A Type Inferencer for ML

In 200 Lines of Scala

Ionuț G. Stan — igstan.ro — ionut@eloquentix.com

About Me

- Software Developer at eloquentix
- Worked with Scala for the past 5 years
- FP, programming languages, compilers
- Mostly-tech blog at igstan.ro

Plan

Plan

- Compilers Overview

Plan

- Compilers Overview
- Vehicle Language: μML

Plan

- Compilers Overview
- Vehicle Language: μML
- Wand's Type Inference Algorithm

Plan

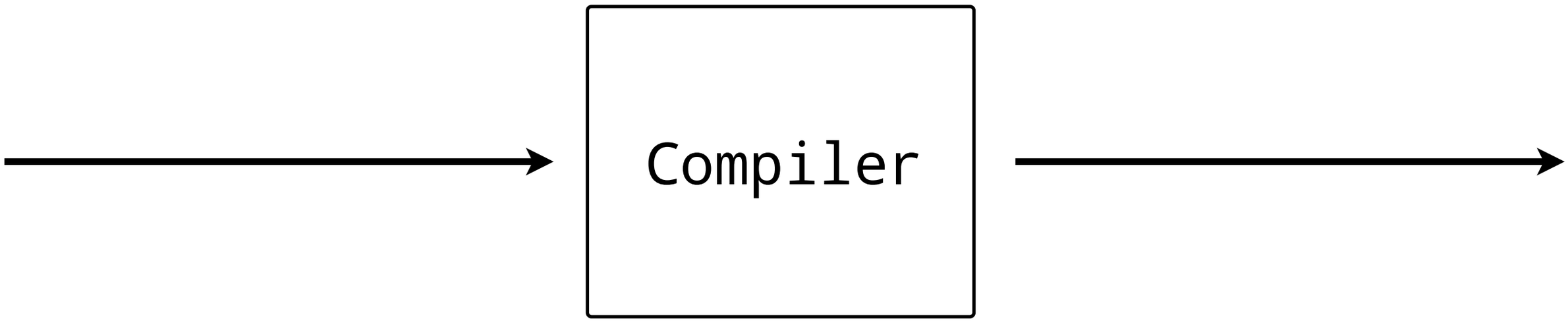
- Compilers Overview
- Vehicle Language: μML
- Wand's Type Inference Algorithm
  - Intuition

Plan

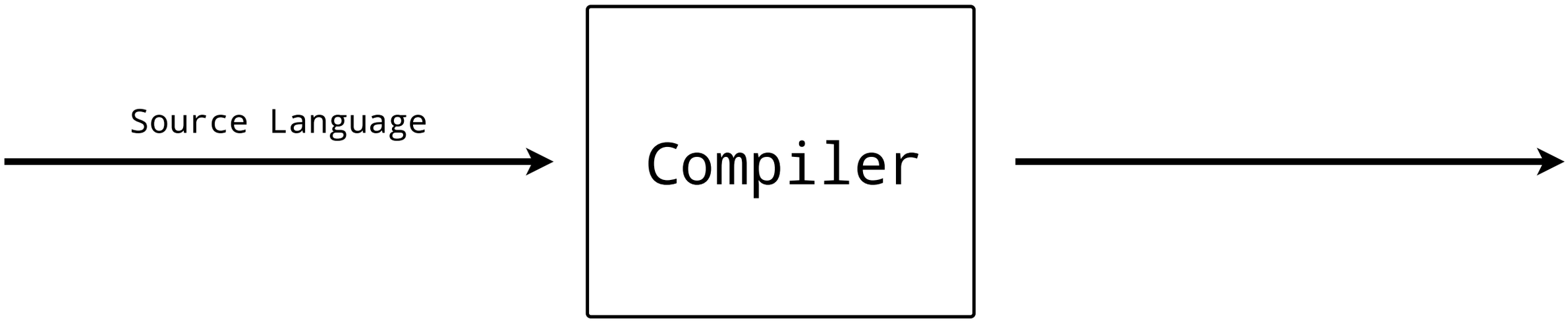
- Compilers Overview
- Vehicle Language: μML
- Wand's Type Inference Algorithm
  - Intuition
  - Code

Compilers Overview

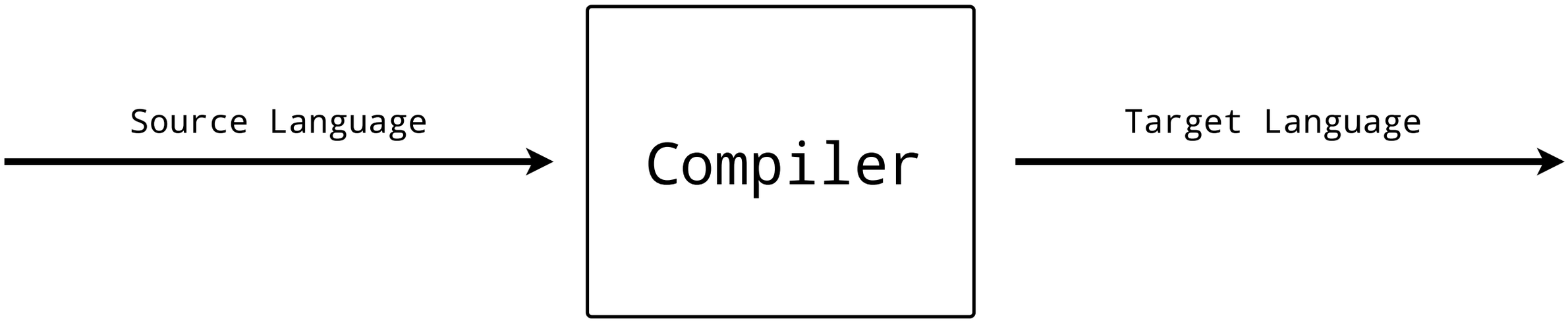
Compilers Overview



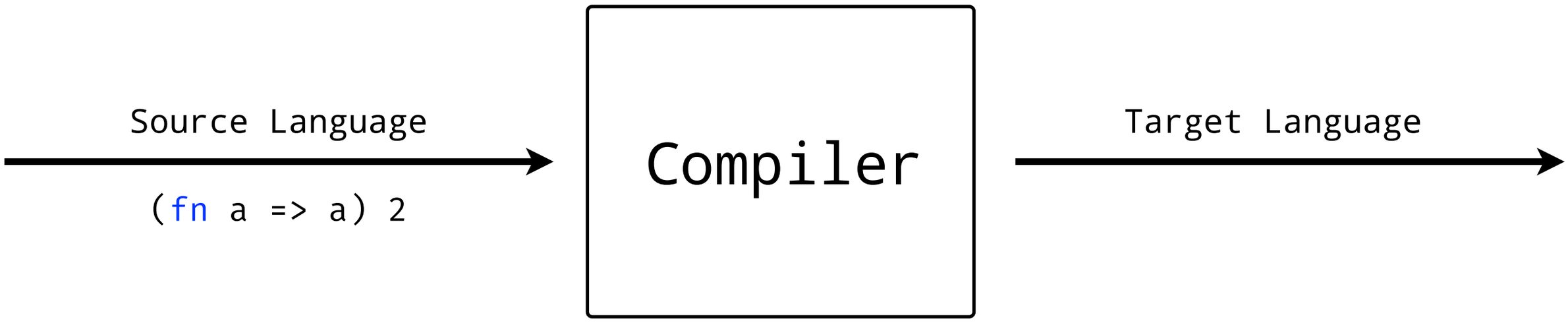
Compilers Overview



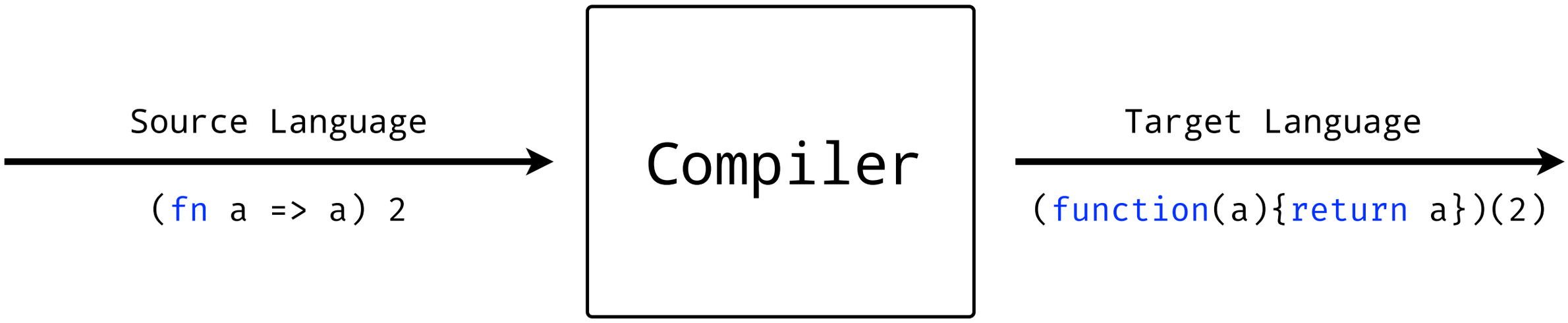
Compilers Overview



Compilers Overview

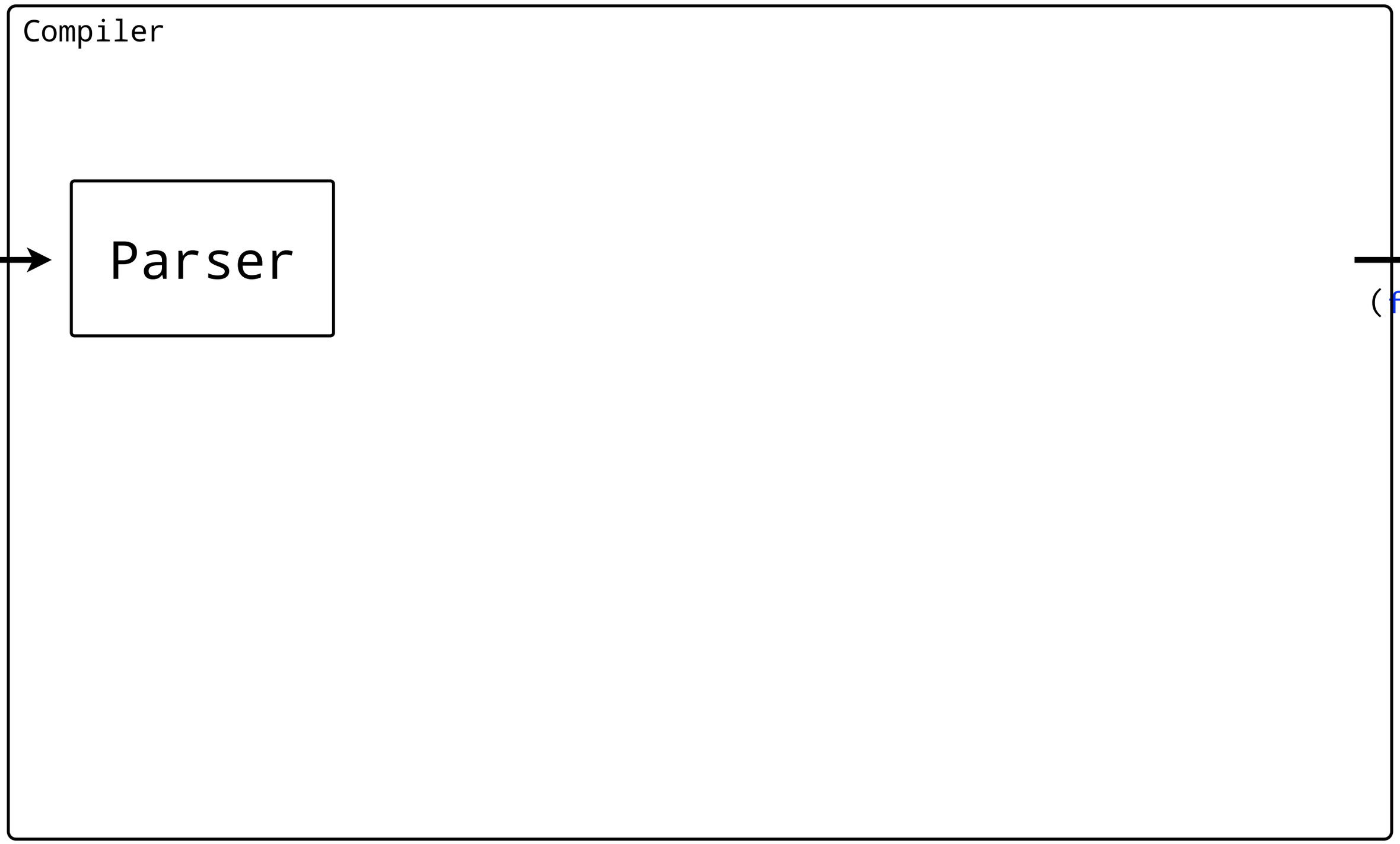


Compilers Overview

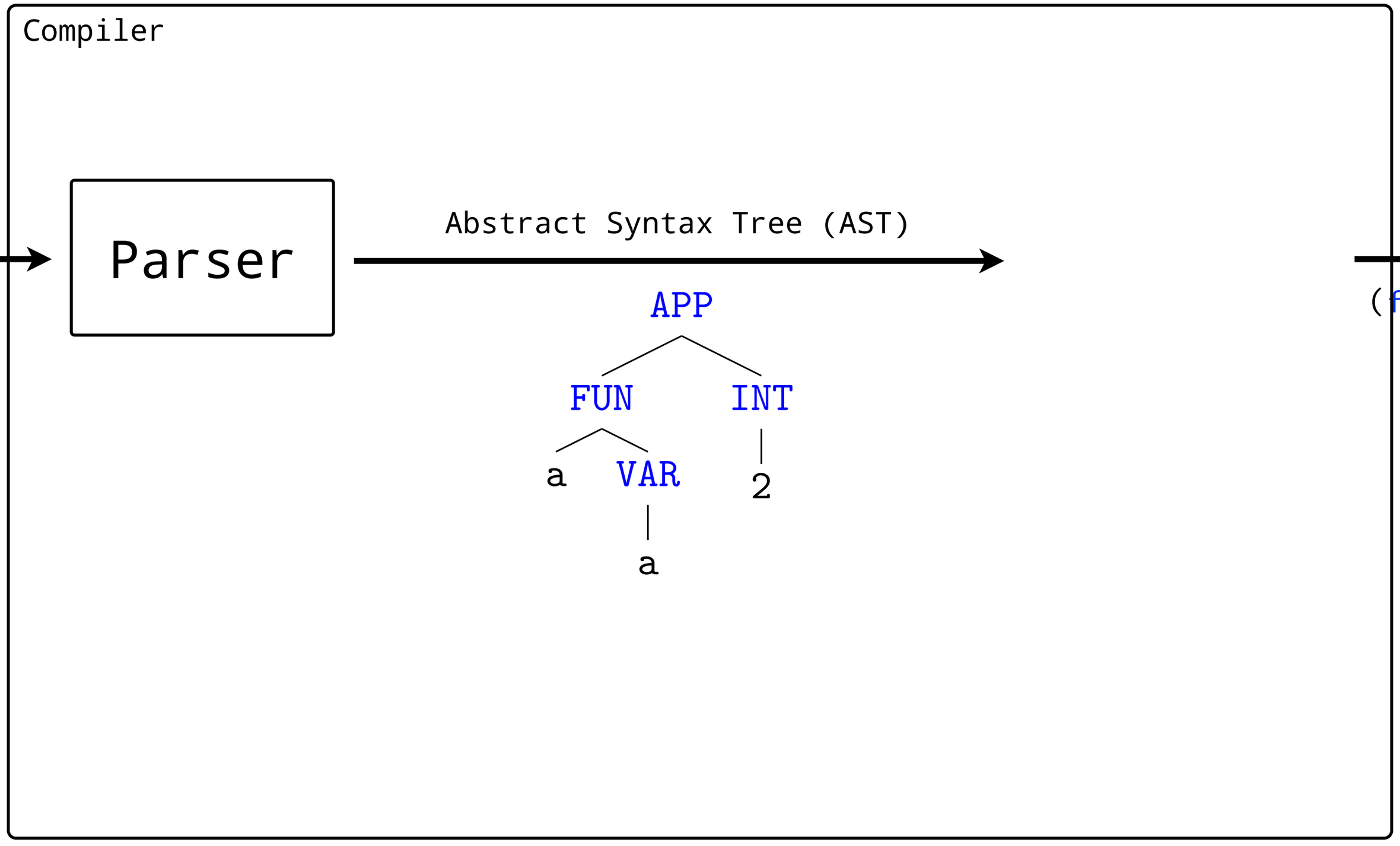


Compilers Overview

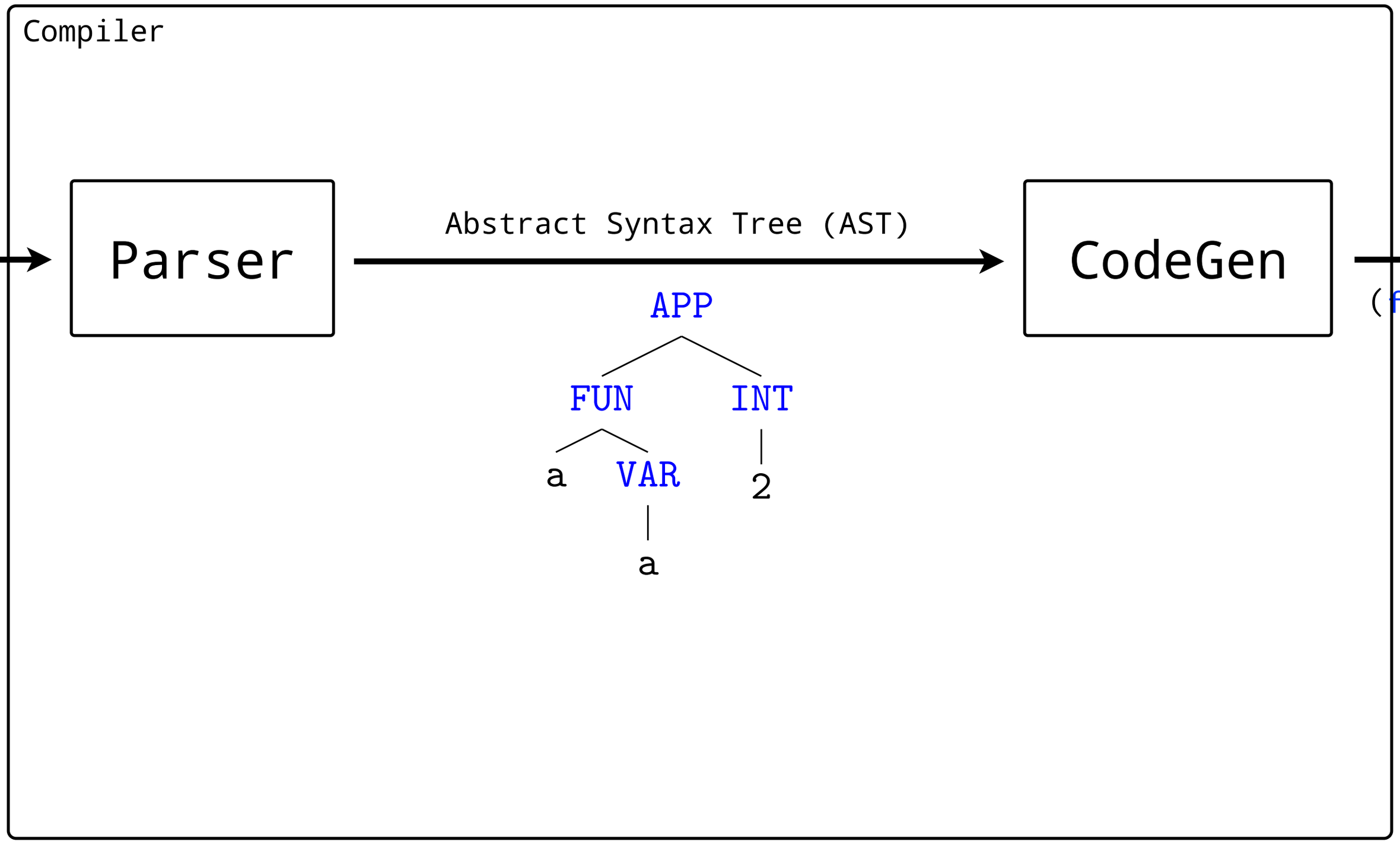
Parsing



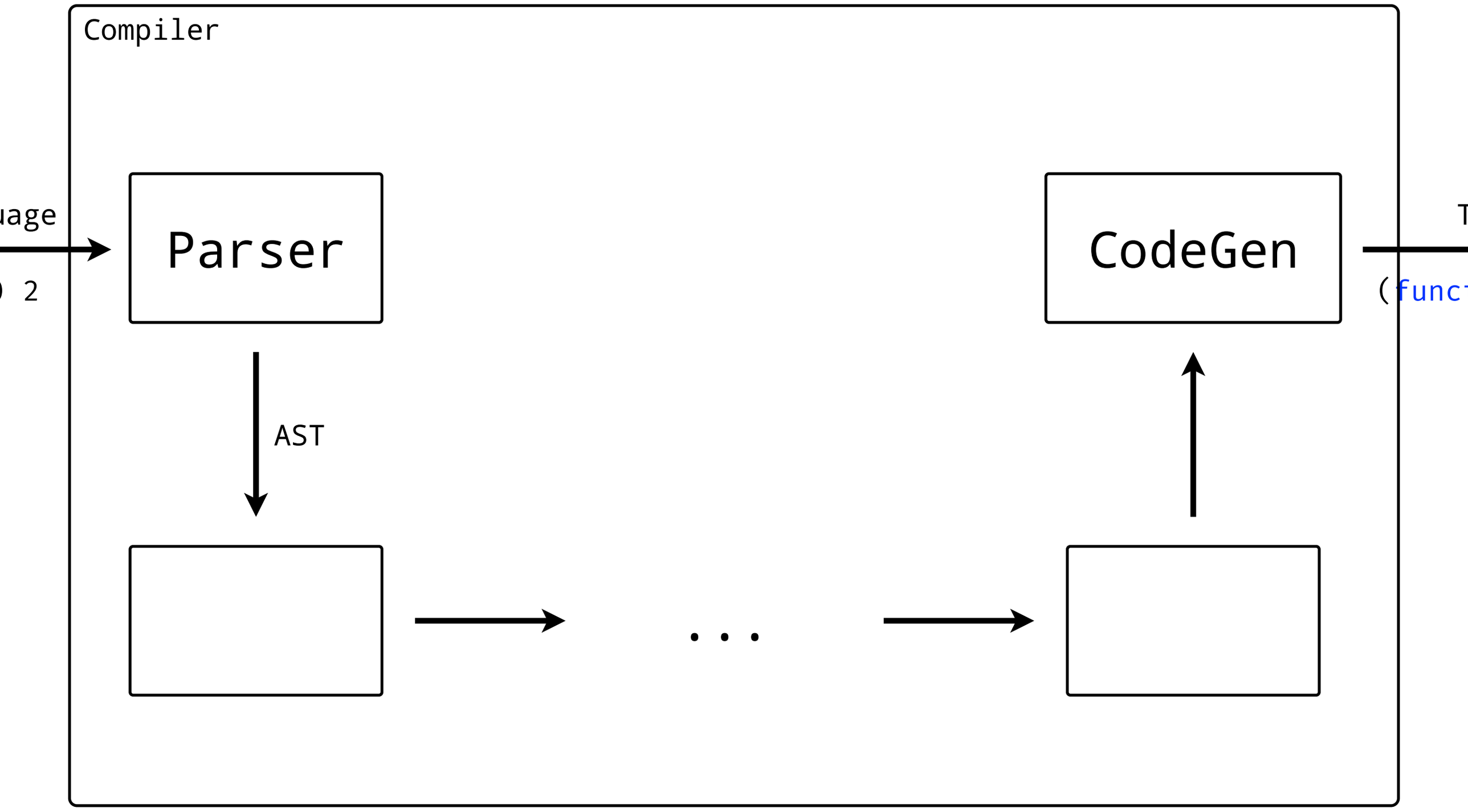
Abstract Syntax Tree



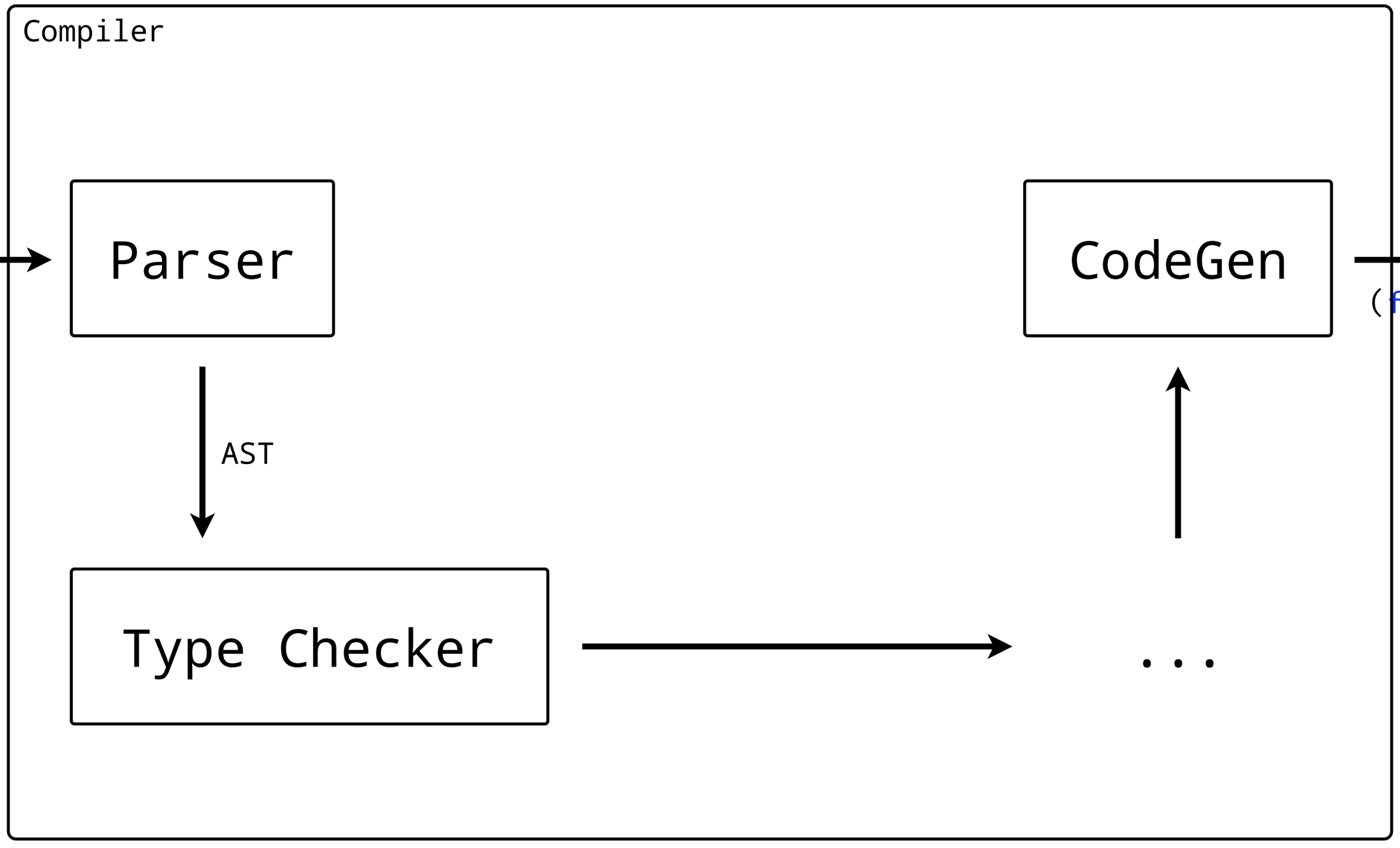
Code Generation



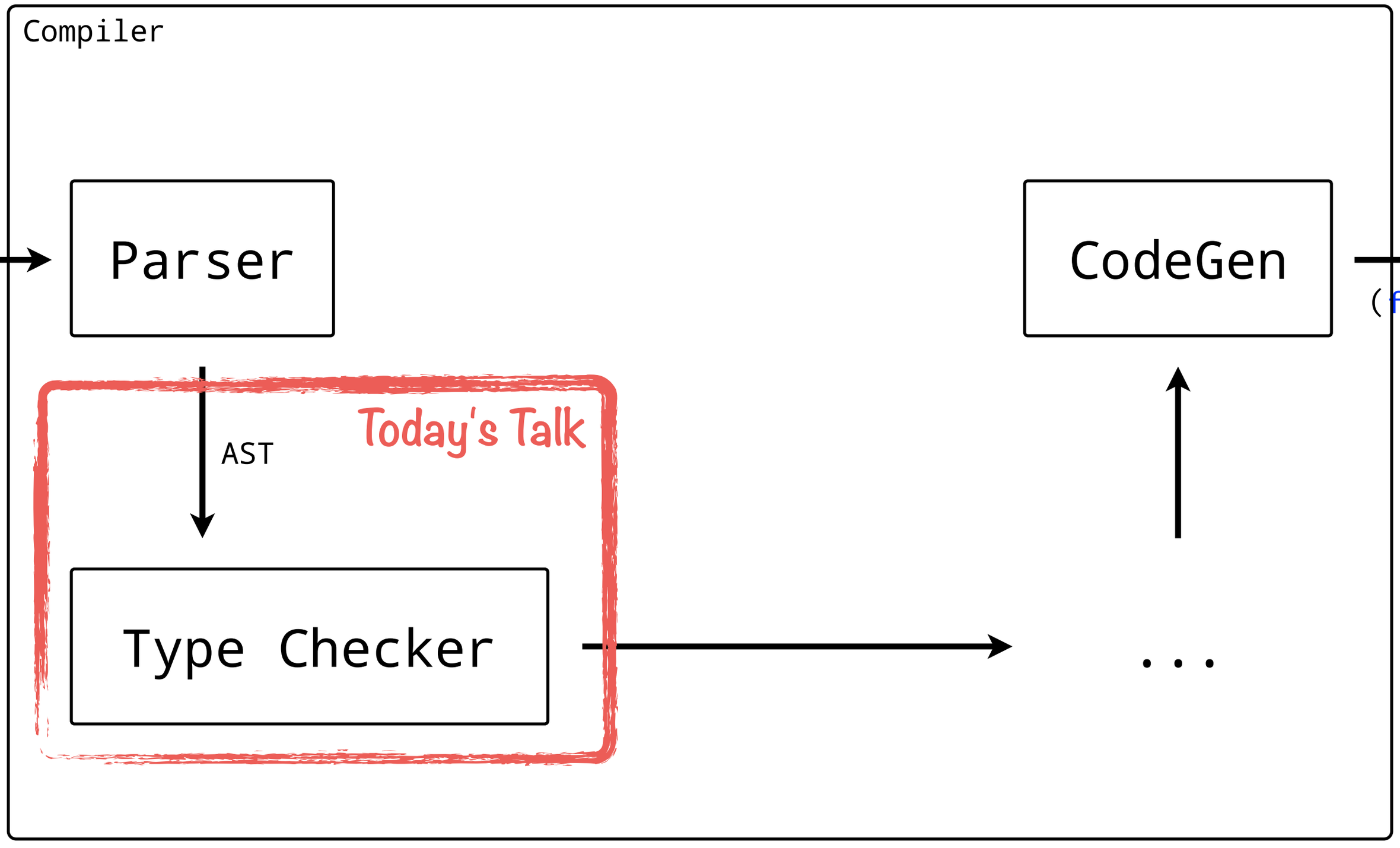
Many Intermediate Phases



Type Checking



Today's Talk


Compiler
uage
) 2
Parser
CodeGen
T
(funct
AST
Today's Talk
Type Checker
. . .

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: μML

Vehicle Language: μML

- Surface Syntax
  - What does the language look like
- Type System
  - What types the language supports

μML — Surface Syntax

1. Integers: `1`, `23`, `456`, etc.

μML — Surface Syntax

1. Integers: `1`, `23`, `456`, etc.
2. Identifiers (only letters): `inc`, `cond`, `a`, etc.

μML — Surface Syntax

1. Integers: `1`, `23`, `456`, etc.
2. Identifiers (only letters): `inc`, `cond`, `a`, etc.
3. Booleans: `true` and `false`

μML — Surface Syntax

1. Integers: `1`, `23`, `456`, etc.
2. Identifiers (only letters): `inc`, `cond`, `a`, etc.
3. Booleans: `true` and `false`
4. Single-argument anonymous functions: `fn a => a`

μML — Surface Syntax

1. Integers: `1`, `23`, `456`, etc.
2. Identifiers (only letters): `inc`, `cond`, `a`, etc.
3. Booleans: `true` and `false`
4. Single-argument anonymous functions: `fn a => a`
5. Function application: `inc 42`

μML — Surface Syntax

1. Integers: `1`, `23`, `456`, etc.
2. Identifiers (only letters): `inc`, `cond`, `a`, etc.
3. Booleans: `true` and `false`
4. Single-argument anonymous functions: `fn a => a`
5. Function application: `inc 42`
6. If expressions: `if cond then t else f`

μML — Surface Syntax

1. Integers: `1`, `23`, `456`, etc.
2. Identifiers (only letters): `inc`, `cond`, `a`, etc.
3. Booleans: `true` and `false`
4. Single-argument anonymous functions: `fn a => a`
5. Function application: `inc 42`
6. If expressions: `if cond then t else f`
7. Addition and subtraction: `a + b`, `a - b`

μML — Surface Syntax

1. Integers: `1`, `23`, `456`, etc.
2. Identifiers (only letters): `inc`, `cond`, `a`, etc.
3. Booleans: `true` and `false`
4. Single-argument anonymous functions: `fn a => a`
5. Function application: `inc 42`
6. If expressions: `if cond then t else f`
7. Addition and subtraction: `a + b`, `a - b`
8. Parenthesized expressions: `(a + b)`

μML — Surface Syntax

9. Let blocks/expressions:

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

Small Example

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

μML — Language Types

μML — Language Types

1. Integer type: `int`

μML — Language Types

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

μML — Language Types

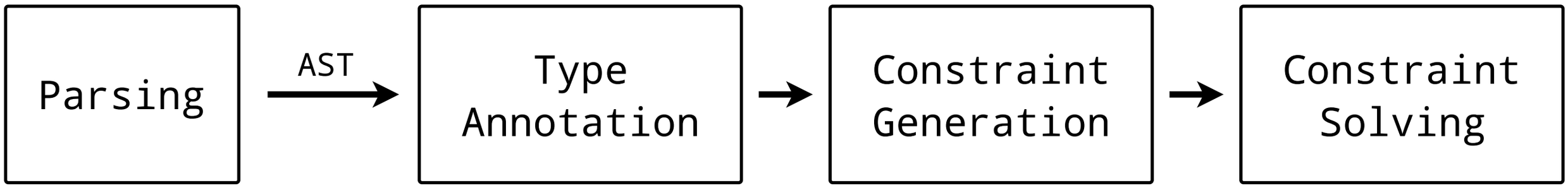
1. Integer type: `int`
2. Boolean type: `bool`
3. Function type: `int -> bool`

Today's Algorithm Overview

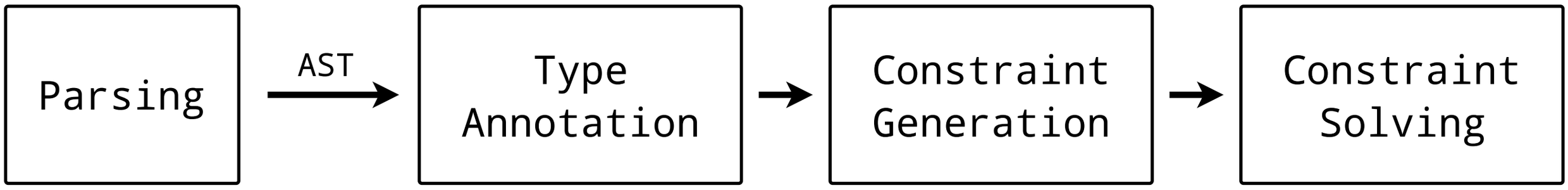
Wand's Algorithm

Overview

Wand's Algorithm Overview

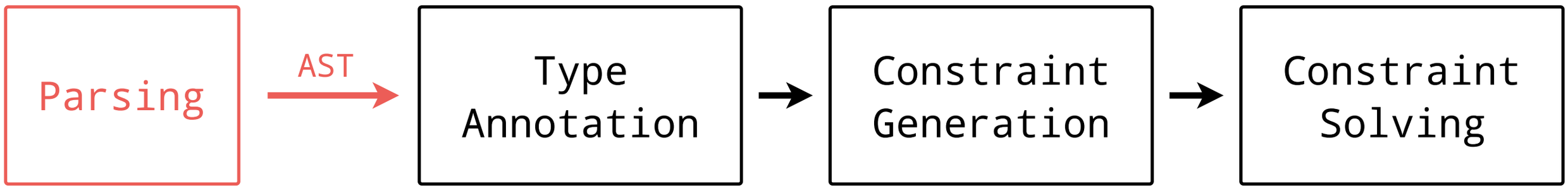


Wand's Algorithm Overview

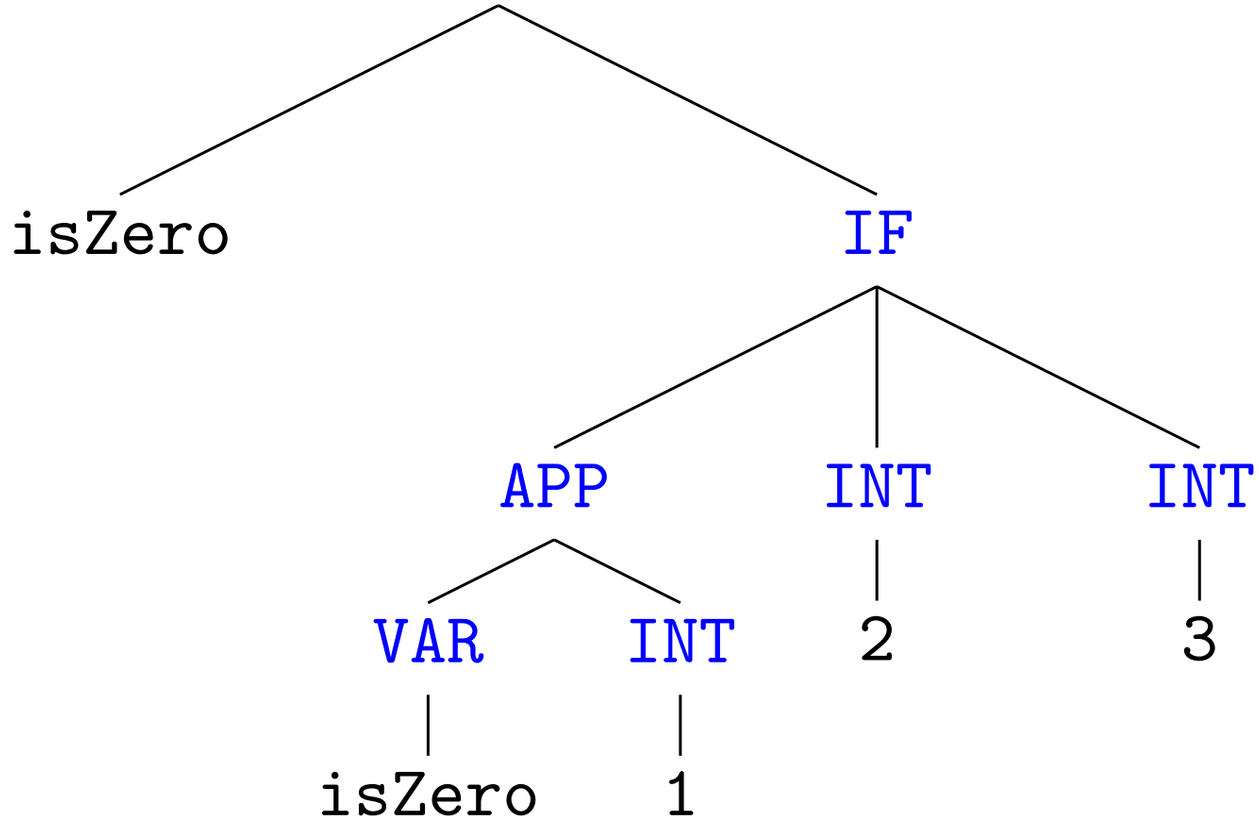



```
fn isZero =>
   if isZero 1
   then 2
   else 3
```

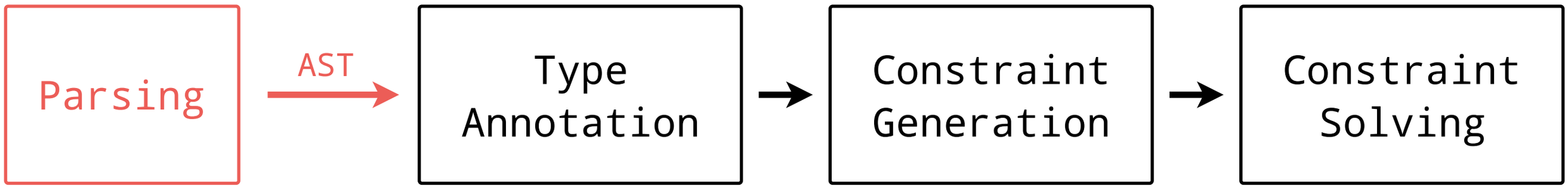
Wand's Algorithm Overview



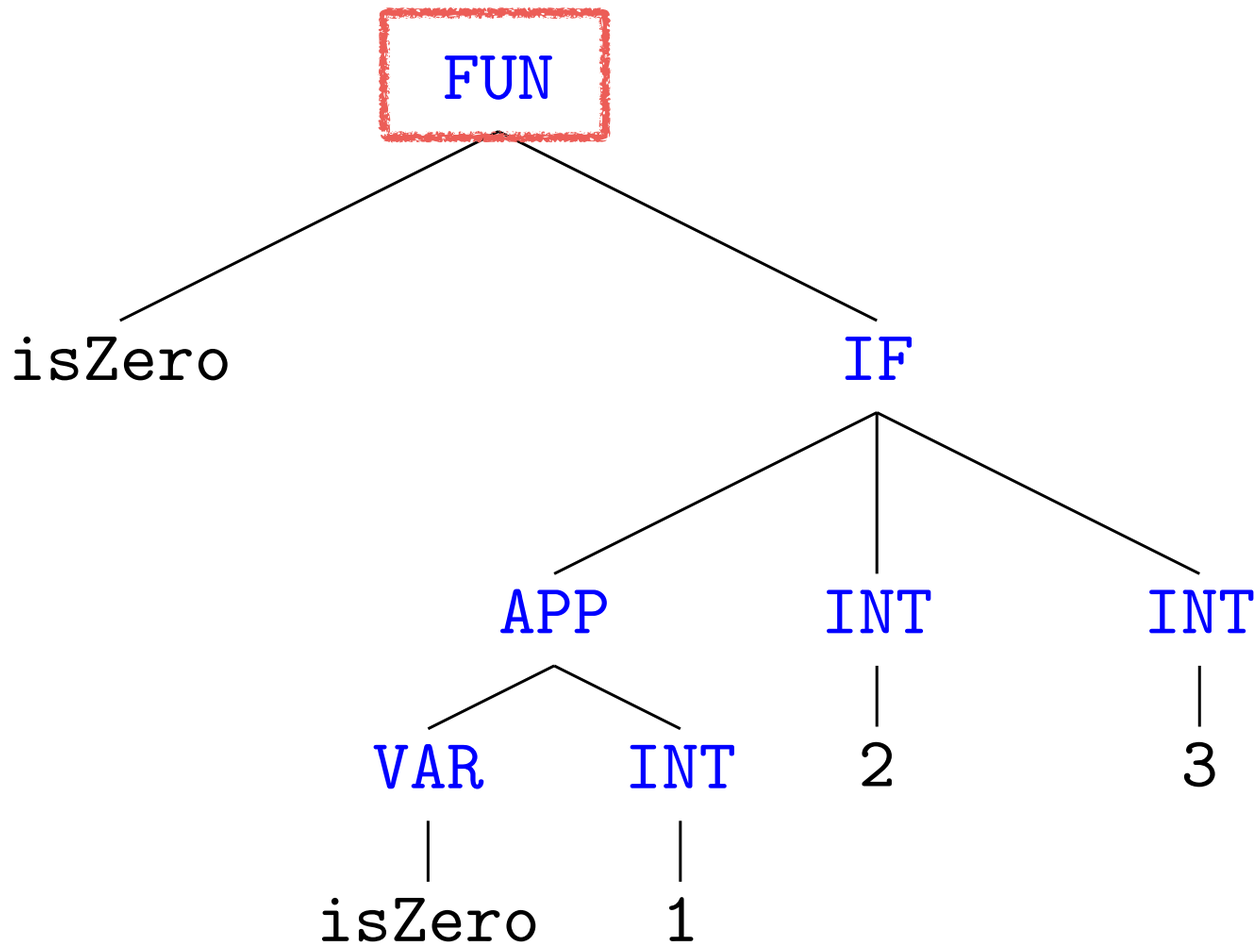

```
fn isZero =>
  if isZero 1
  then 2
  else 3
```



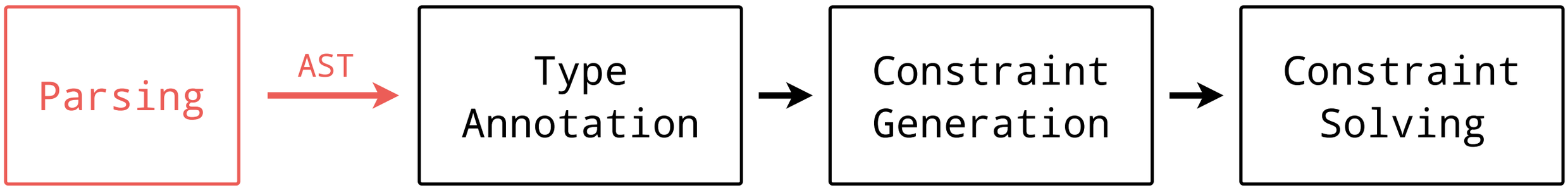
Wand's Algorithm Overview



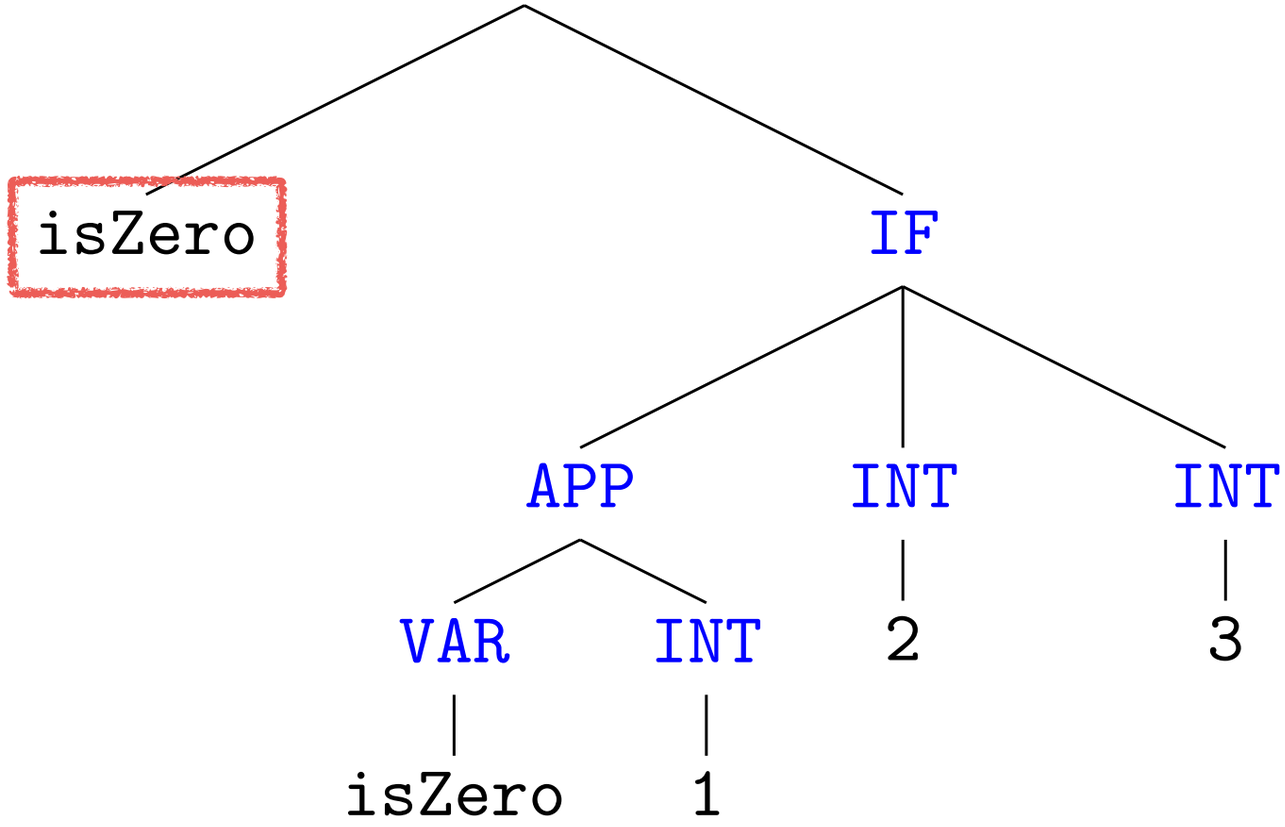

```
fn isZero =>
  if isZero 1
  then 2
  else 3
```



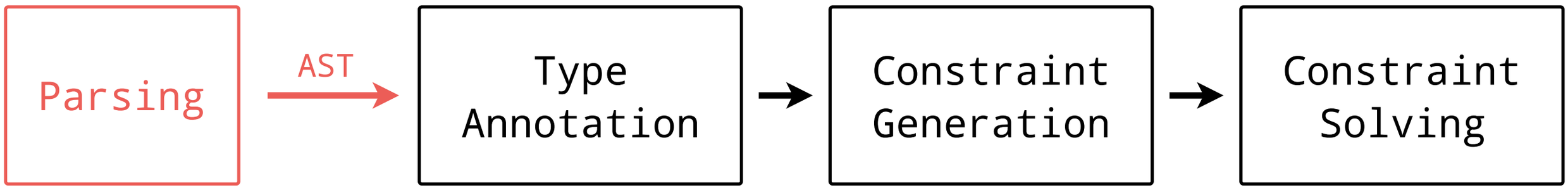
Wand's Algorithm Overview



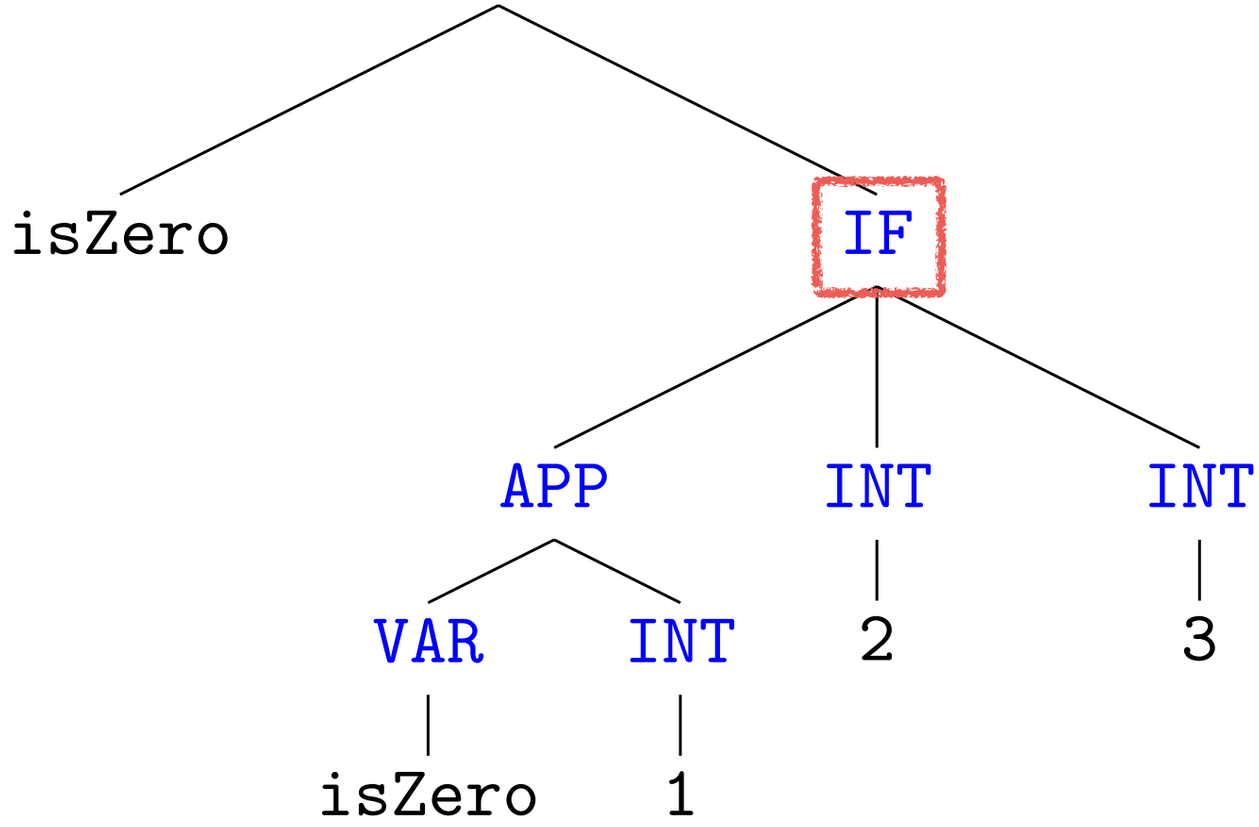

```
fn isZero =>
   if isZero 1
   then 2
   else 3
```



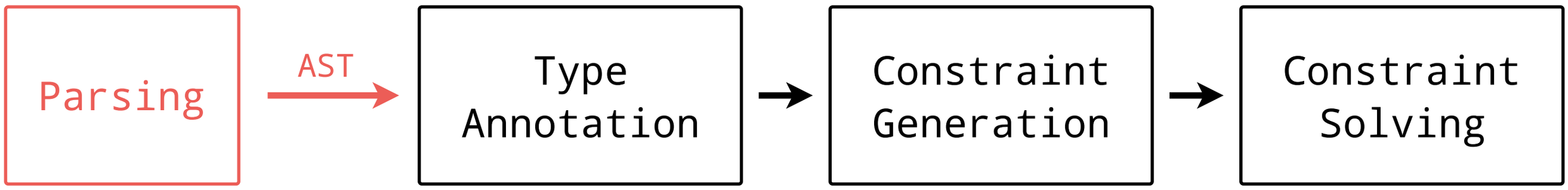
Wand's Algorithm Overview



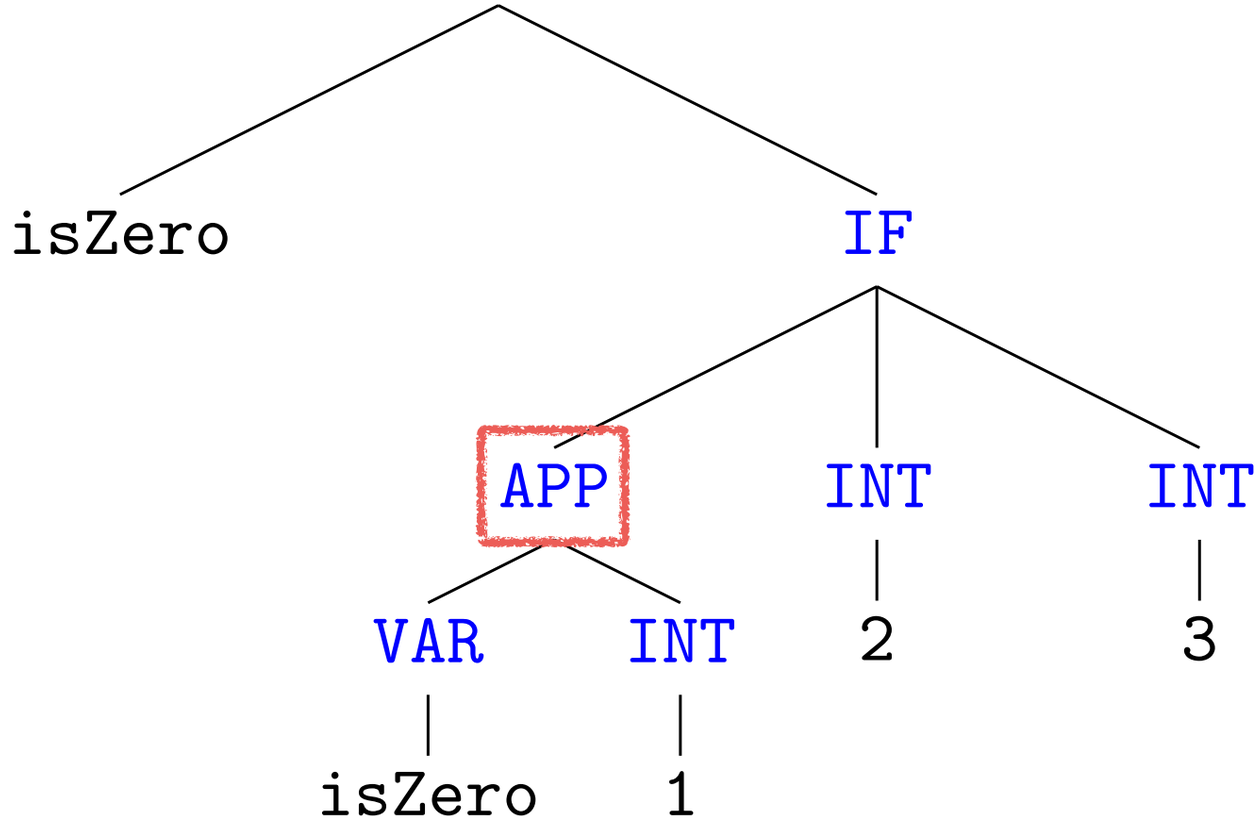

```
fn isZero =>
  if isZero 1
  then 2
  else 3
```



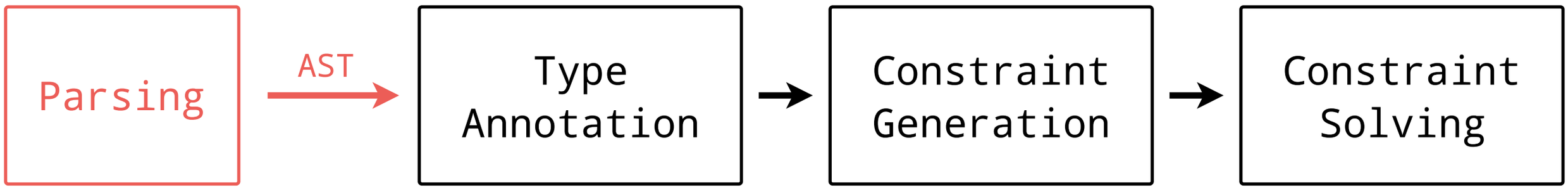
Wand's Algorithm Overview



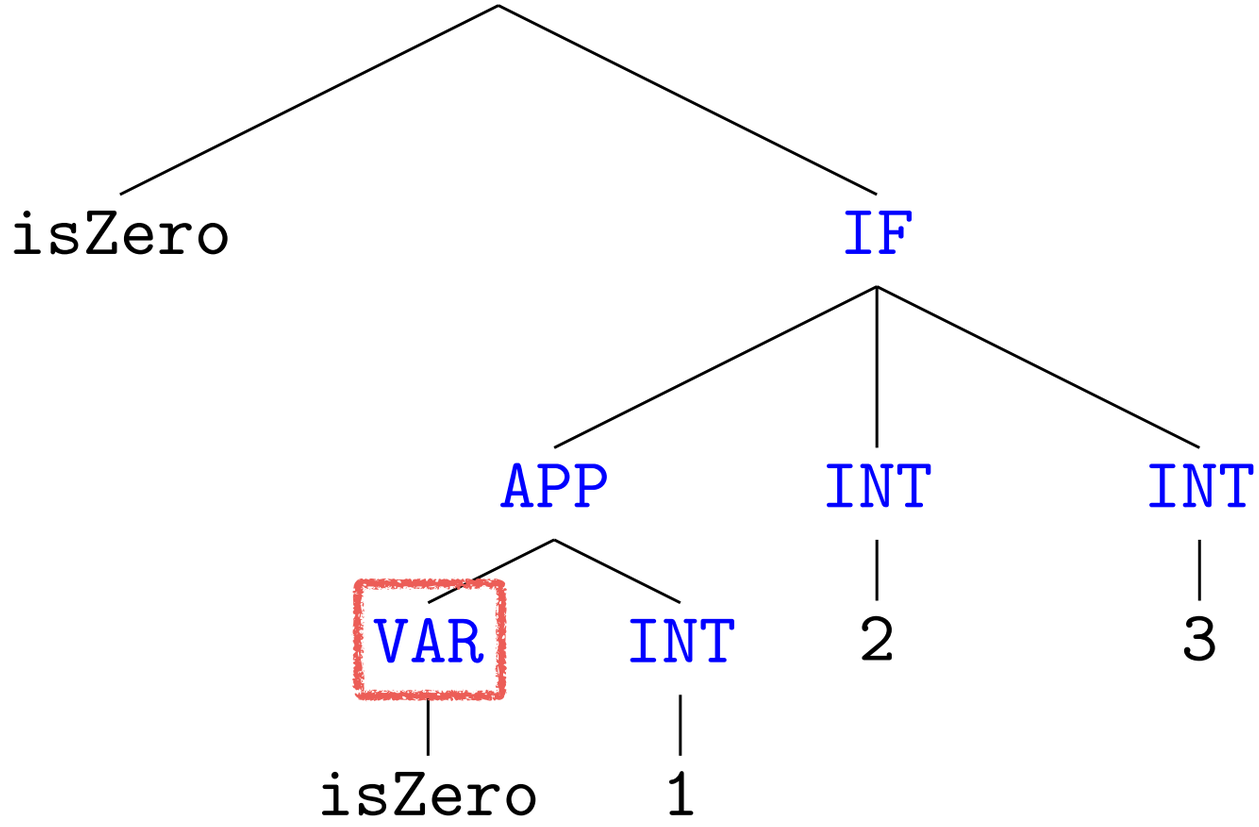

```
fn isZero =>
  if isZero 1
  then 2
  else 3
```



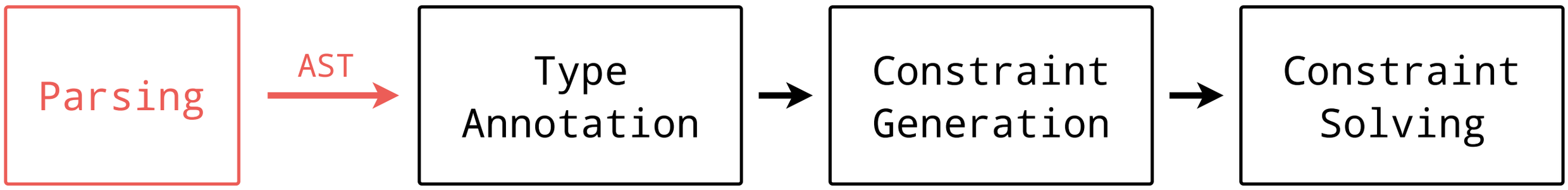
Wand's Algorithm Overview



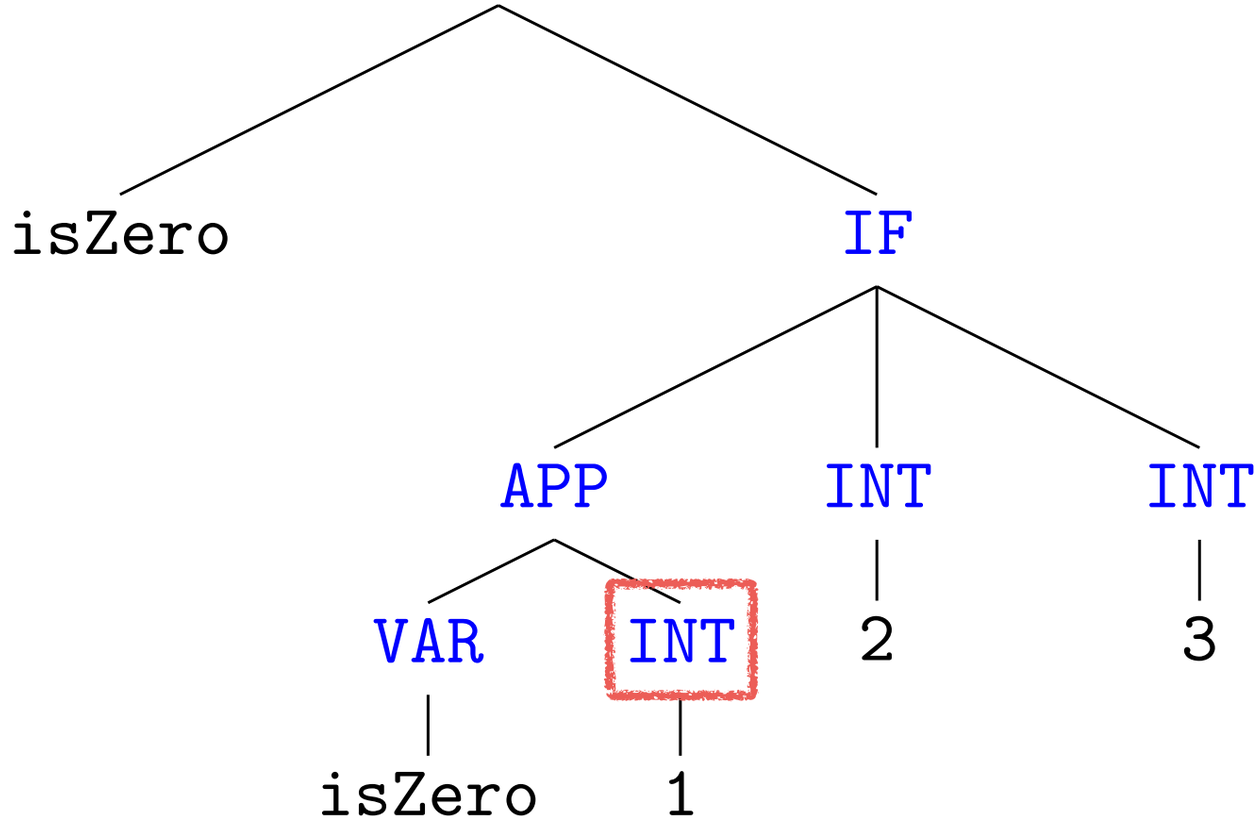

```
fn isZero =>
  if isZero 1
  then 2
  else 3
```



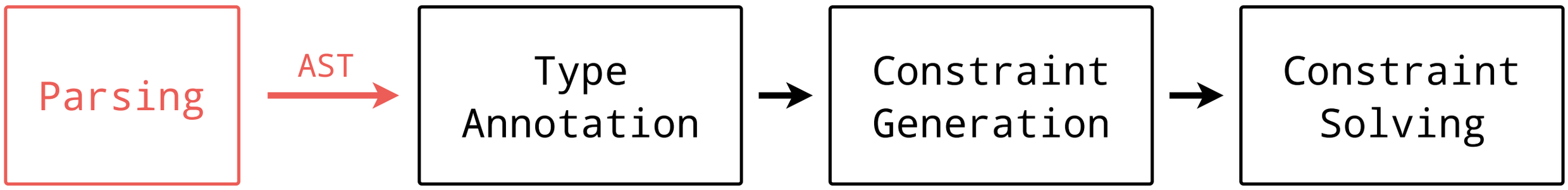
Wand's Algorithm Overview



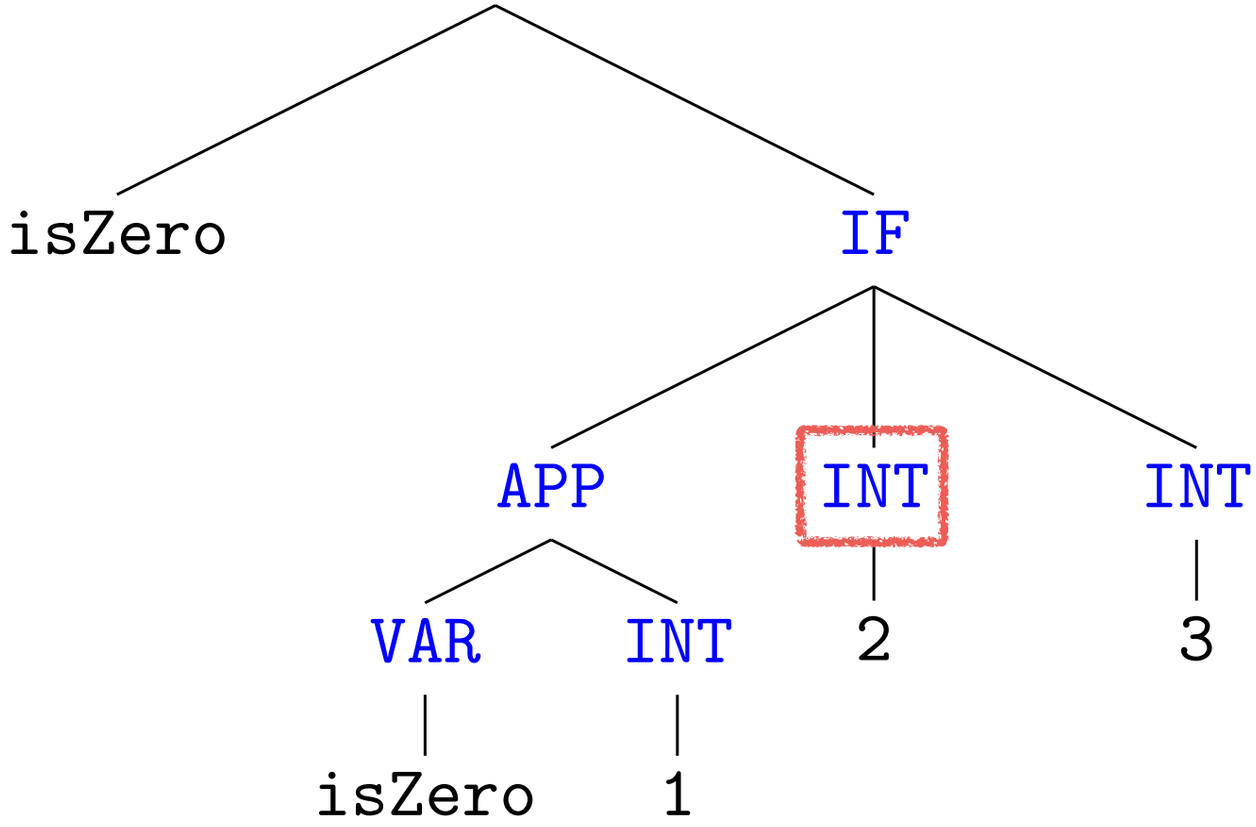

```
fn isZero =>
  if isZero 1
  then 2
  else 3
```



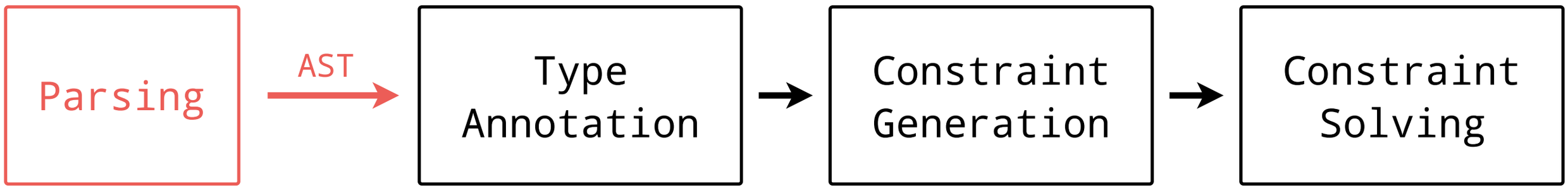
Wand's Algorithm Overview



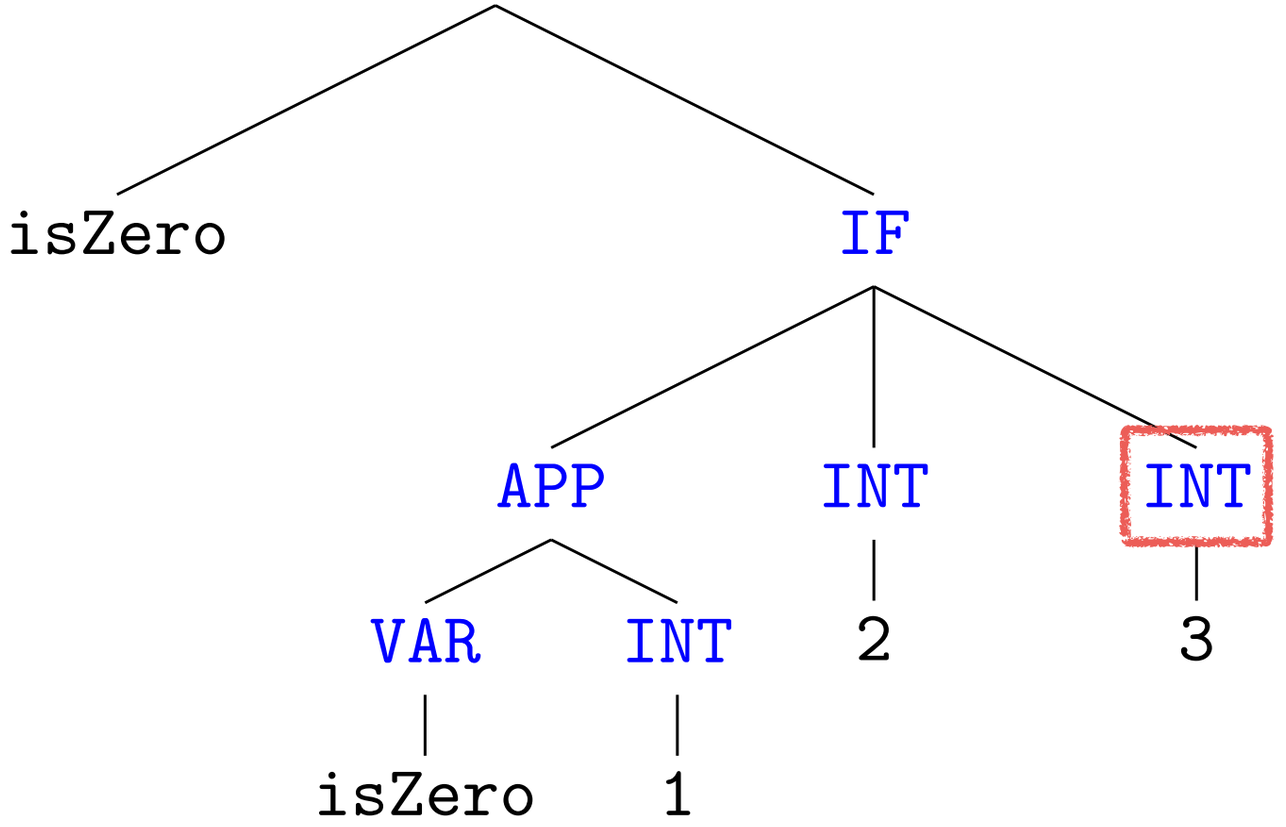

```
fn isZero =>
   if isZero 1
   then 2
   else 3
```



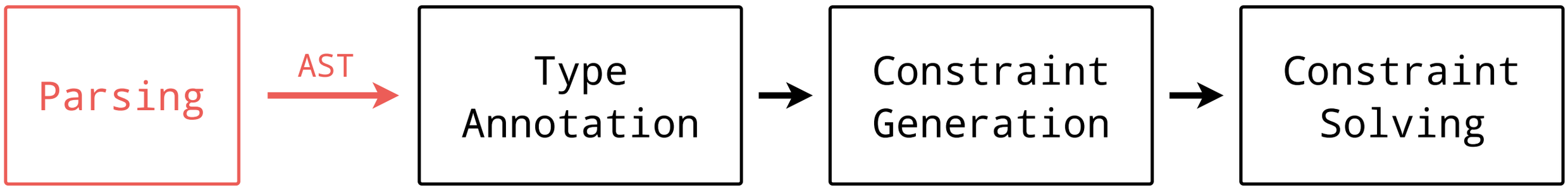
Wand's Algorithm Overview



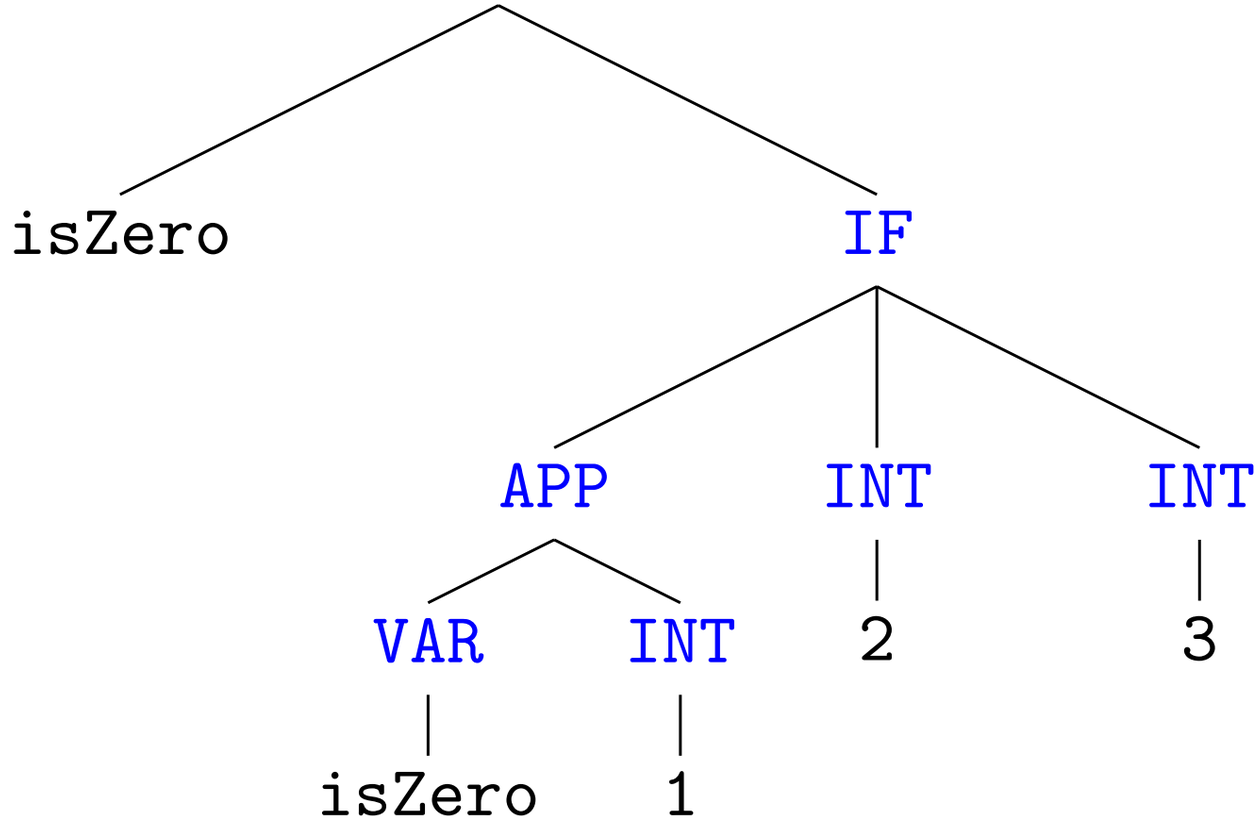

```
fn isZero =>
  if isZero 1
  then 2
  else 3
```



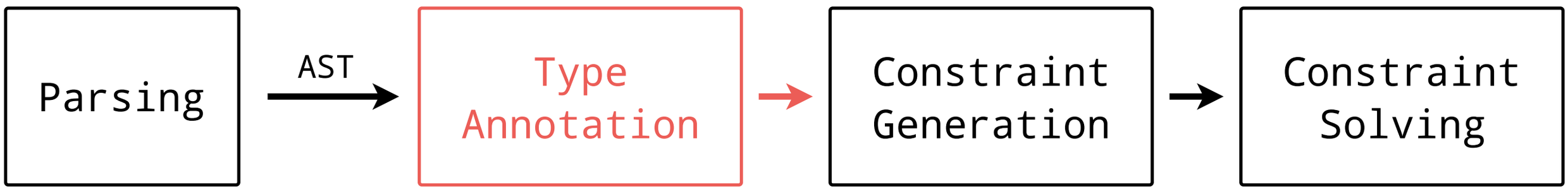
Wand's Algorithm Overview

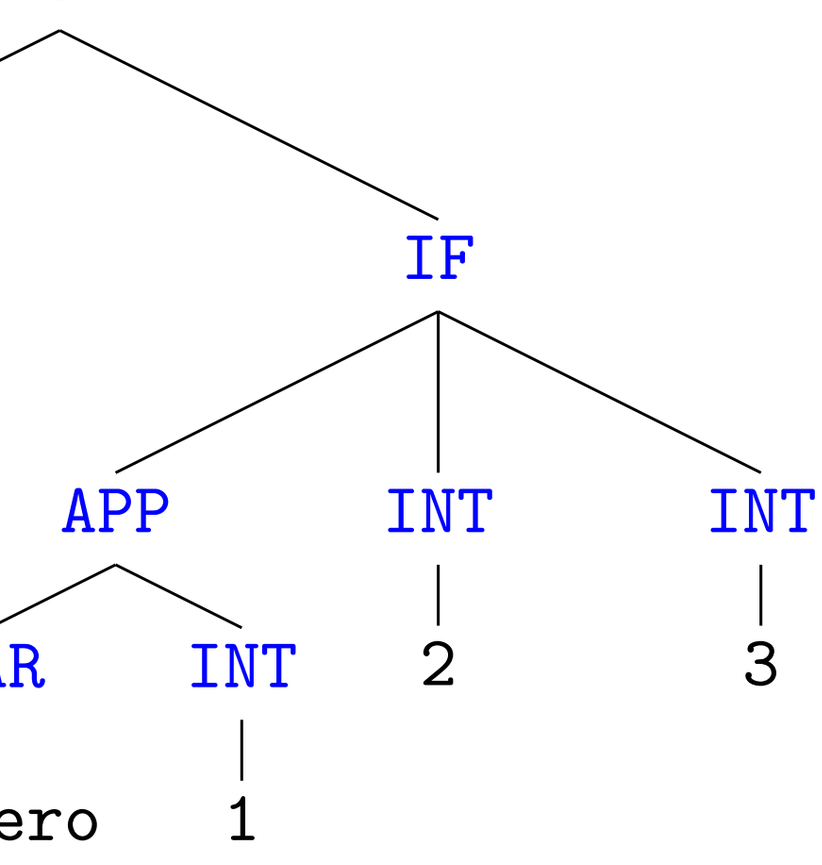


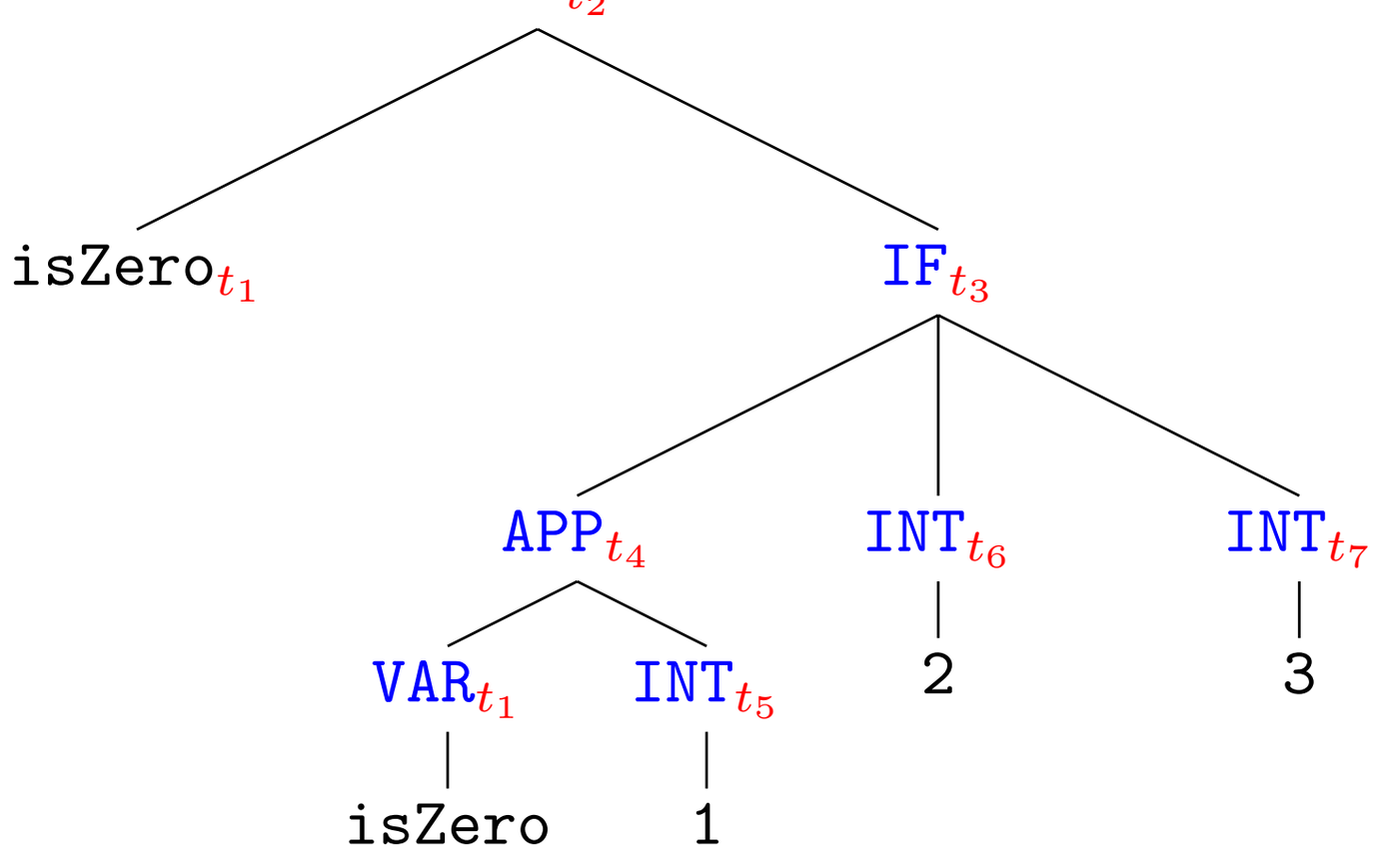

```
fn isZero =>
  if isZero 1
  then 2
  else 3
```



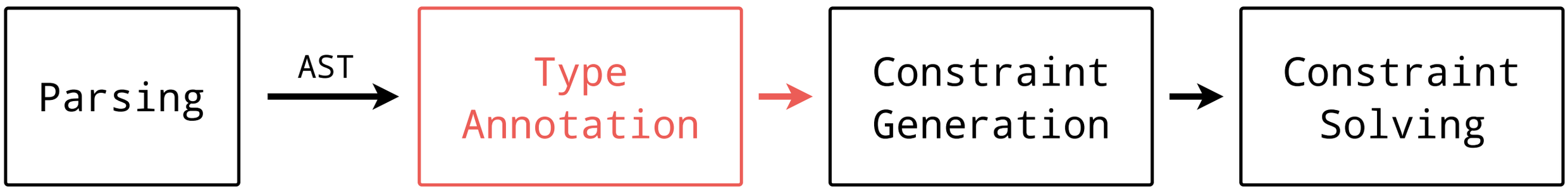
Wand's Algorithm Overview

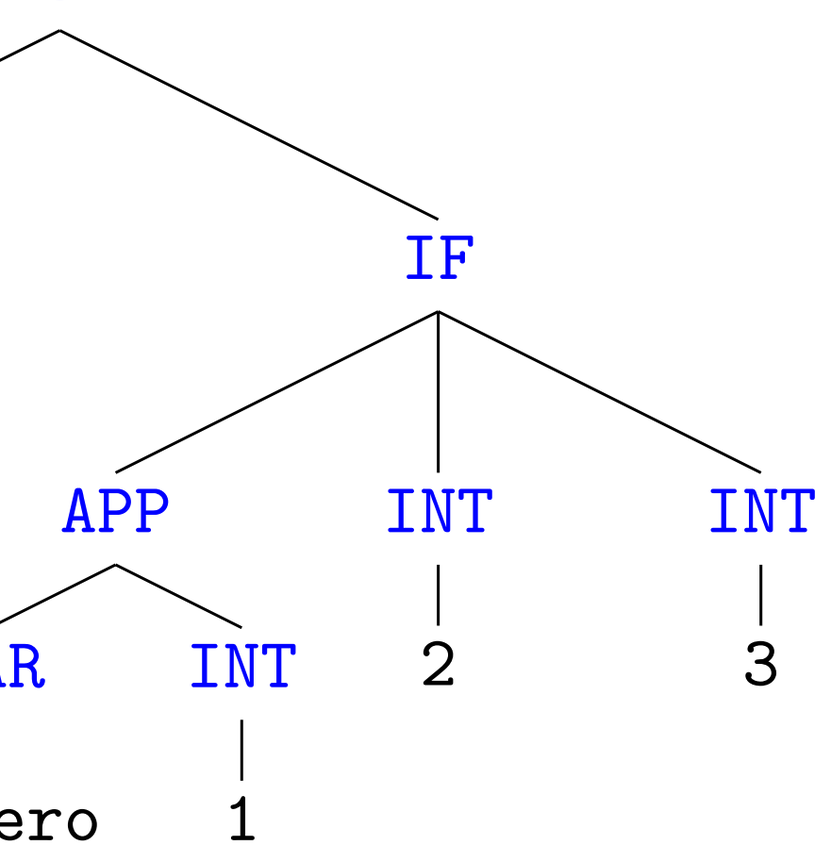


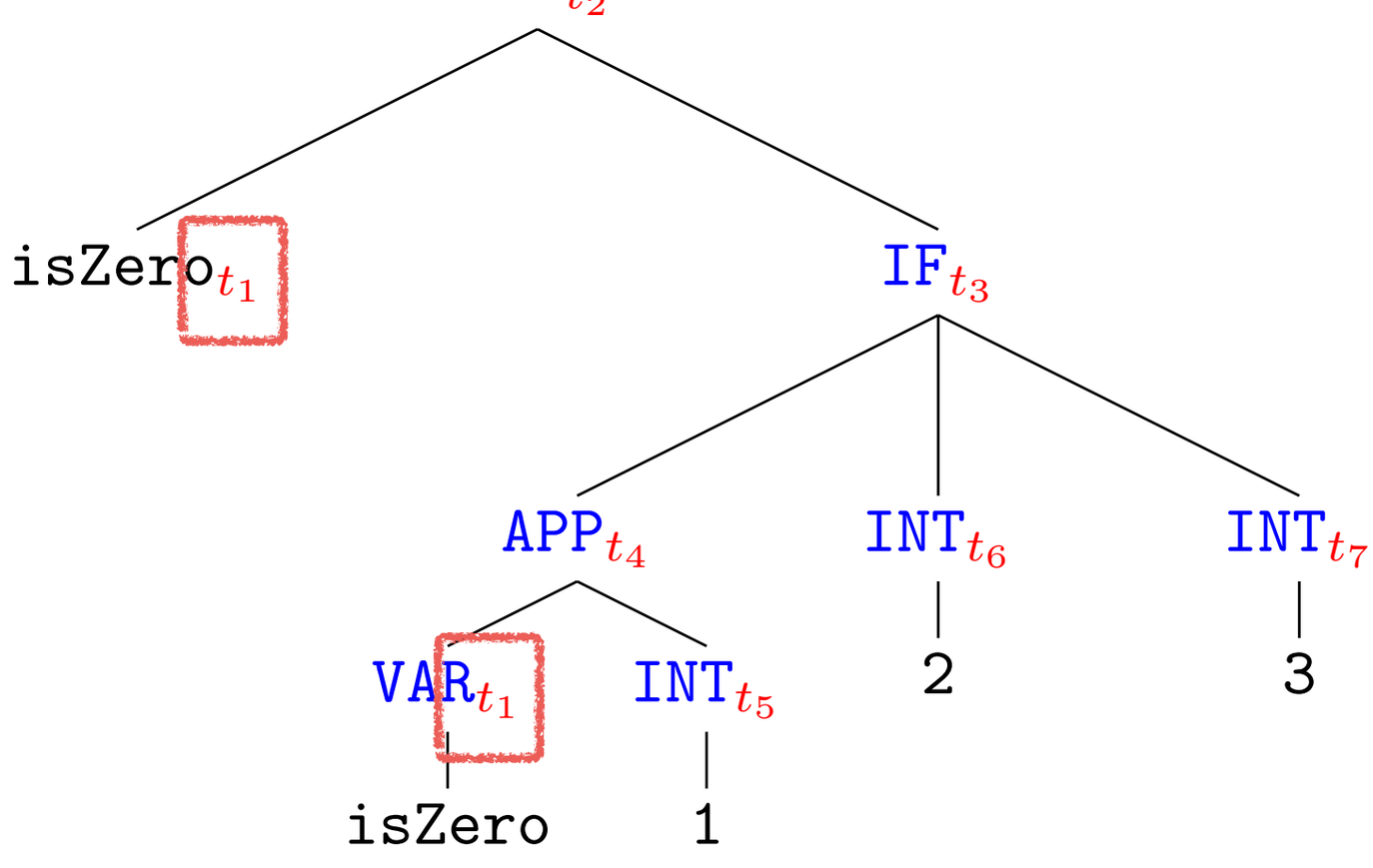




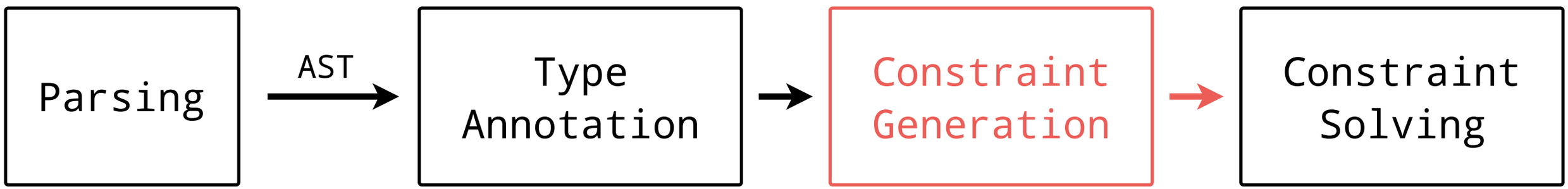
Wand's Algorithm Overview

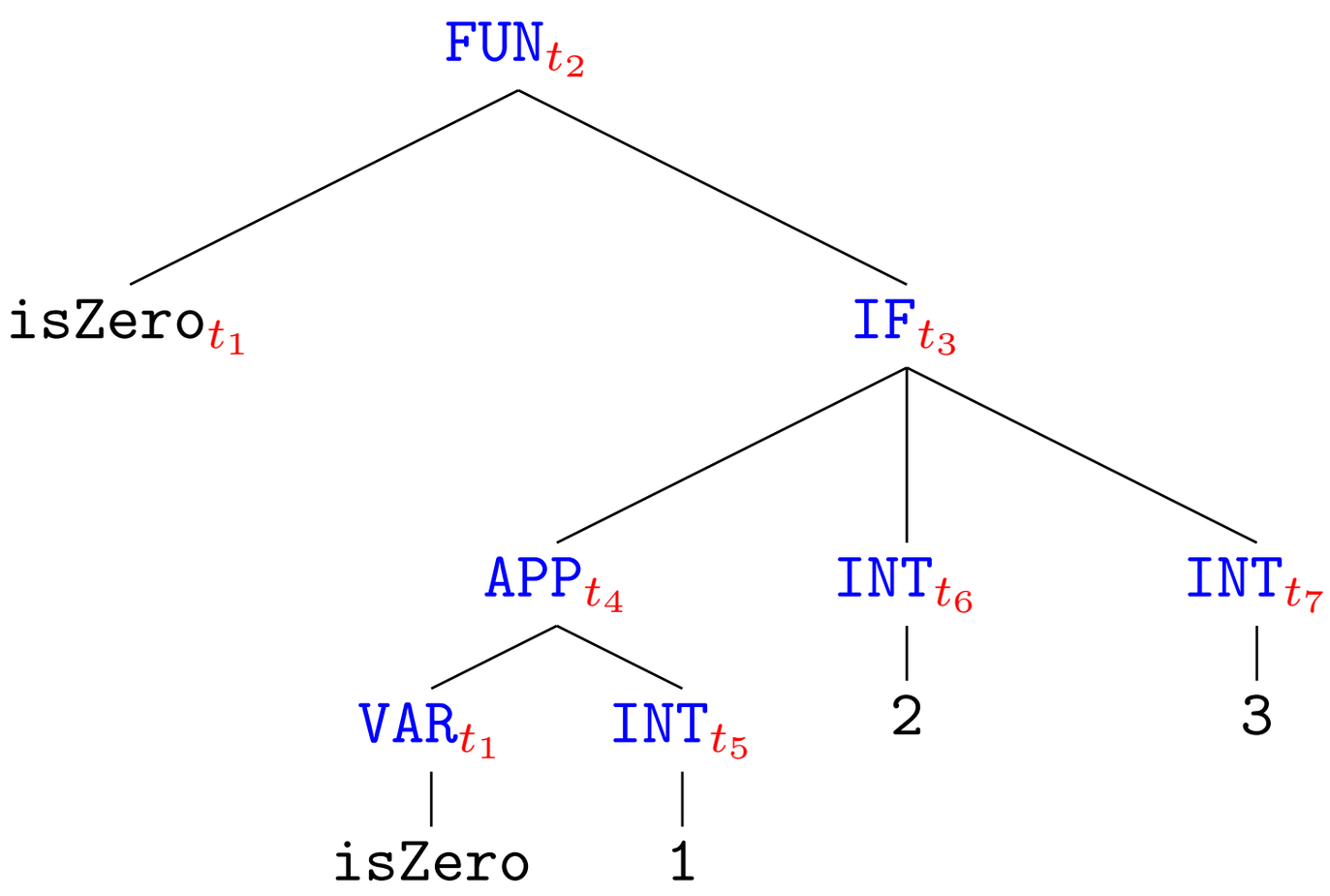






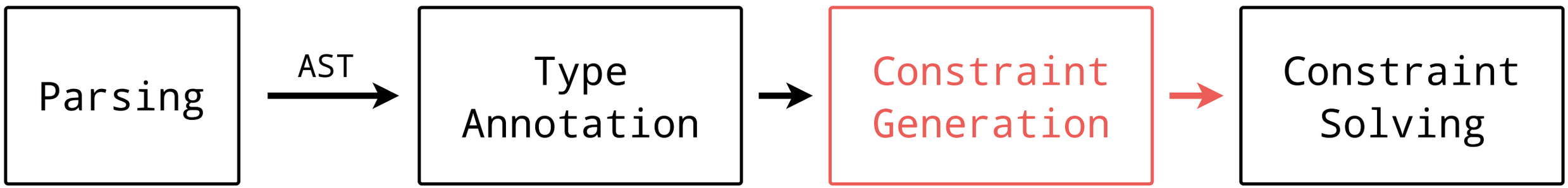
Wand's Algorithm Overview

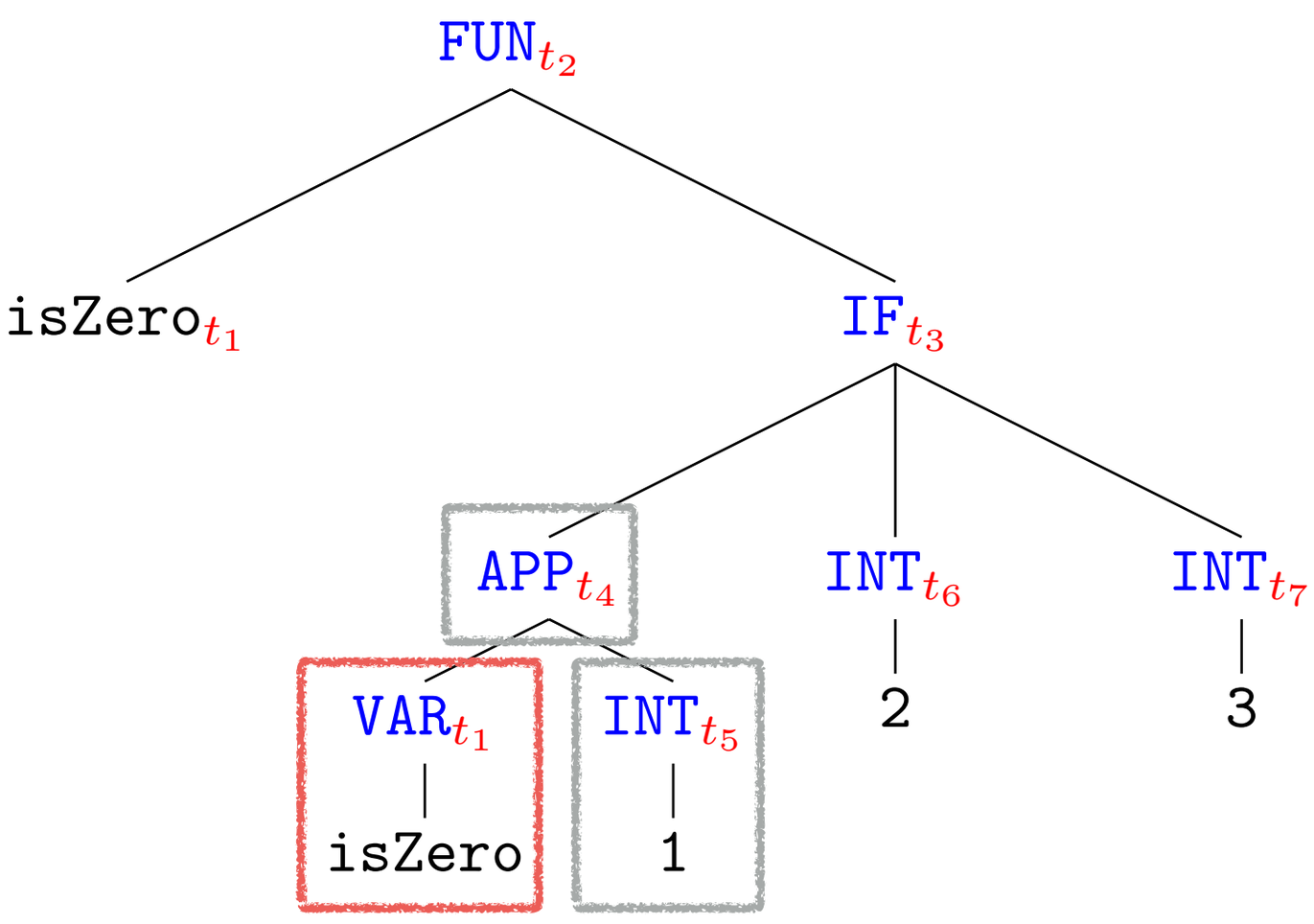




Constraint Set

Wand's Algorithm Overview

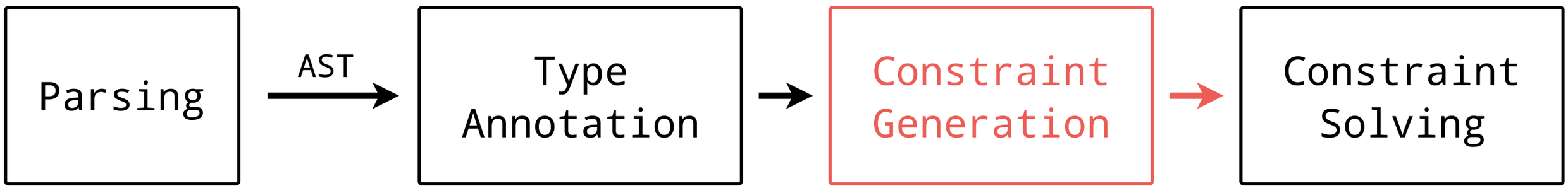


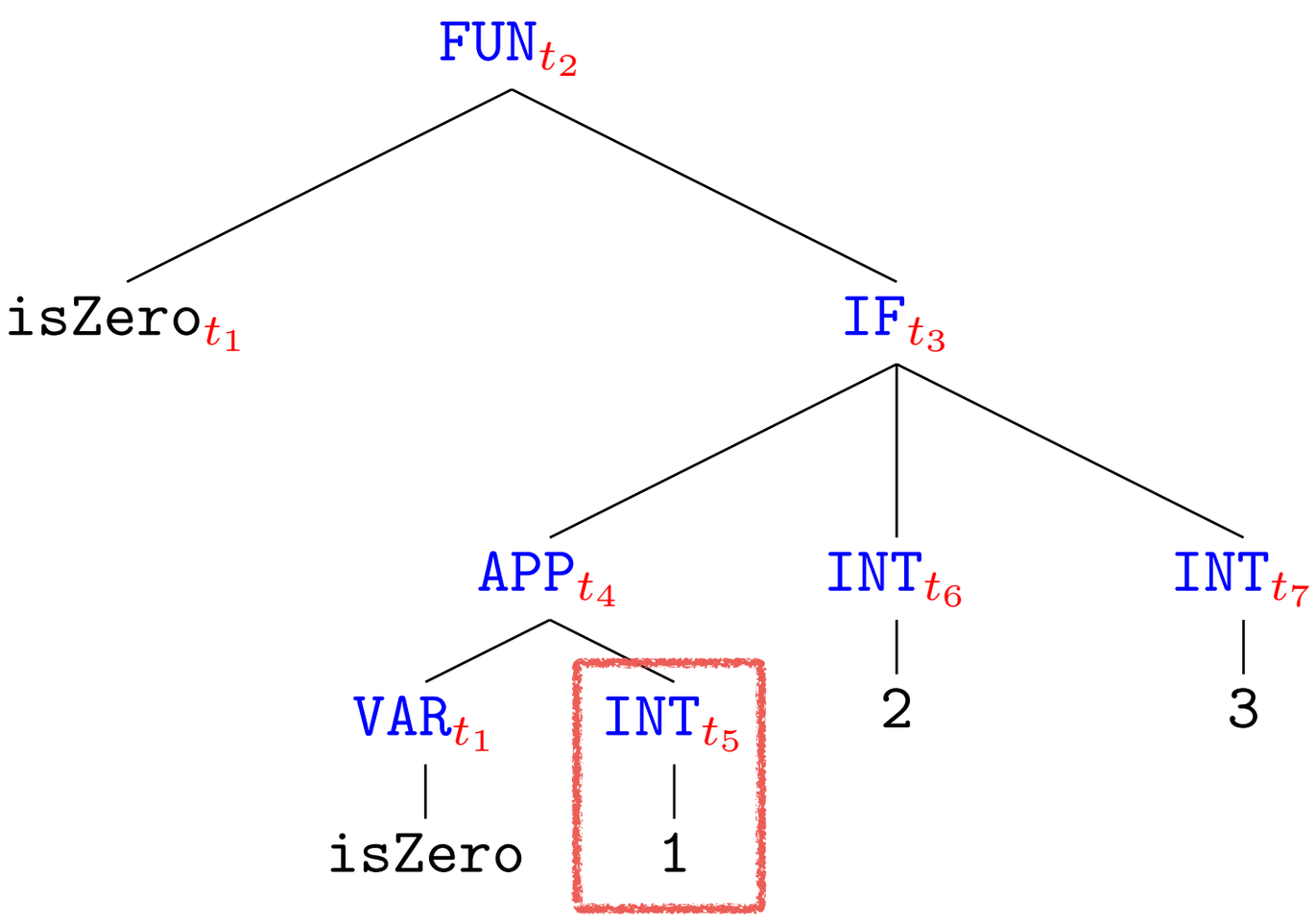


Constraint Set

```
t1 ≡ t5 → t4
```

Wand's Algorithm Overview

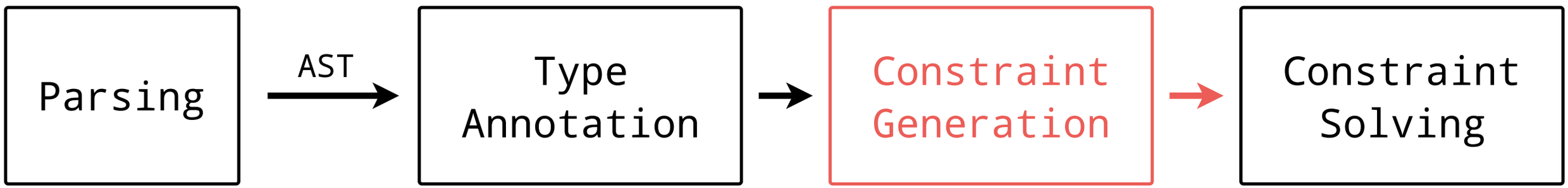


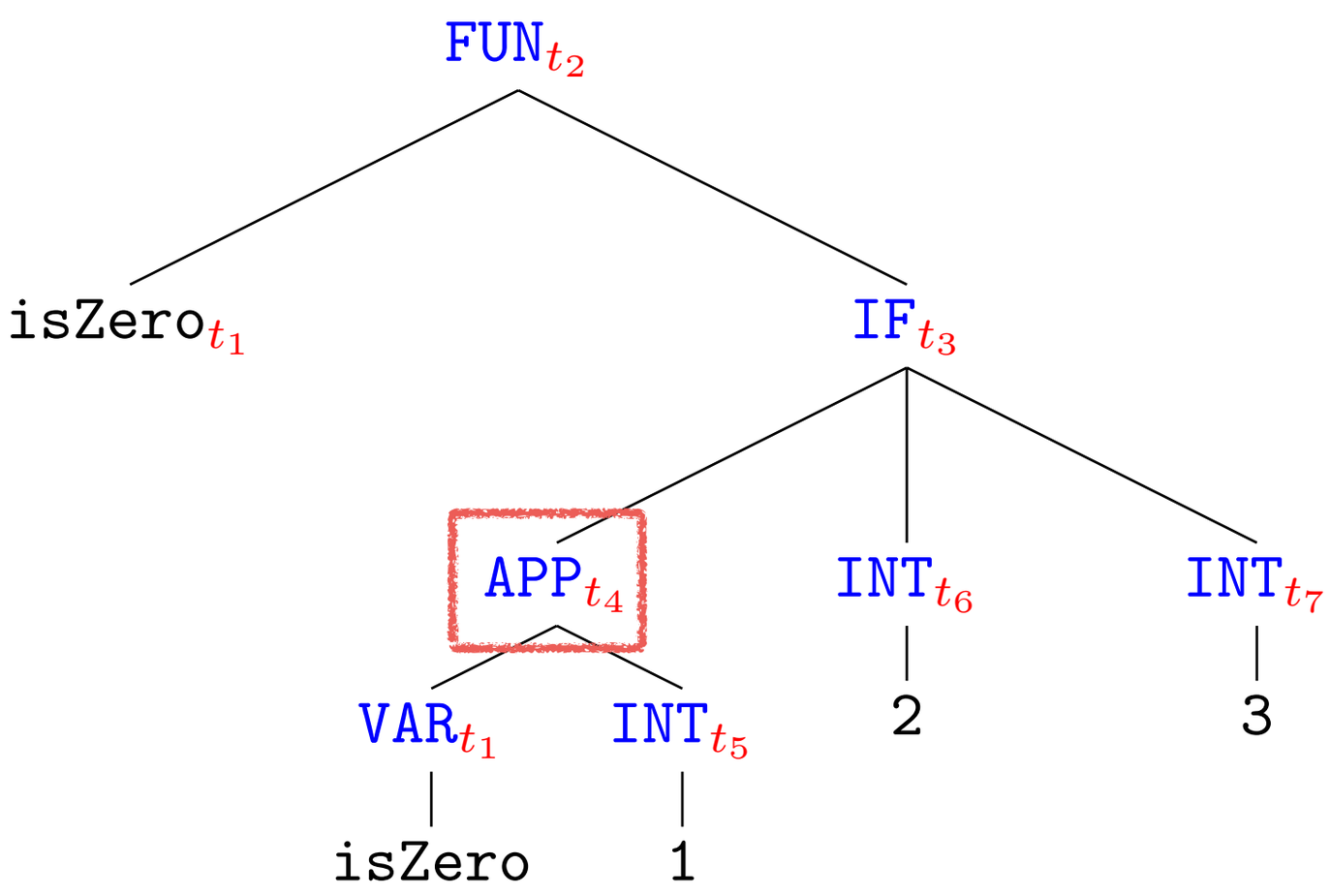


Constraint Set

```
t1 ≡ t5 → t4
t5 ≡ int
```

Wand's Algorithm Overview

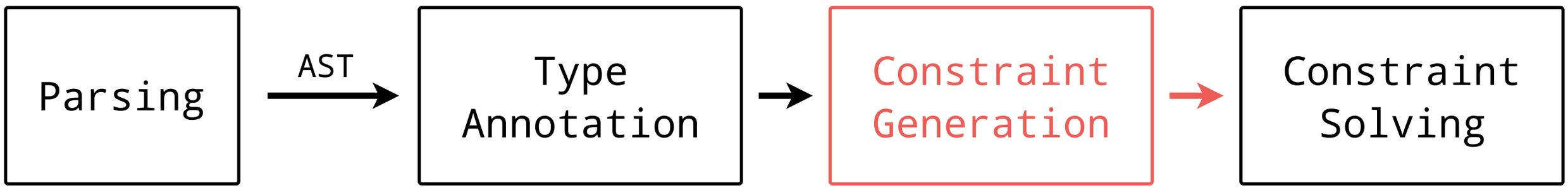


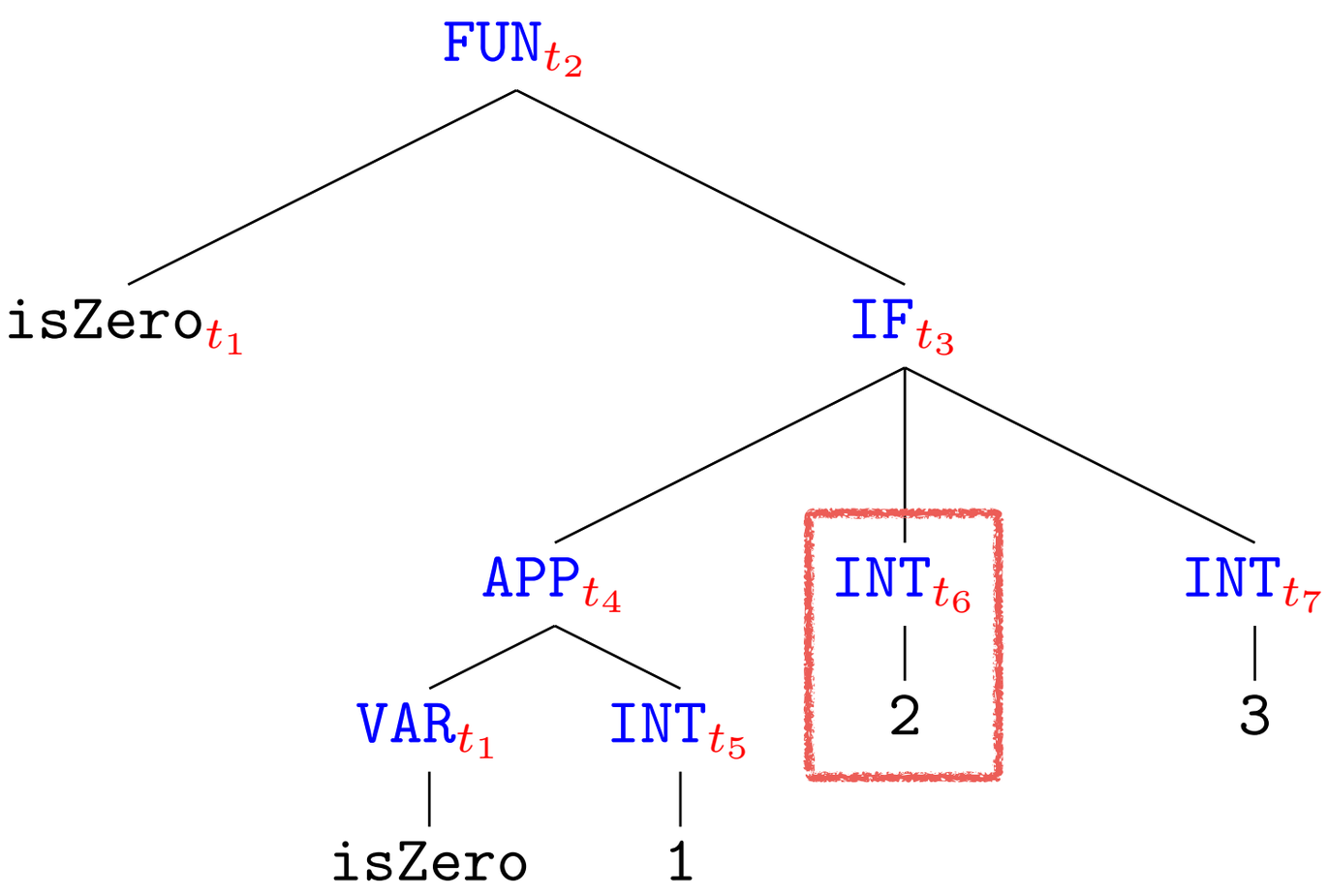


Constraint Set

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

Wand's Algorithm Overview

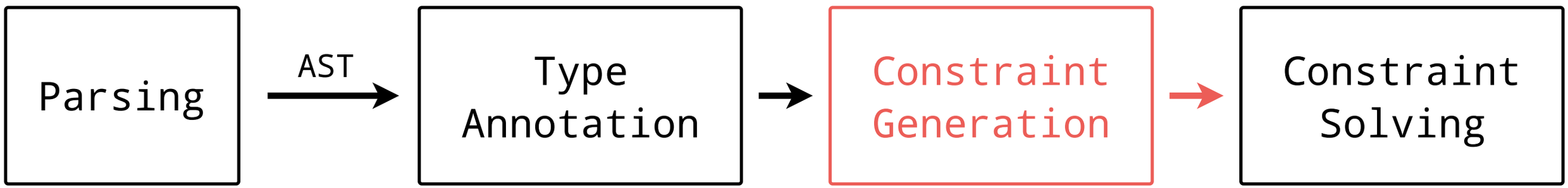


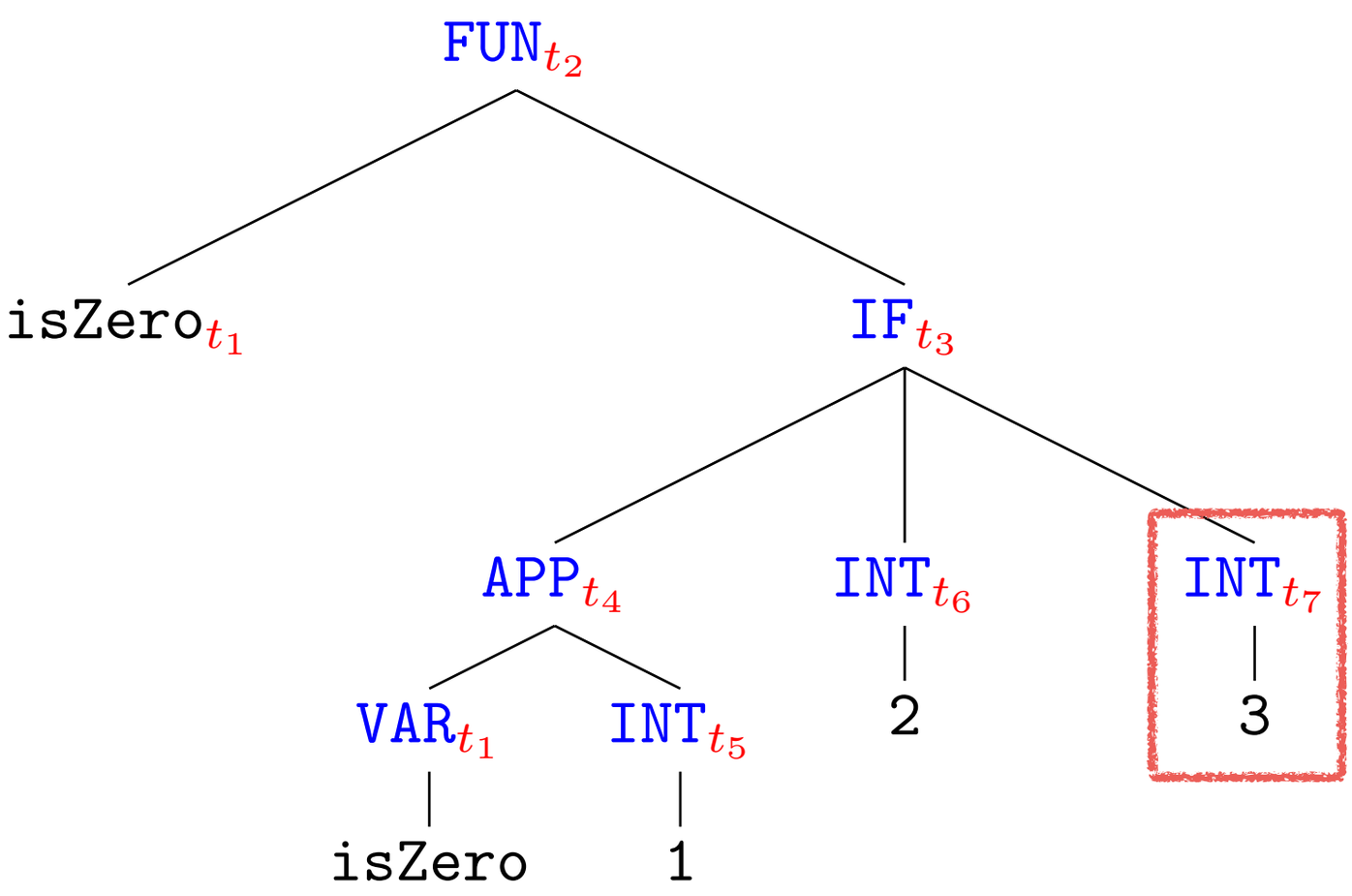


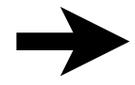
Constraint Set

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

Wand's Algorithm Overview



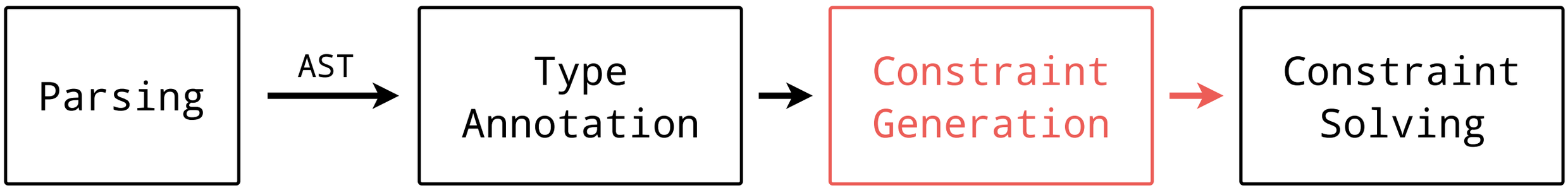


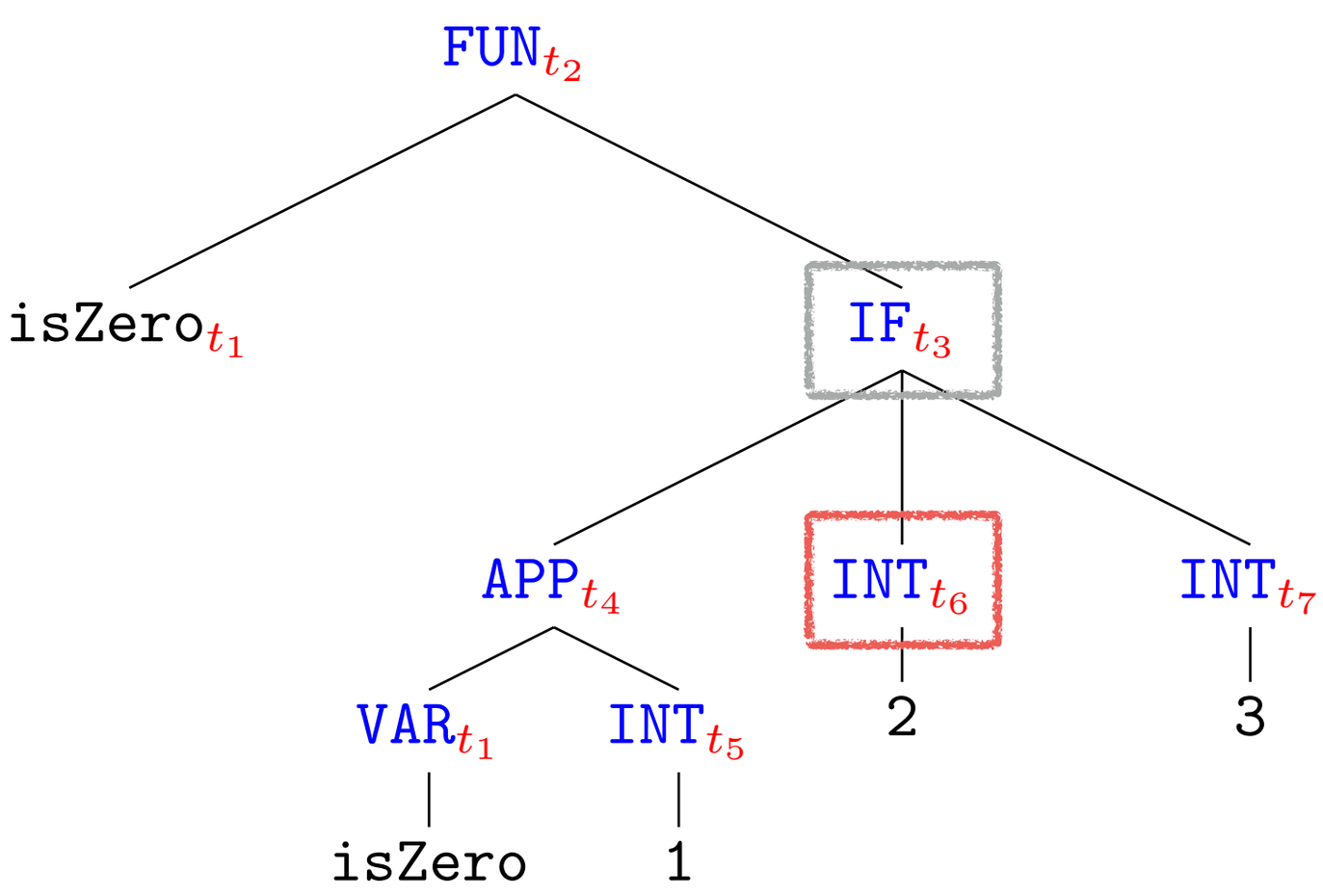


Constraint Set

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

Wand's Algorithm Overview

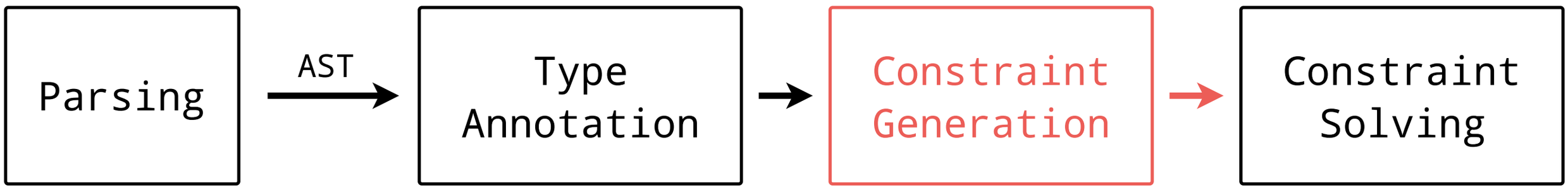


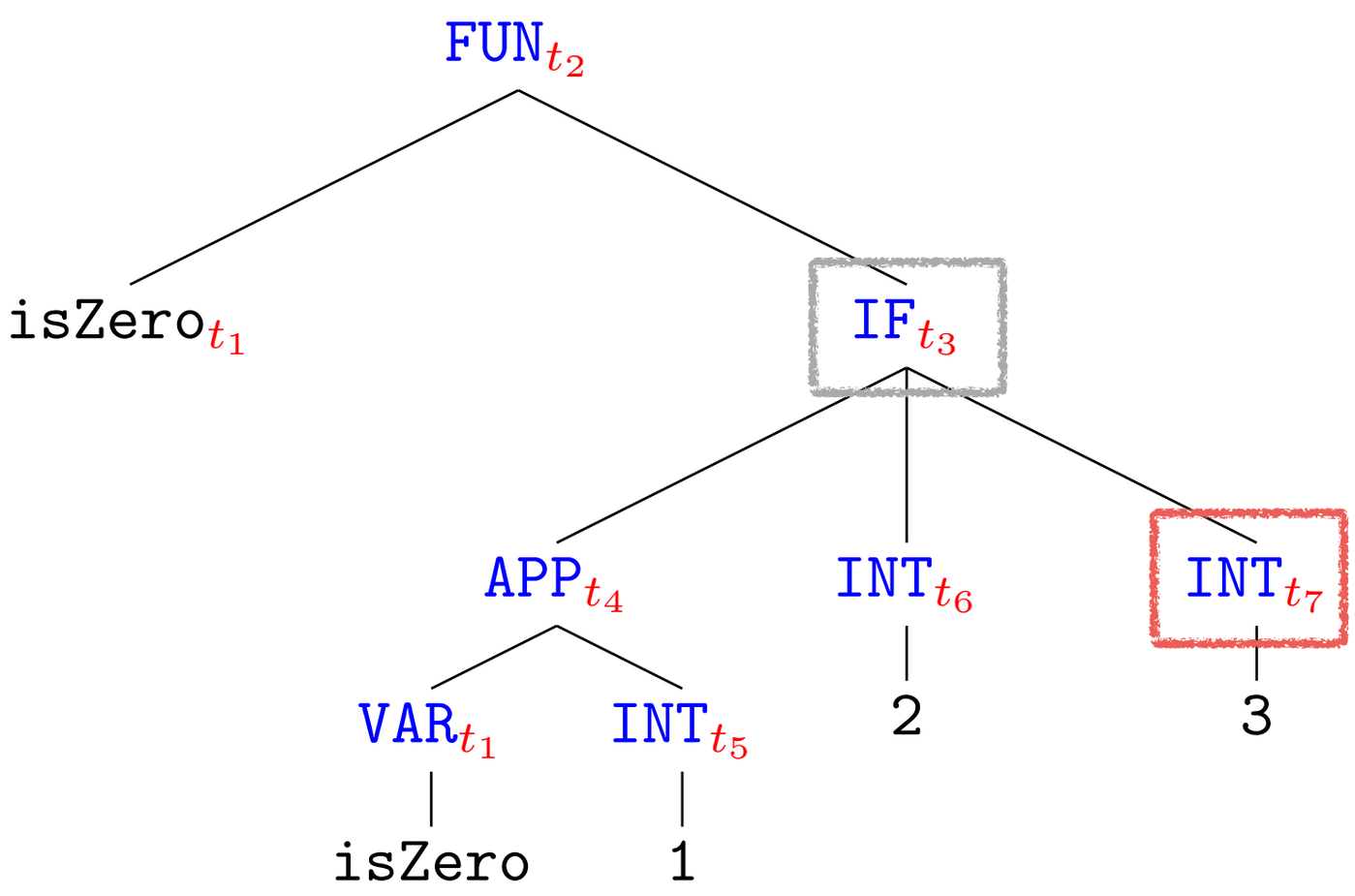


Constraint Set

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

Wand's Algorithm Overview

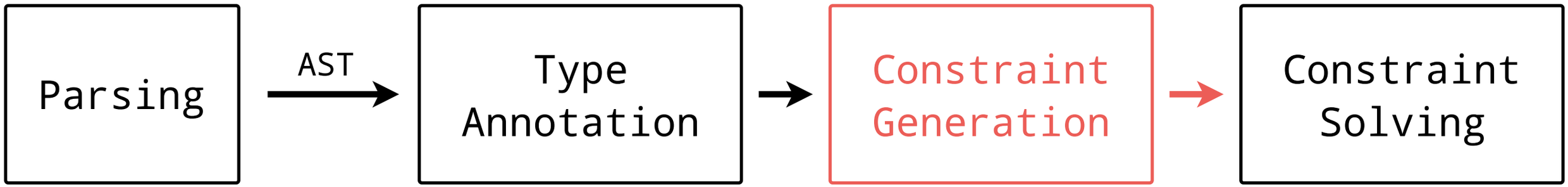


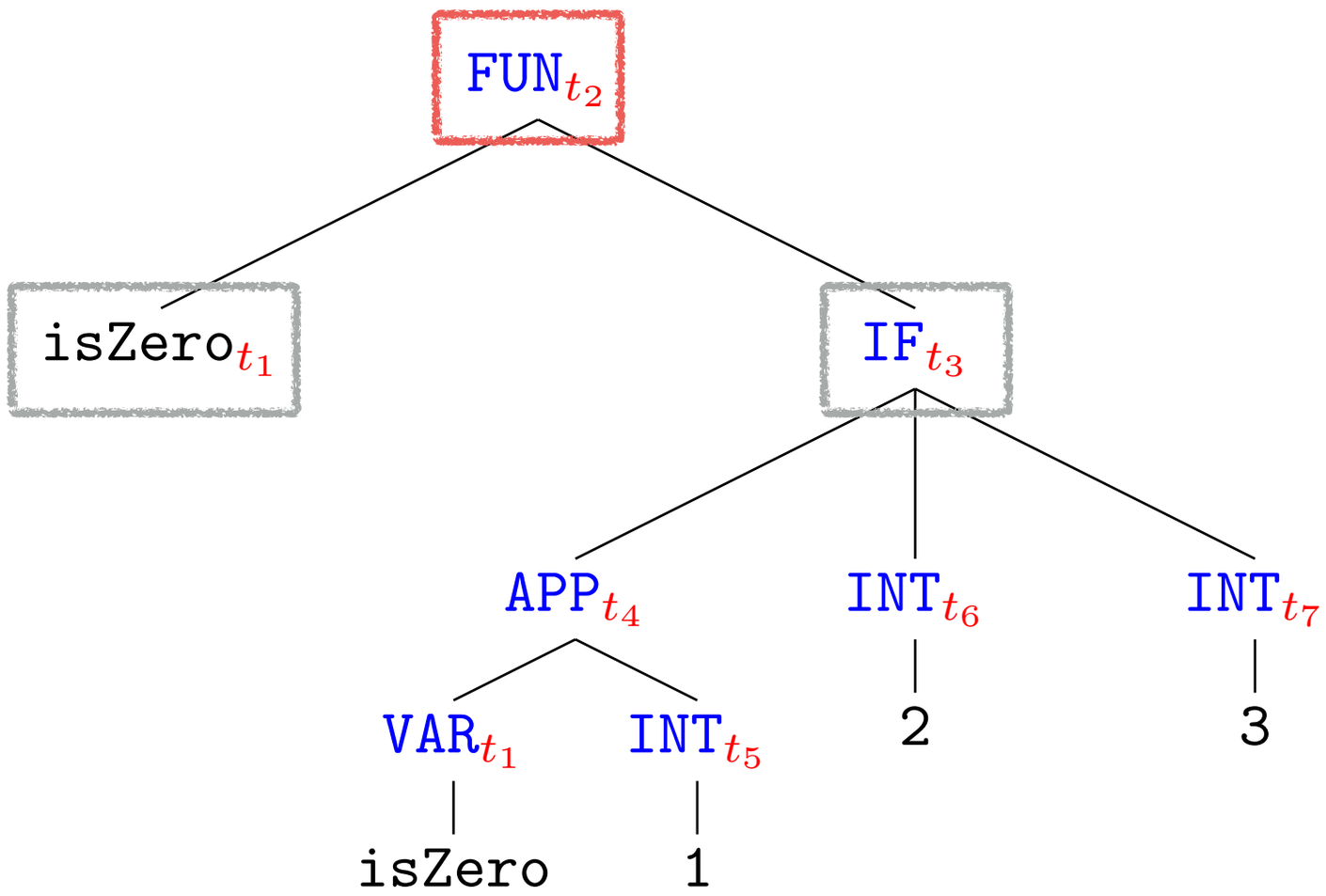


Constraint Set

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

Wand's Algorithm Overview

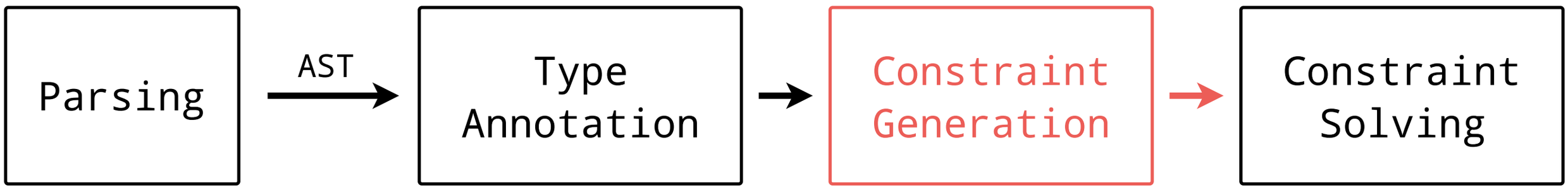


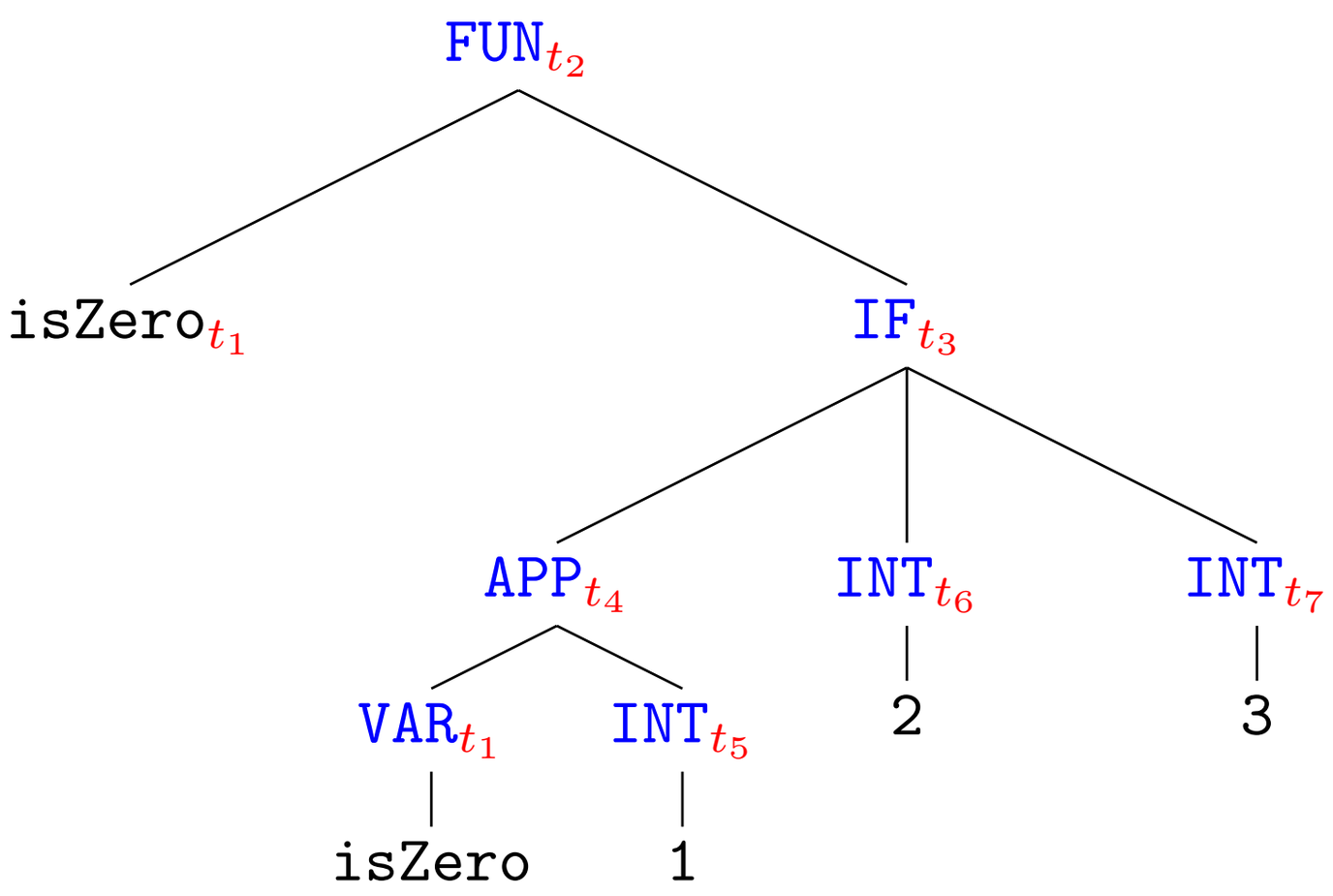


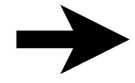
Constraint Set

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

Wand's Algorithm Overview



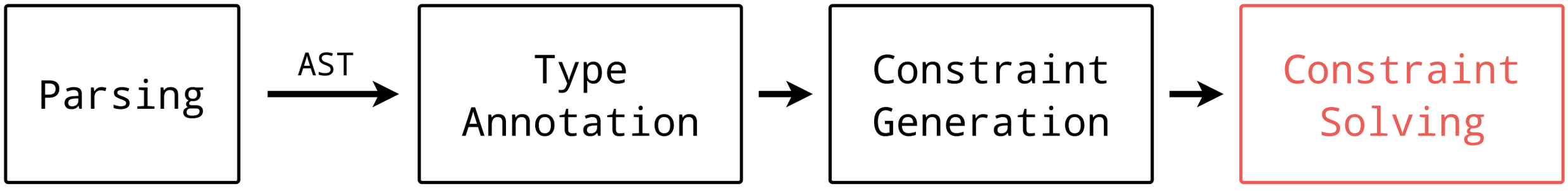




Constraint Set

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

Wand's Algorithm Overview

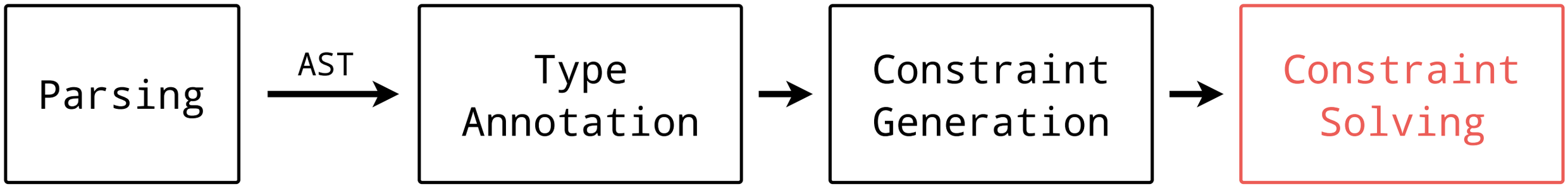


Constraint Set

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

Solution Map

Wand's Algorithm Overview



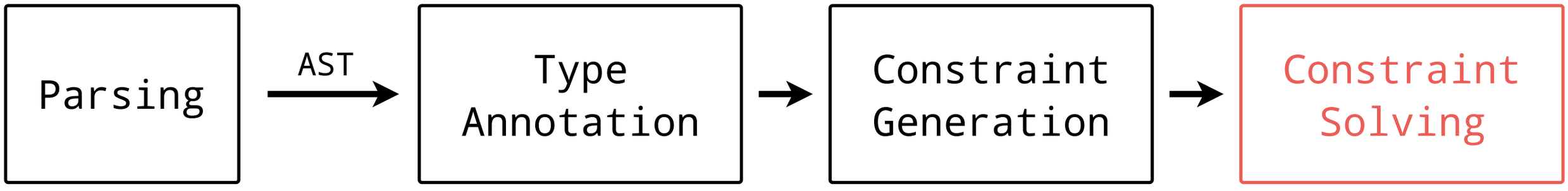
Constraint Set

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

Solution Map

```
t1: t5 → t4
```

Wand's Algorithm Overview



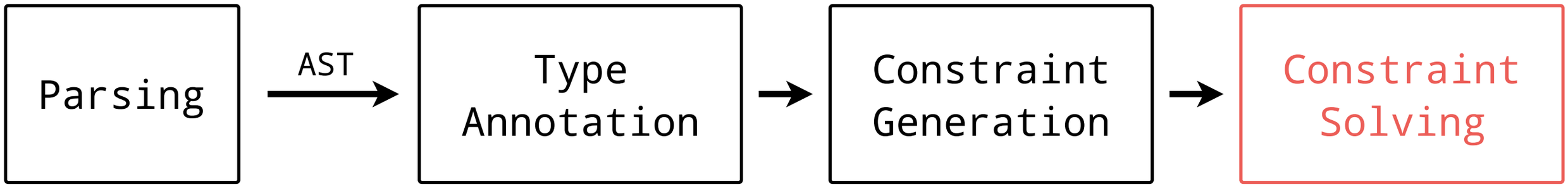
Constraint Set

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

Solution Map

```
t1: t5 → t4
```

Wand's Algorithm Overview



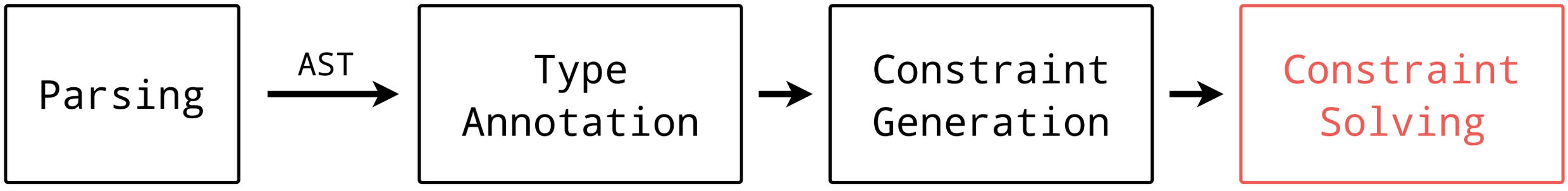
Constraint Set

```
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



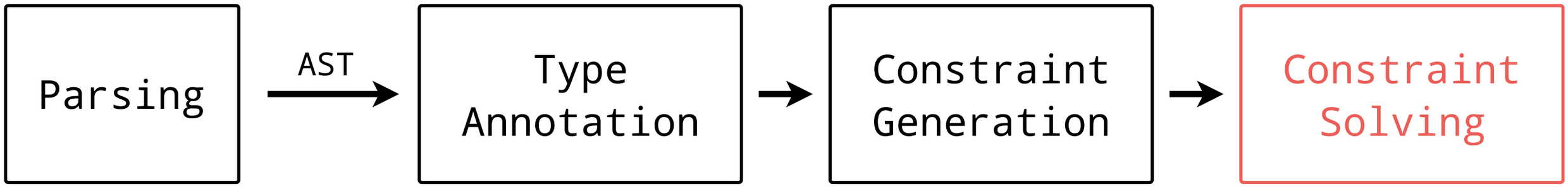
Constraint Set

```
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



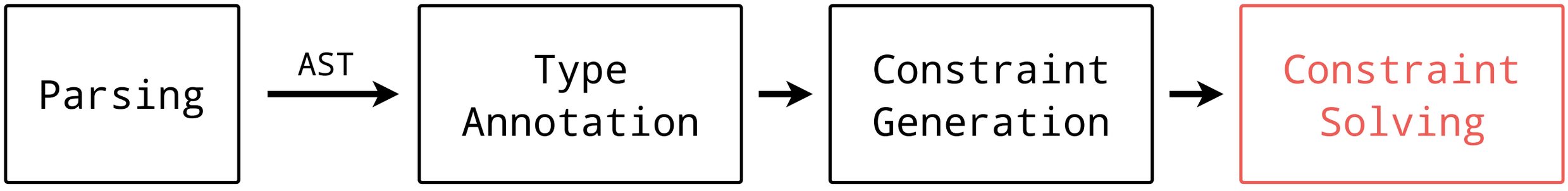
Constraint Set

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

Solution Map

```
t1: t5 → t4
t5: int
```

Wand's Algorithm Overview



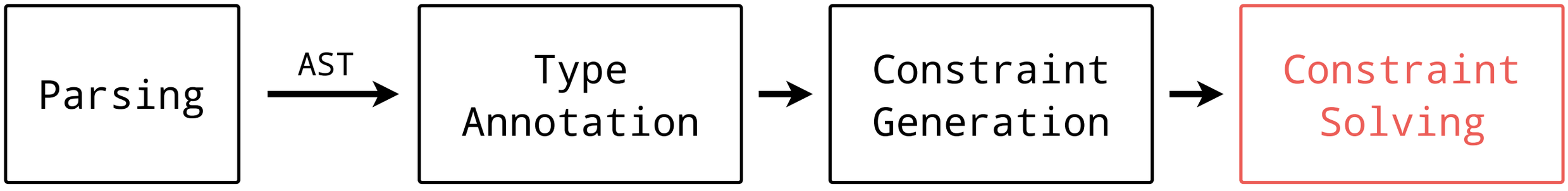
Constraint Set

```
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



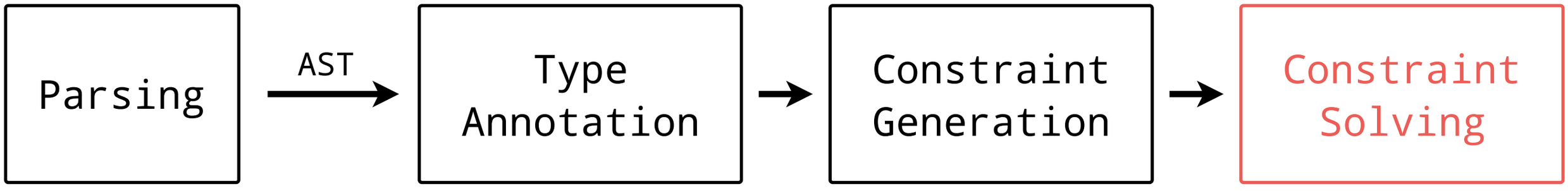
Constraint Set

```
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



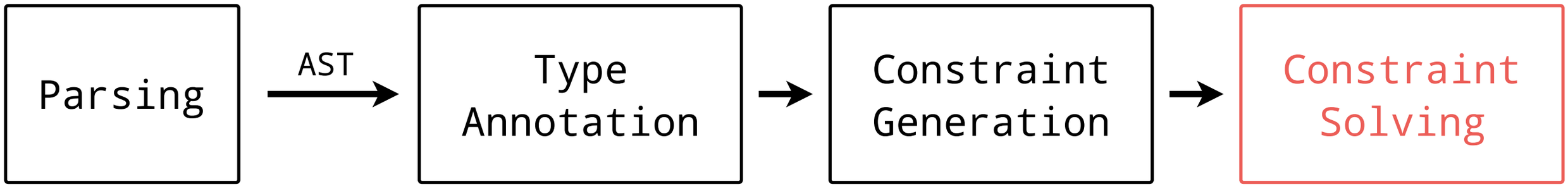
Constraint Set

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

Solution Map

```
t1: int → t4
t5: int
t4: bool
```

Wand's Algorithm Overview



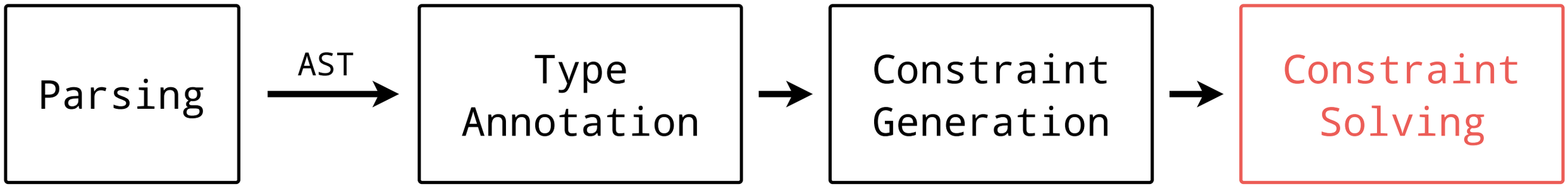
Constraint Set

```
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



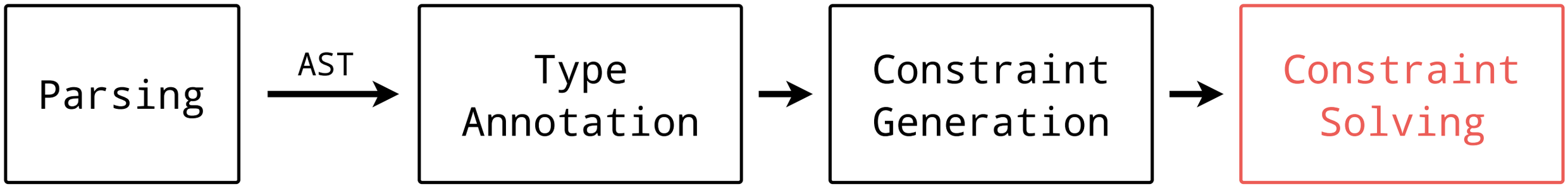
Constraint Set

```
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



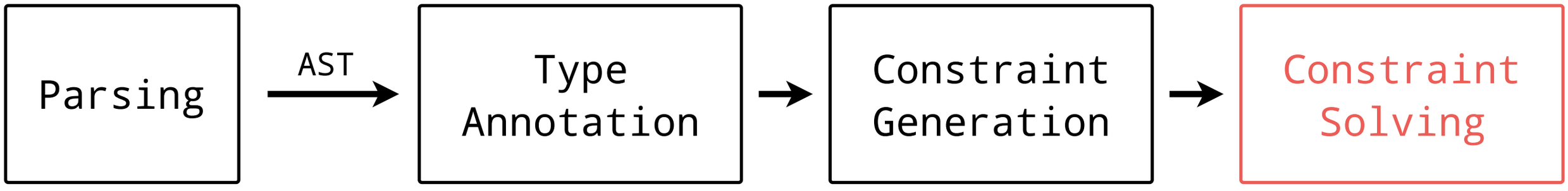
Constraint Set

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

Solution Map

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

Wand's Algorithm Overview



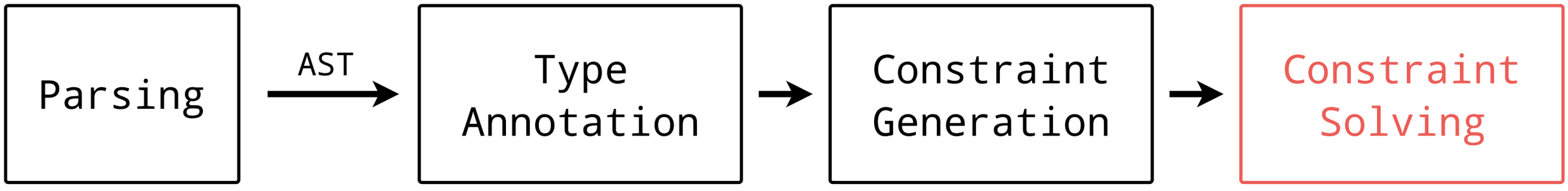
Constraint Set

```
 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



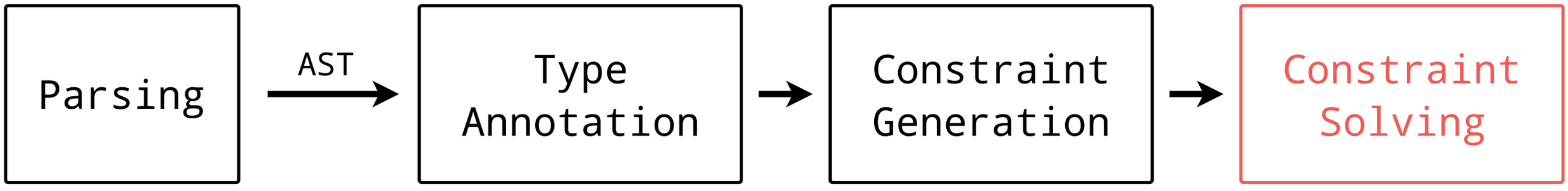
Constraint Set

```
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



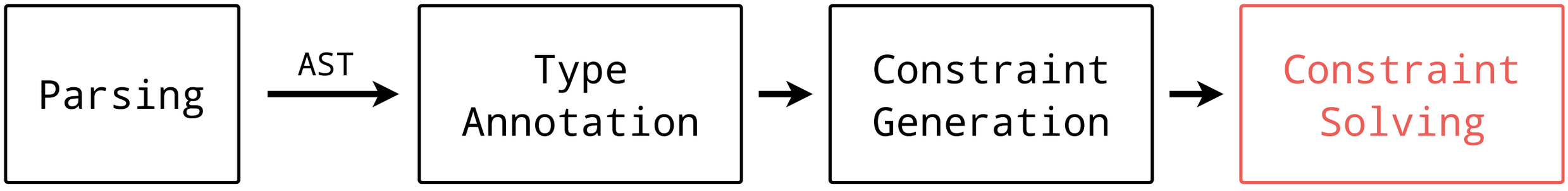
Constraint Set

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

Solution Map

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

Wand's Algorithm Overview



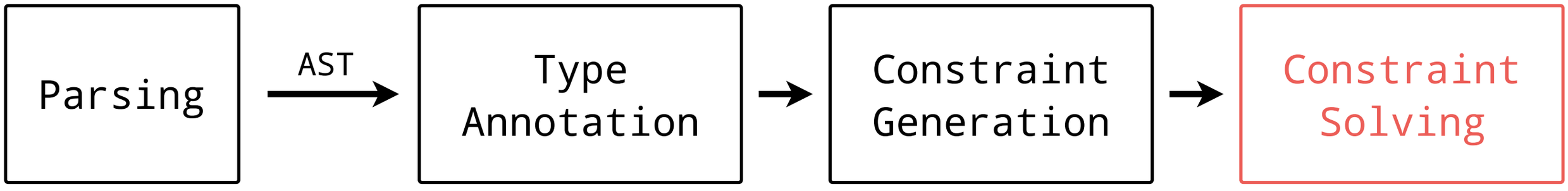
Constraint Set

```
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



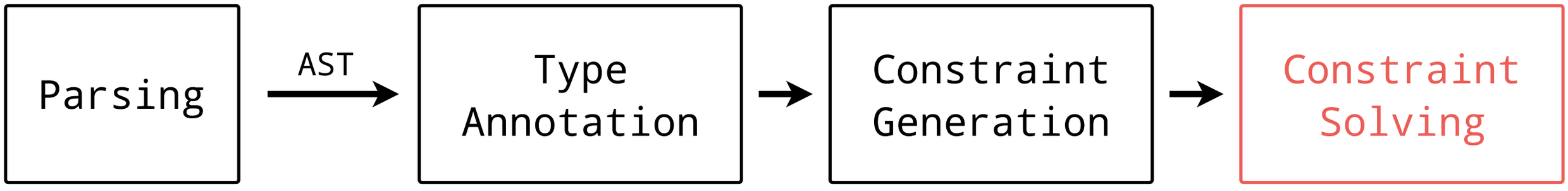
Constraint Set

```
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



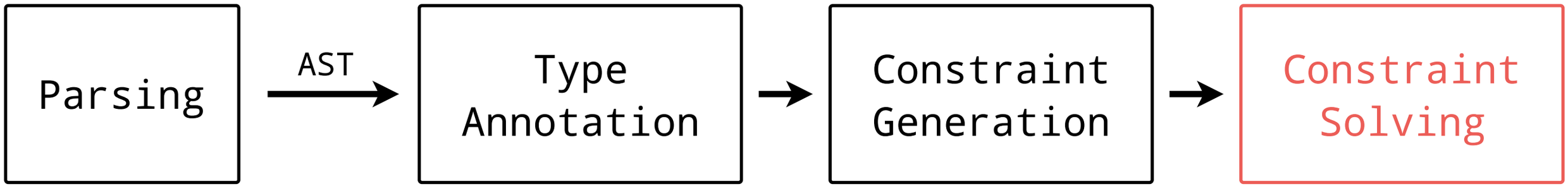
Constraint Set

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

Solution Map

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

Wand's Algorithm Overview



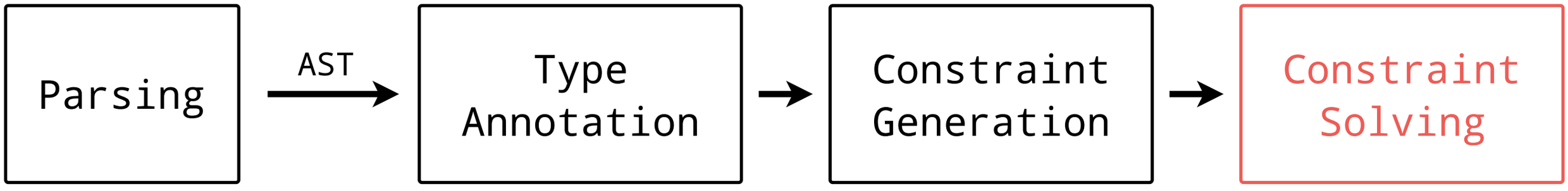
Constraint Set

```
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



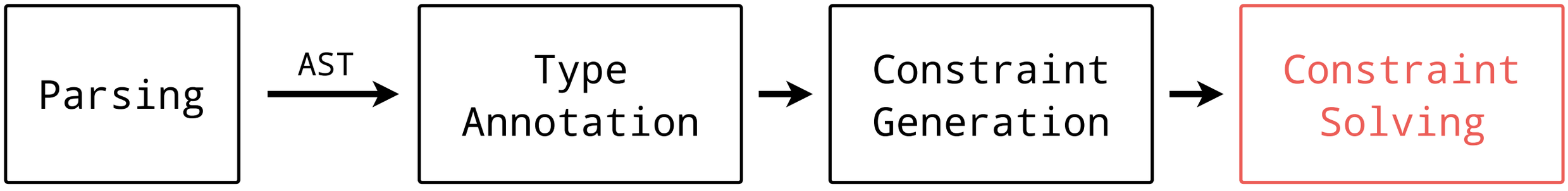
Constraint Set

```
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



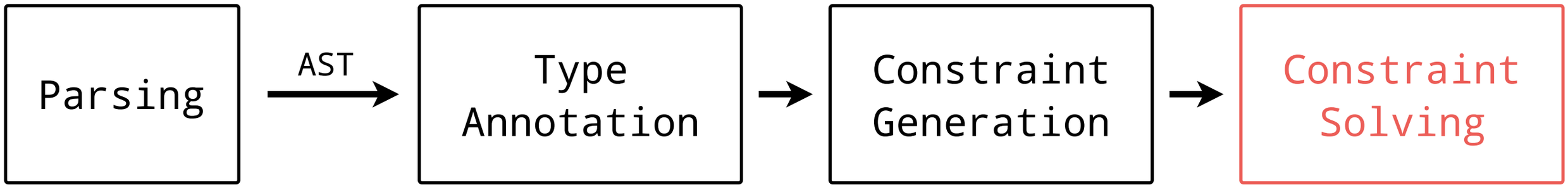
Constraint Set

```
t2 ≡ (int → bool) → int
```

Solution Map

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

Wand's Algorithm Overview

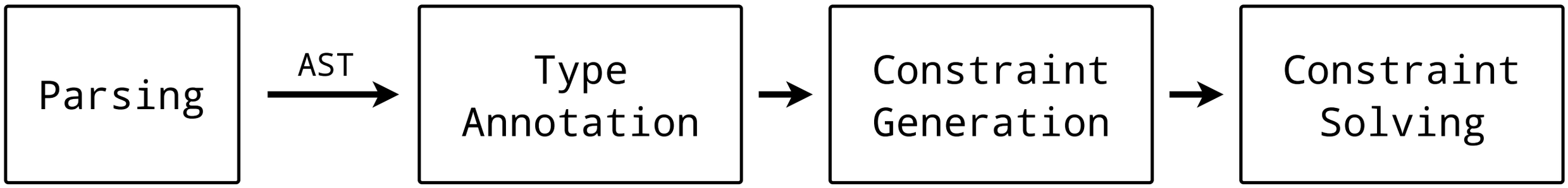


Constraint Set

Solution Map

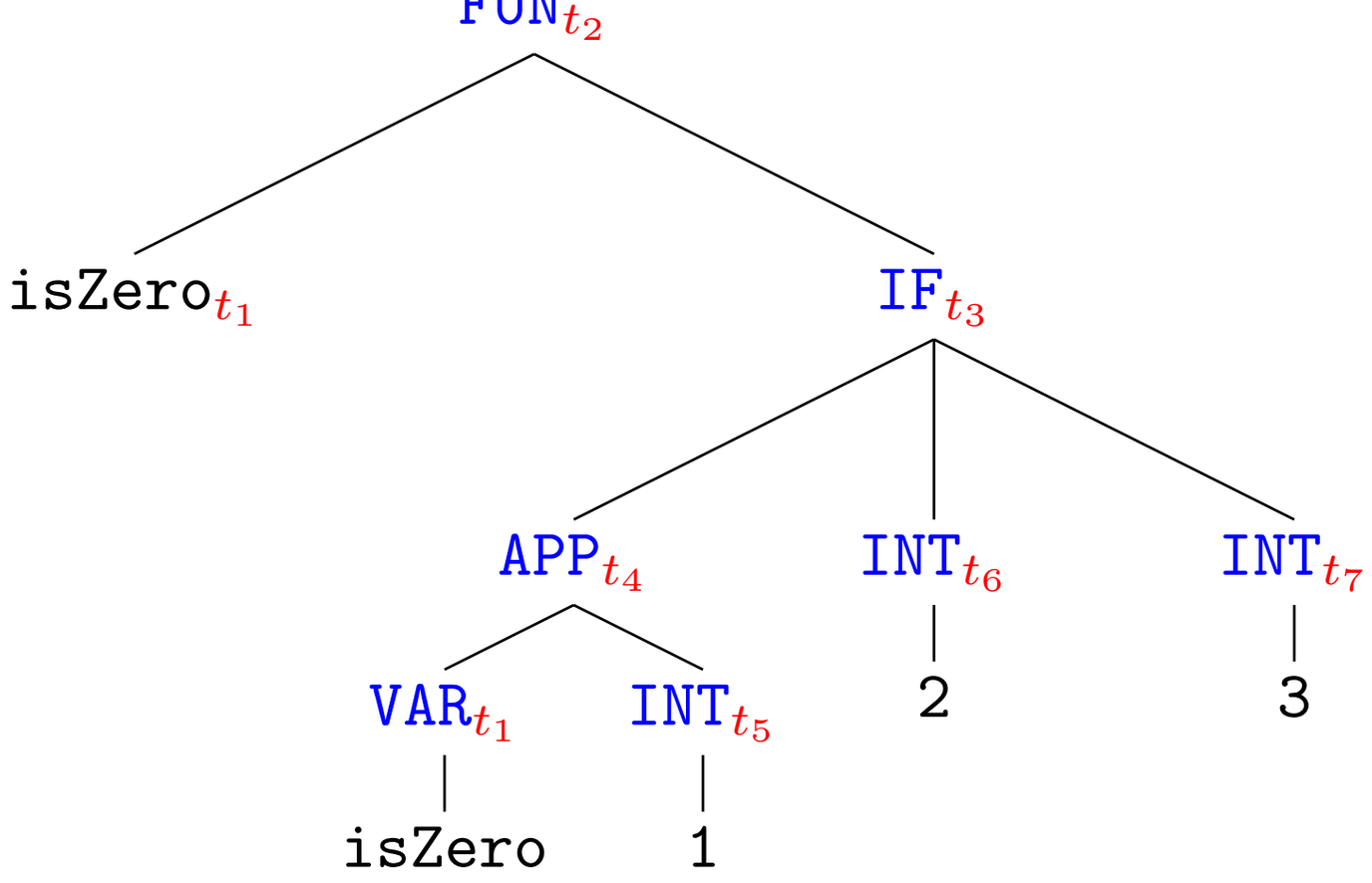
```
t1: int → bool
t5: int
t4: bool
t6: int
t7: int
t3: int
t2: (int → bool) → int
```

Wand's Algorithm Overview

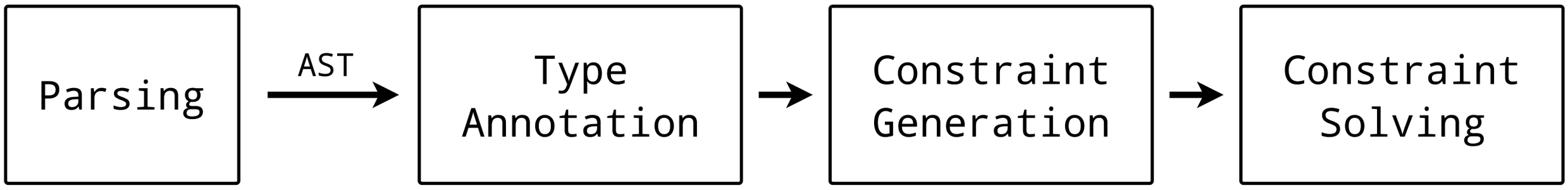


Solution Map

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 Inference Algorithms

- As I said, there are two main classes.
- Constraint-based: we've just seen one example — Wand's algorithm.
- Substitution-based: all phases are interleaved, e.g., Algorithm W.

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

github.com / igstan / typelevel-nyc-2017

Thank You!

Questions!