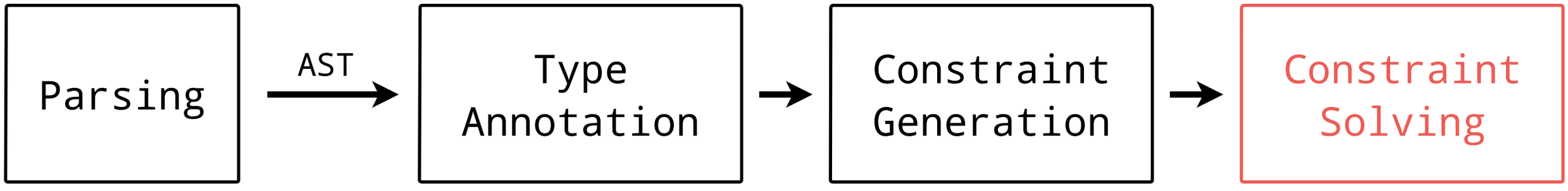


Constraint Set

```
t1 ≡ t5 → t4
t5 ≡ int
t4 ≡ bool
t6 ≡ int
t7 ≡ int
t6 ≡ t3
t7 ≡ t3
t2 ≡ t1 → t3
```

Solution Map

Wand's Algorithm Overview



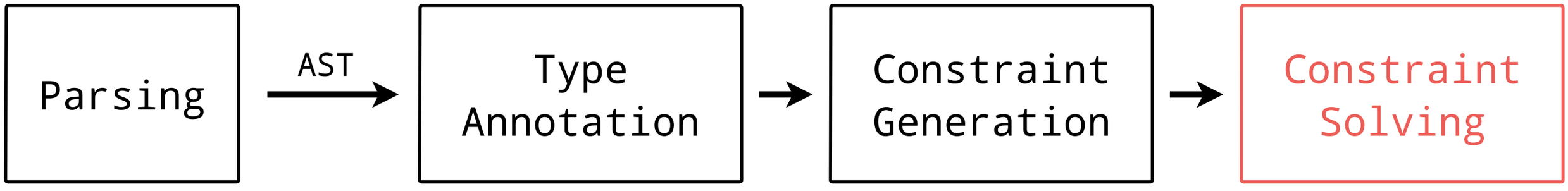
Constraint Set

```
t5 ≡ int
t4 ≡ bool
t6 ≡ int
t7 ≡ int
t6 ≡ t3
t7 ≡ t3
t2 ≡ t1 → t3
```

Solution Map

```
t1: t5 → t4
```

Wand's Algorithm Overview



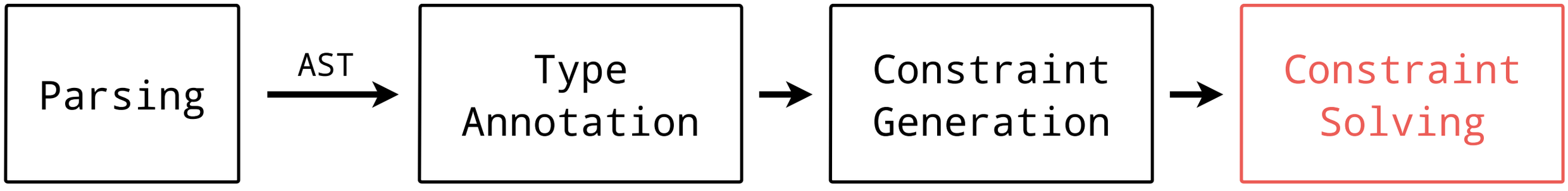
Constraint Set

```
t5 ≡ int
t4 ≡ bool
t6 ≡ int
t7 ≡ int
t6 ≡ t3
t7 ≡ t3
t2 ≡ (t5 → t4) → t3
```

Solution Map

```
t1: t5 → t4
```

Wand's Algorithm Overview



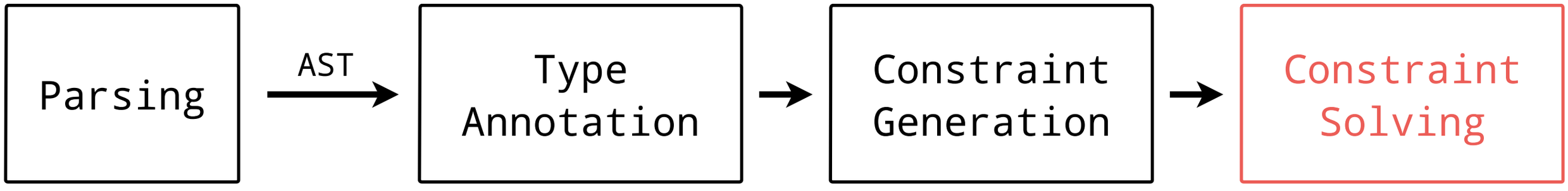
Constraint Set

```
t4 ≡ bool
t6 ≡ int
t7 ≡ int
t6 ≡ t3
t7 ≡ t3
t2 ≡ (t5 → t4) → t3
```

Solution Map

```
t1: t5 → t4
t5: int
```

Wand's Algorithm Overview



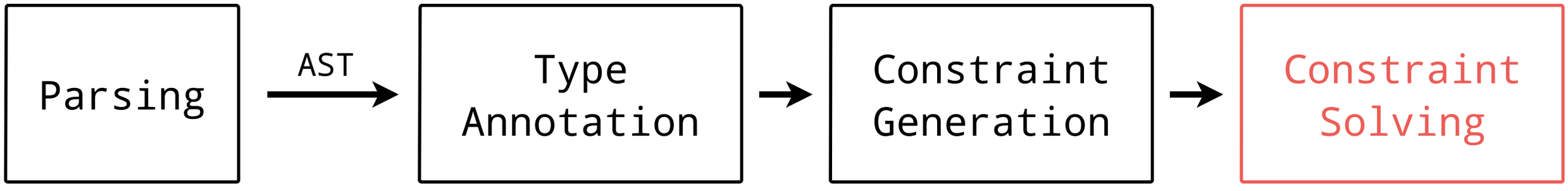
Constraint Set

```
t4 ≡ bool
t6 ≡ int
t7 ≡ int
t6 ≡ t3
t7 ≡ t3
t2 ≡ (int → t4) → t3
```

Solution Map

```
t1: int → t4
t5: int
```

Wand's Algorithm Overview



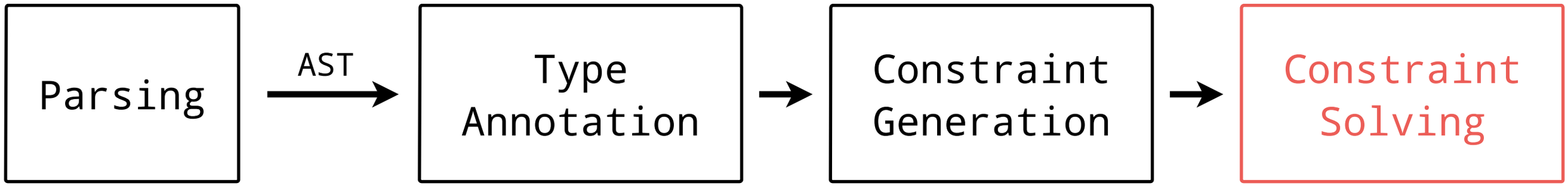
Constraint Set

t6 ≡ int
t7 ≡ int
t6 ≡ t3
t7 ≡ t3
t2 ≡ (int → t4) → t3

Solution Map

t1: int → t4
t5: int
t4: bool

Wand's Algorithm Overview



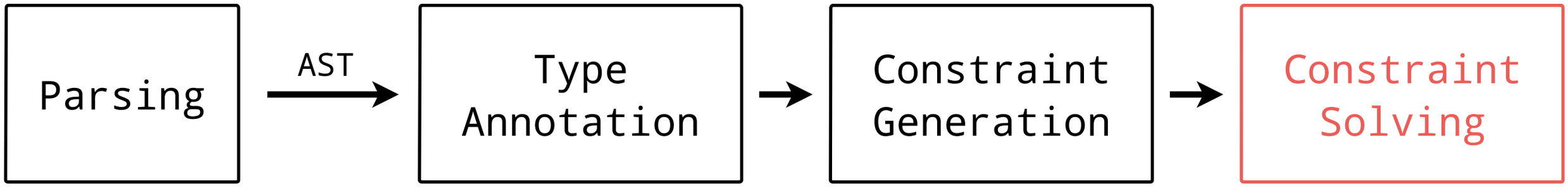
Constraint Set

```
t6 ≡ int
t7 ≡ int
t6 ≡ t3
t7 ≡ t3
t2 ≡ (int → bool) → t3
```

Solution Map

```
t1: int → bool
t5: int
t4: bool
```

Wand's Algorithm Overview



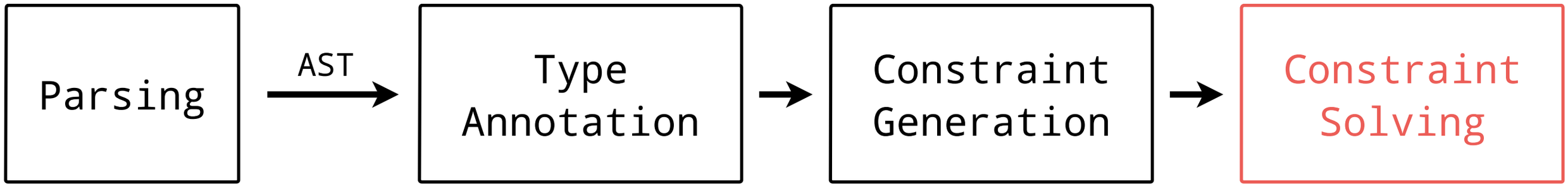
Constraint Set

t7 ≡ int
t6 ≡ t3
t7 ≡ t3
t2 ≡ (int → bool) → t3

Solution Map

t1: int → bool
t5: int
t4: bool
t6: int

Wand's Algorithm Overview



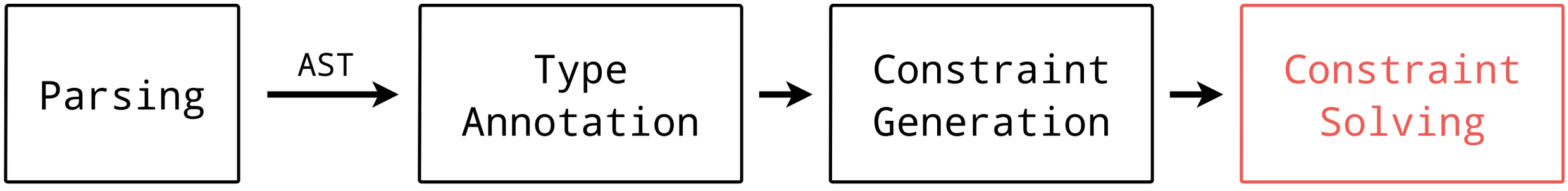
Constraint Set

```
 t7 ≡ int
int ≡ t3
 t7 ≡ t3
 t2 ≡ (int → bool) → t3
```

Solution Map

```
t1: int → bool
t5: int
t4: bool
t6: int
```

Wand's Algorithm Overview



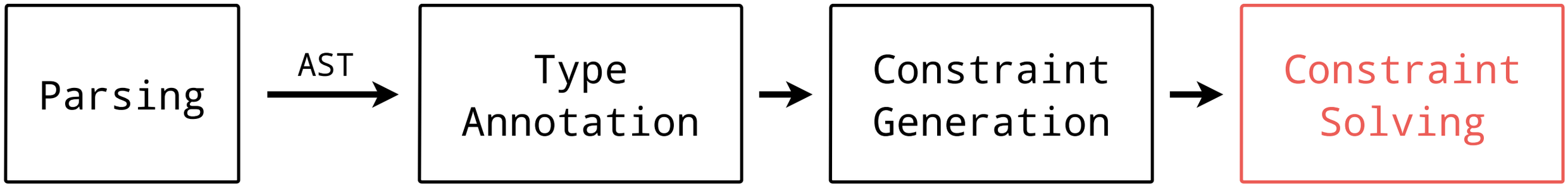
Constraint Set

```
int ≡ t3
 t7 ≡ t3
 t2 ≡ (int → bool) → t3
```

Solution Map

```
t1: int → bool
t5: int
t4: bool
t6: int
t7: int
```

Wand's Algorithm Overview



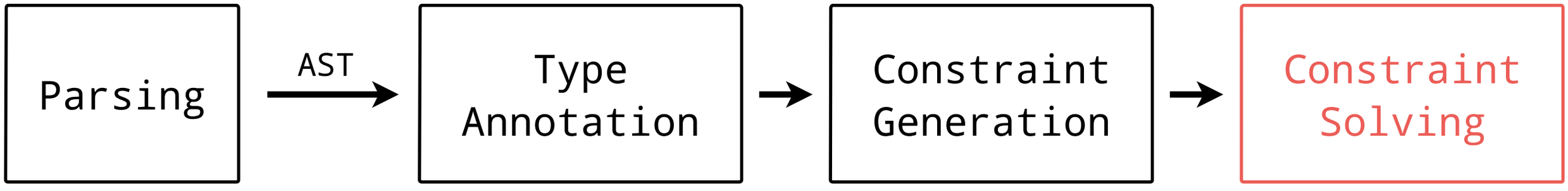
Constraint Set

```
int ≡ t3
int ≡ t3
 t2 ≡ (int → bool) → t3
```

Solution Map

```
t1: int → bool
t5: int
t4: bool
t6: int
t7: int
```

Wand's Algorithm Overview



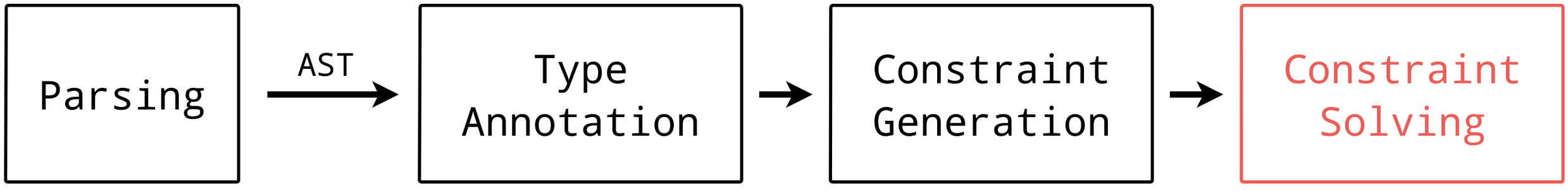
Constraint Set

```
int ≡ t3
 t2 ≡ (int → bool) → t3
```

Solution Map

```
t1: int → bool
t5: int
t4: bool
t6: int
t7: int
t3: int
```

Wand's Algorithm Overview



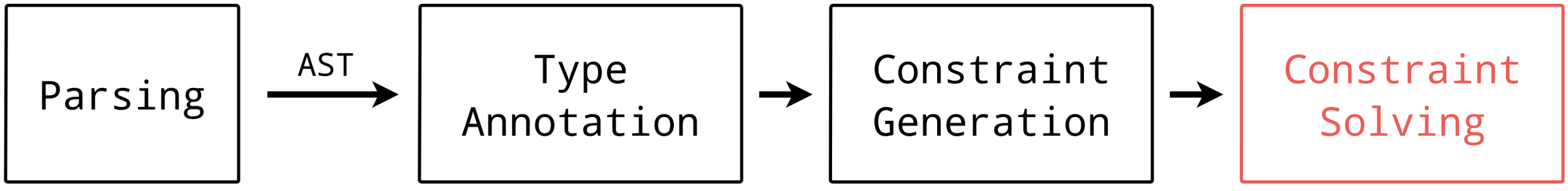
Constraint Set

```
int ≡ int
 t2 ≡ (int → bool) → int
```

Solution Map

```
t1: int → bool
t5: int
t4: bool
t6: int
t7: int
t3: int
```

Wand's Algorithm Overview



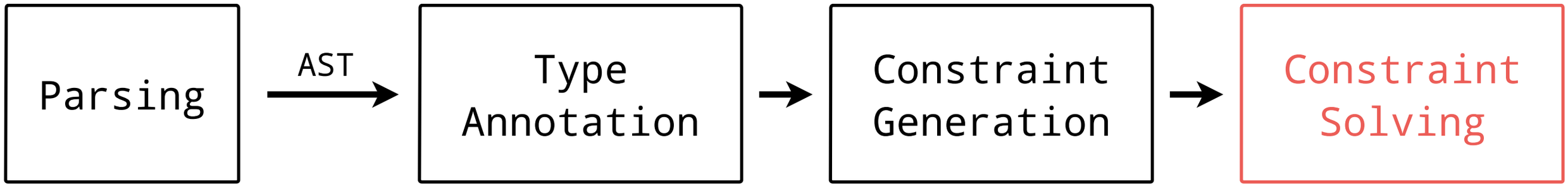
Constraint Set

```
int ≡ int
 t2 ≡ (int → bool) → int
```

Solution Map

```
t1: int → bool
t5: int
t4: bool
t6: int
t7: int
t3: int
```

Wand's Algorithm Overview



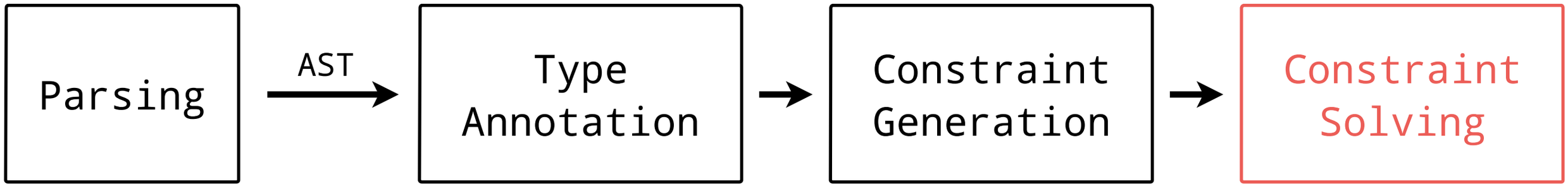
Constraint Set

t2 ≡ (int → bool) → int

Solution Map

t1: int → bool
t5: int
t4: bool
t6: int
t7: int
t3: int

Wand's Algorithm Overview

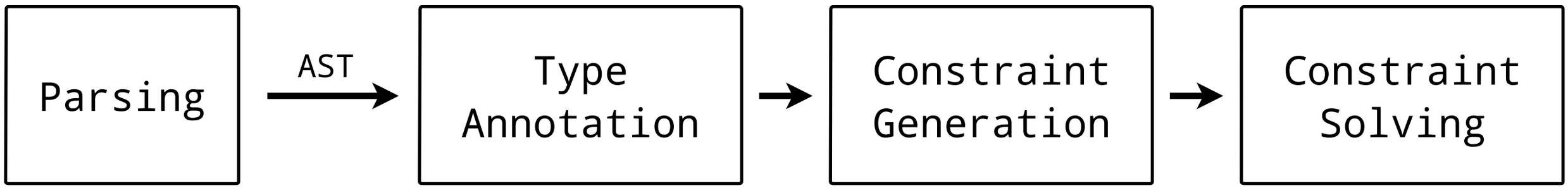


Constraint Set

Solution Map

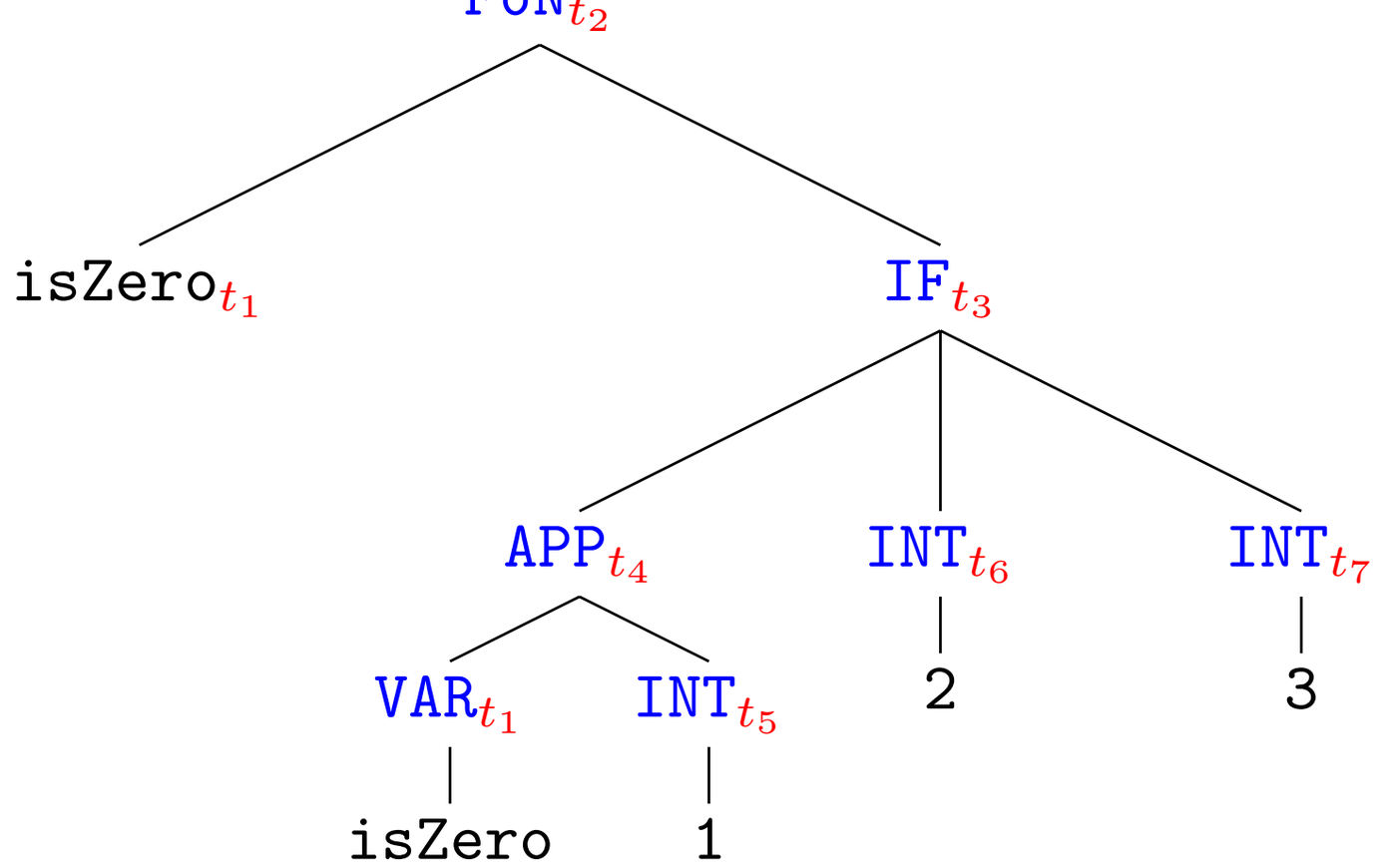
```
t1: int → bool
t5: int
t4: bool
t6: int
t7: int
t3: int
t2: (int → bool) → int
```

Wand's Algorithm Overview

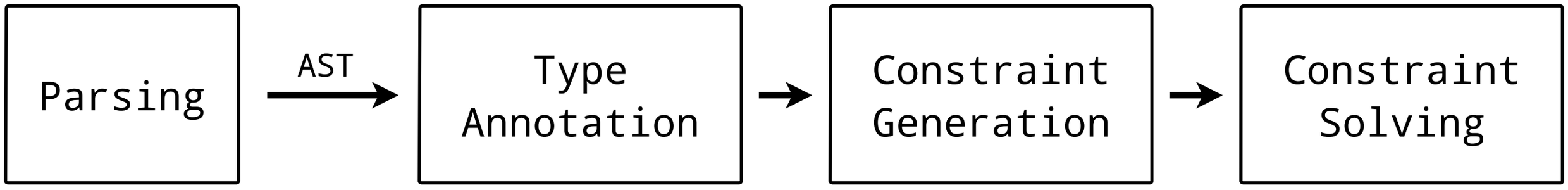


Solution Map

t1: int → bool
t5: int
t4: bool
t6: int
t7: int
t3: int
t2: (int → bool) → int



Wand's Algorithm Overview




```
fn isZero =>
  if isZero 1
  then 2
  else 3
```

➔ `(int -> bool) -> int`

Type Checking Algorithms

- As I said, there are two main classes
- Constraint-based ones. We've just seen one example — Wand's algorithm
- Substitution-based ones, where constraint generation and solving are not two separate processes, they are interleaved. Example: the classic Hindley-Milner Algorithm W.

Live Demonstration

https://github.com/igstan/itake-2015

Thank You!

Resources

- **Sunil Kothari and James L. Caldwell**. Type Reconstruction Algorithms - A Survey
- **Mitchell Wand**. A simple algorithm and proof for type inference.
- **Bastiaan Heeren, Jurriaan Hage and Doaitse Swierstra**. Generalizing Hindley-Milner Type Inference Algorithms
- **Oleg Kiselyov and Chung-chieh Shan**. Interpreting Types as Abstract Values
- **Shriram Krishnamurthi**. Programming Languages: Application and Interpretation, chapter 15
- **Shriram Krishnamurthi**. Programming Languages: Application and Interpretation, lecture 24
- **Shriram Krishnamurthi**. Programming Languages: Application and Interpretation, lecture 25
- **Bastiaan Heeren**. Top Quality Type Error Messages
- **Stephen Diehl**. Write You a Haskell, chapter 6
- **Andrew Appel**. Modern Compiler Implementation in ML, chapter 16
- **Benjamin Pierce**. Types and Programming Languages, chapter 22
- **Martin Odersky**. Scala by Example, chapter 16
- **Danny Gratzer**. https://github.com/jozefg/hm
- **Arlen Cox**. ML Type Inference and Unification
- **Radu Rugină.** CS 312, Type Inference