Semantic Analysis with Abstract Parsing

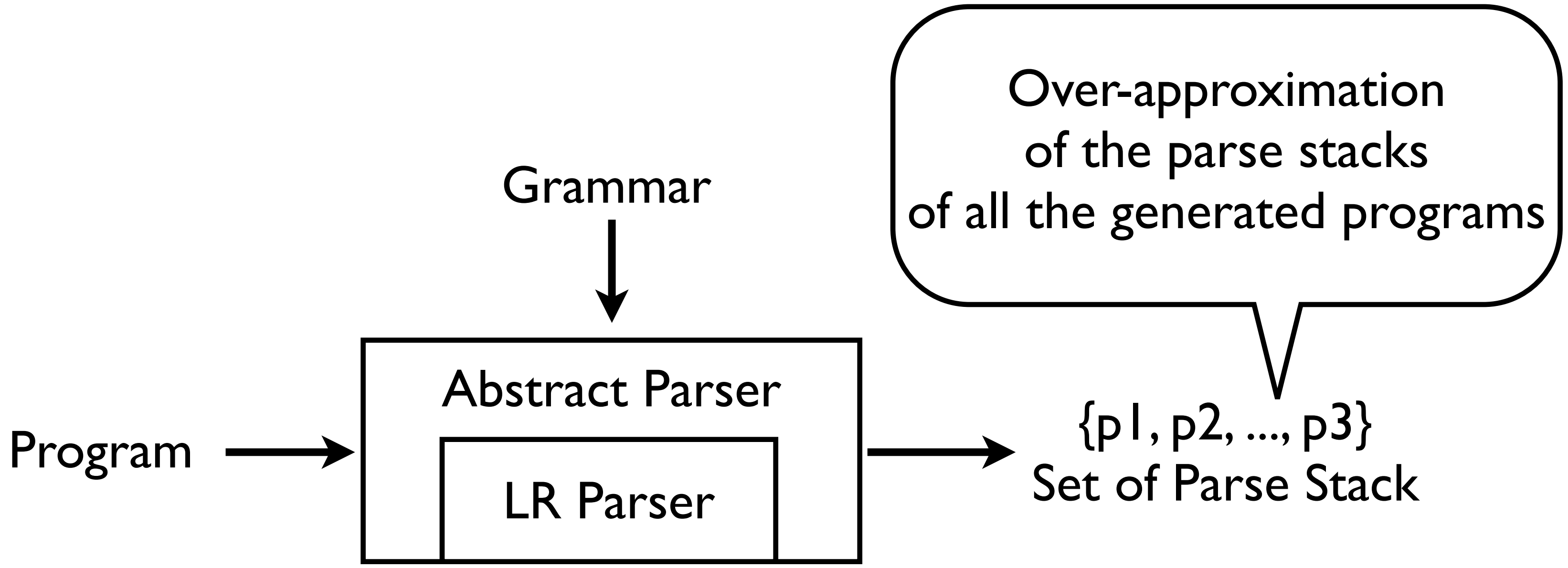
Soonho Kong
soon@ropas.snu.ac.kr

Wontae Choi
wtchoi@ropas.snu.ac.kr

2009/10/9
ROPAS Show & Tell

Abstract Parsing

- Powerful static string analysis technique by Doh, Kim, and Schmidt[1] (2009)
- Syntax checking of generated strings
- Use LR parser as a component

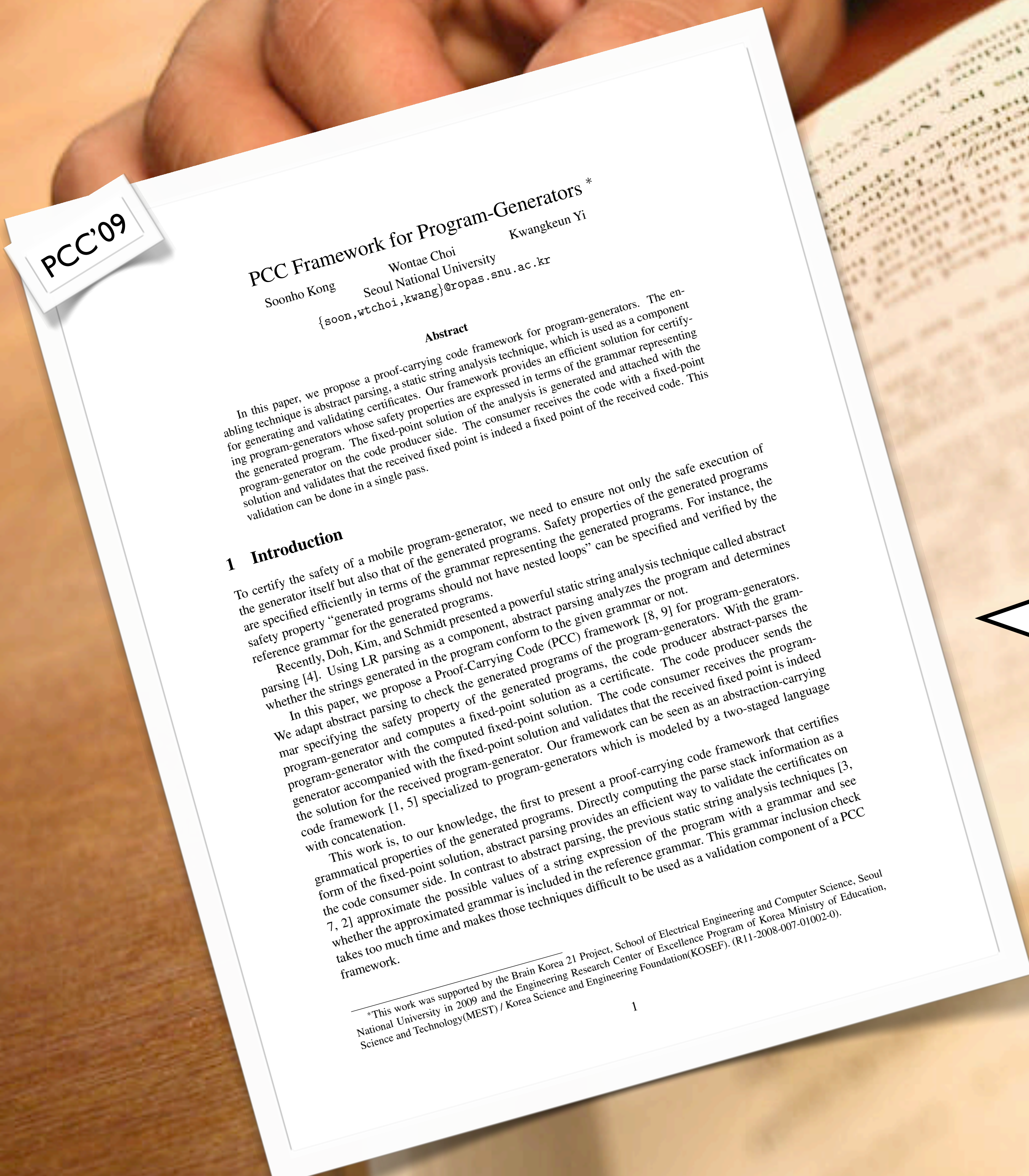


1. Kyung-Goo Doh, Hyunha Kim, and David Schmidt. *"Abstract parsing: static analysis of dynamically generated string output using LR-parsing technology."* In Proceeeding of the International Static Analysis Symposium, 2009.

Previous Work

1. Formalize/Generalize abstract parsing in the abstract interpretation framework
2. A correct and parameterized basis for its variants
3. For two-staged language with concatenation

Previous Work



PCC Framework for Program-Generators *

Soonho Kong Wontae Choi Kwangkeun Yi
Seoul National University
{soon,wtchoi,kwang}@ropas.snu.ac.kr

Abstract

In this paper, we propose a proof-carrying code framework for program-generators. The enabling technique is abstract parsing, a static string analysis technique, which is used as a component for generating and validating certificates. Our framework provides an efficient solution for certifying program-generators whose safety properties are expressed in terms of the grammar representing the generated program. The fixed-point solution of the analysis is generated and attached with the program-generator on the code producer side. The consumer receives the code with a fixed-point solution and validates that the received fixed point is indeed a fixed point of the received code. This validation can be done in a single pass.

1 Introduction

To certify the safety of a mobile program-generator, we need to ensure not only the safe execution of the generator itself but also that of the generated programs. Safety properties of the generated programs are specified efficiently in terms of the grammar representing the generated programs. For instance, the safety property "generated programs should not have nested loops" can be specified and verified by the reference grammar for the generated programs.

Recently, Doh, Kim, and Schmidt presented a powerful static string analysis technique called abstract parsing [4]. Using LR parsing as a component, abstract parsing analyzes the program and determines whether the strings generated in the program conform to the given grammar or not.

In this paper, we propose a Proof-Carrying Code (PCC) framework [8, 9] for program-generators. We adapt abstract parsing to check the generated programs of the program-generators. With the grammar specifying the safety property of the generated programs, the code producer abstract-parses the program-generator and computes a fixed-point solution as a certificate. The code producer sends the program-generator with the computed fixed-point solution. The code consumer receives the program-generator accompanied with the fixed-point solution and validates that the received fixed point is indeed the solution for the received program-generator. Our framework can be seen as an abstraction-carrying code framework [1, 5] specialized to program-generators which is modeled by a two-staged language with concatenation.

This work is, to our knowledge, the first to present a proof-carrying code framework that certifies grammatical properties of the generated programs. Directly computing the parse stack information as a form of the fixed-point solution, abstract parsing provides an efficient way to validate the certificates on the code consumer side. In contrast to abstract parsing, the previous static string analysis techniques [3, 7, 2] approximate the possible values of a string expression of the program with a grammar and see whether the approximated grammar is included in the reference grammar. This grammar inclusion check takes too much time and makes those techniques difficult to be used as a validation component of a PCC framework.

*This work was supported by the Brain Korea 21 Project, School of Electrical Engineering and Computer Science, Seoul National University in 2009 and the Engineering Research Center of Excellence Program of Korea Ministry of Education, Science and Technology(MEST) / Korea Science and Engineering Foundation(KOSEF). (R11-2008-007-01002-0).

1

Building a Proof-Carrying Code framework using abstract parsing

Motivation

Naoki Kobayashi

Q: Possible to check
semantic property?

A: Not now.
Only support syntactic property.

Soonho Kong

Motivation

Can we extend **abstract parsing** to check **semantic** property?


Soonho Kong


Wontae Choi

Language

- Two-staged language with concatenation

Syntax

$$e \in \mathit{Exp} ::= x \mid \texttt{let}\; x\; e_1\; e_2 \mid \texttt{or}\; e_1\; e_2 \mid \texttt{re}\; x\; e_1\; e_2\; e_3 \mid \text{`}f$$

$$f \in \mathit{Frag} ::= x \mid \texttt{let} \mid \texttt{or} \mid \texttt{re} \mid \texttt{(} \mid \texttt{)} \mid f_1 . f_2 \mid \texttt{,}e$$

Operational Semantics

$\boxed{\sigma \vdash^0 e \Rightarrow v}$

$$\sigma \vdash^0 x \Rightarrow \sigma(x) \quad \text{(variable)}$$

$$\frac{\sigma \vdash^0 e_1 \Rightarrow v \qquad \sigma[x \mapsto v] \vdash^0 e_2 \Rightarrow v'}{\sigma \vdash^0 \texttt{let}\; x\; e_1\; e_2 \Rightarrow v'} \quad \text{(let binding)}$$

$$\frac{\sigma \vdash^0 e_1 \Rightarrow v}{\sigma \vdash^0 \texttt{or}\; e_1\; e_2 \Rightarrow v} \qquad \frac{\sigma \vdash^0 e_2 \Rightarrow v}{\sigma \vdash^0 \texttt{or}\; e_1\; e_2 \Rightarrow v} \quad \text{(branch)}$$

$$\frac{\sigma \vdash^0 e_1 \Rightarrow v \qquad \sigma[x \mapsto v] \vdash^0 \mathit{loop}\; x\; e_2\; e_3 \Rightarrow v'}{\sigma \vdash^0 \texttt{re}\; x\; e_1\; e_3 \Rightarrow v'}$$

$$\frac{\sigma \vdash^0 e_2 \Rightarrow v \qquad \sigma[x \mapsto v] \vdash^0 \mathit{loop}\; x\; e_2\; e_3 \Rightarrow v'}{\sigma \vdash^0 \mathit{loop}\; x\; e_2\; e_3 \Rightarrow v'} \quad \text{(loop)}$$

$$\frac{\sigma \vdash^0 e_3 \Rightarrow v}{\sigma \vdash^0 \mathit{loop}\; x\; e_2\; e_3 \Rightarrow v}$$

$$\frac{\sigma \vdash^1 f \Rightarrow v}{\sigma \vdash^0 \text{`}f \Rightarrow v} \quad \text{(back quote)}$$

$\boxed{\sigma \vdash^1 f \Rightarrow v}$

$$\frac{}{\sigma \vdash^1 x \Rightarrow x} \qquad \frac{}{\sigma \vdash^1 \texttt{let} \Rightarrow \texttt{let}} \quad \text{(token)}$$

$$\frac{}{\sigma \vdash^1 \texttt{or} \Rightarrow \texttt{or}} \qquad \frac{}{\sigma \vdash^1 \texttt{re} \Rightarrow \texttt{re}}$$

$$\frac{}{\sigma \vdash^1 \texttt{(} \Rightarrow \texttt{(}} \qquad \frac{}{\sigma \vdash^1 \texttt{)} \Rightarrow \texttt{)}}$$

$$\frac{\sigma \vdash^1 f_1 \Rightarrow v_1 \qquad \sigma \vdash^1 f_2 \Rightarrow v_2}{\sigma \vdash^1 f_1 . f_2 \Rightarrow v_1 v_2} \quad \text{(concatenation)}$$

$$\frac{\sigma \vdash^0 e \Rightarrow v}{\sigma \vdash^1 \texttt{,}e \Rightarrow v} \quad \text{(comma)}$$

Language

Example

x is initialized with *a*

Loop body is not executed.

```
re x ‘a
  ‘or . ,x
  ‘,x . b
```

value is *a* b

```
=> a b
```

Language

Example

Loop body is executed once

```
re x ‘a
    ‘or  .  ,x
    ‘,x  .  b

=> or a b
```

x is initialized with *a*

x is or *a*

value is or *a* b

Language

Example

Loop body is executed twice

```
re x ‘a
   ‘or . ,x
   ‘,x . b

=> or or a b
```

x is initialized with a

x is or or a

value is or or a b

Language

Example

```
re x `a
   `or . ,x
   `,x . b
```

This program possibly generates one of the followings:

```
a b
or a b
or or a b
or or or a b
...
```

Only this one is syntactically correct

This program is possible to generate syntactically incorrect code.

Language

Example

```
re x ‘a
   ‘or . ,x
   ‘,x . b
```

This program possibly generates one of the following code:

```
 a  b
 or a  b
 or or a  b
 or or or a  b
...
```

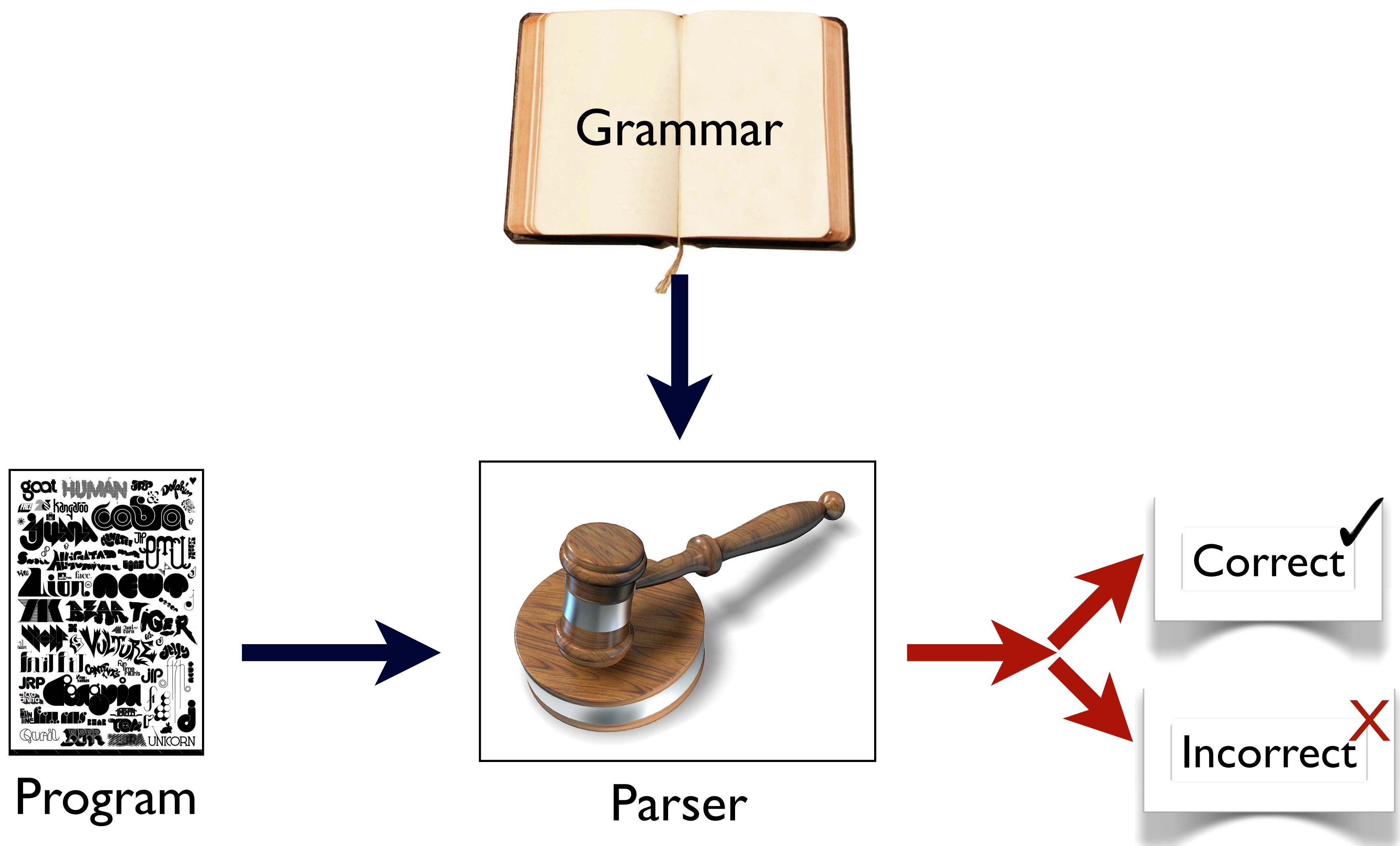
$$[\![\texttt{re}\ x\ \text{‘}a\ (\text{‘}\texttt{or}\ .\ ,x)\ (\text{‘},\texttt{x}\ .\ \texttt{b})]\!]^0\{\sigma_0\}$$

$$= \underline{\{\mathrm{a\,b, or\,a\,b, or\,or\,a\,b, or\,or\,or\,a\,b}, \ldots\}}$$

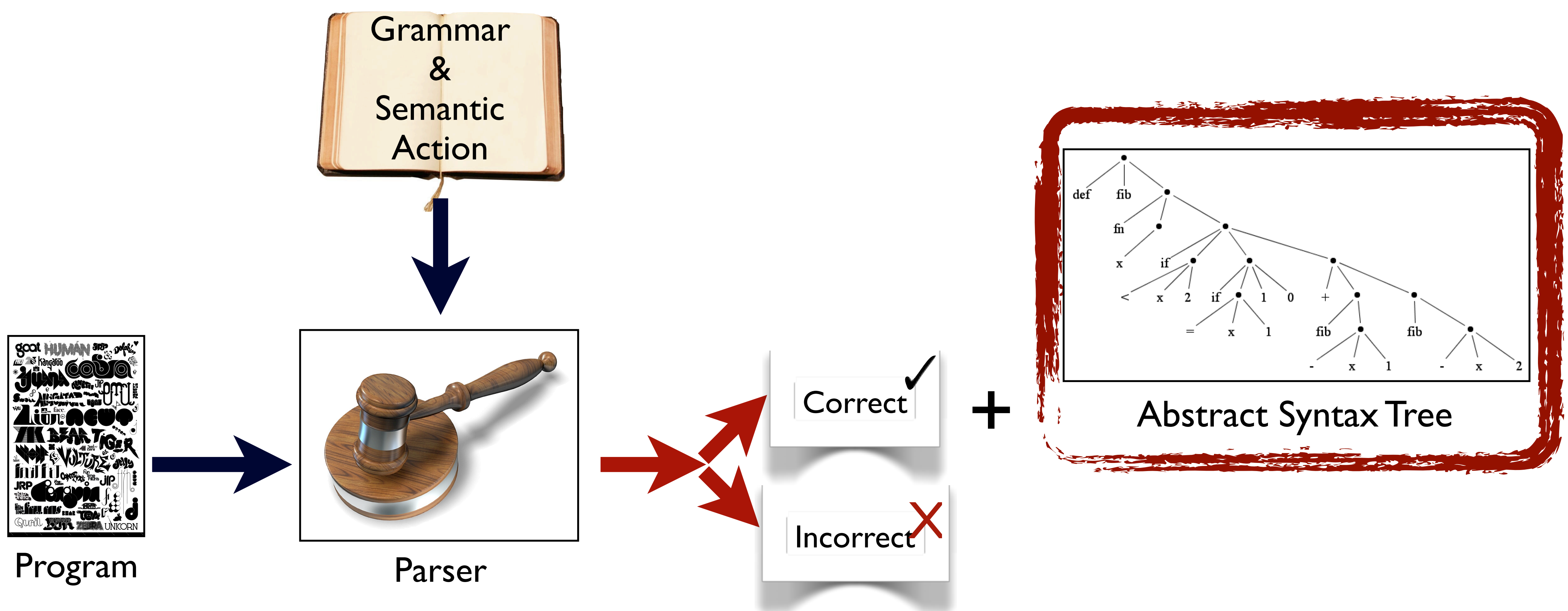
IDEA

“**Rethinking**
what we get by **parsing**!”

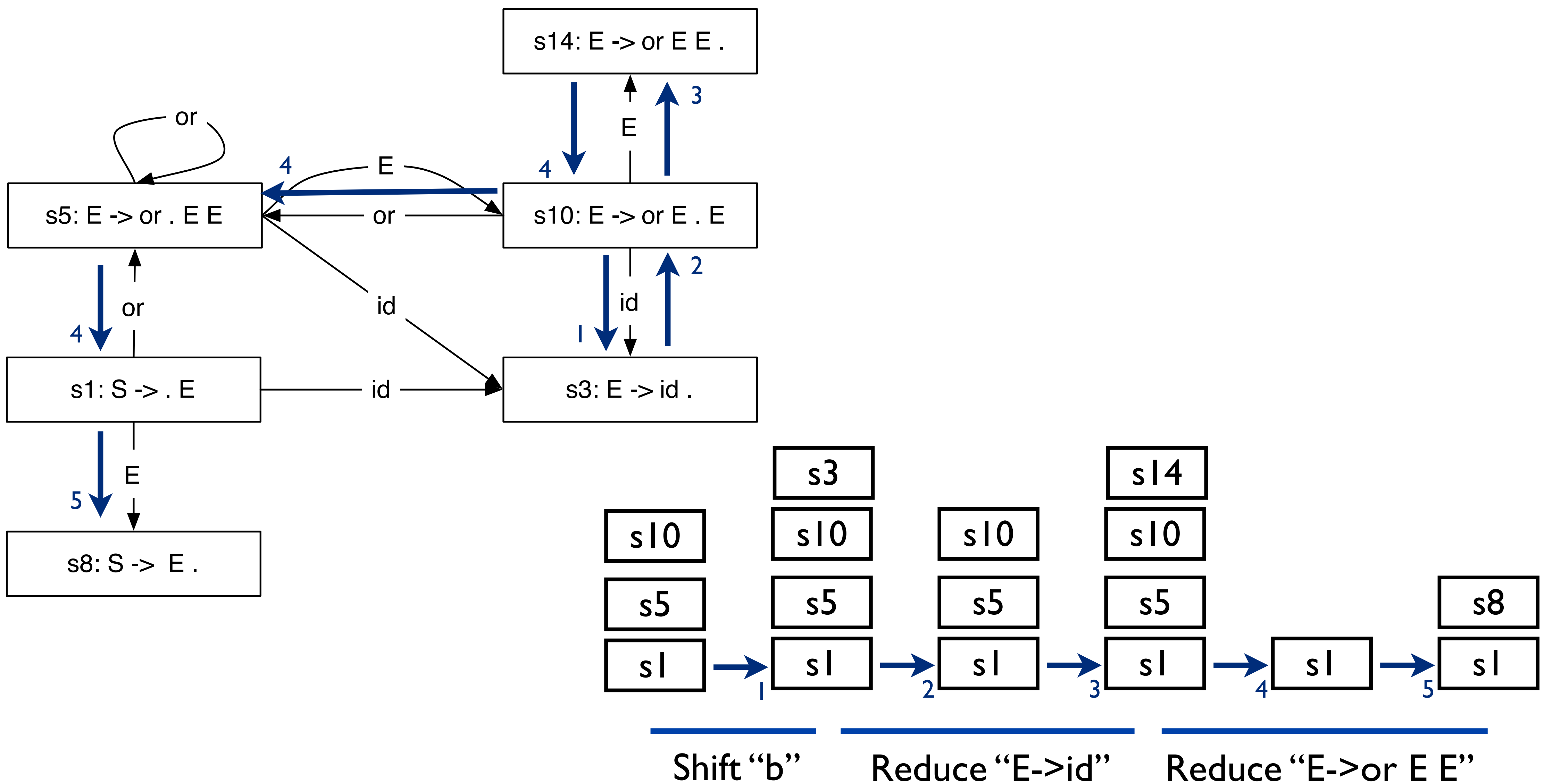
Parsing as a Decision Procedure



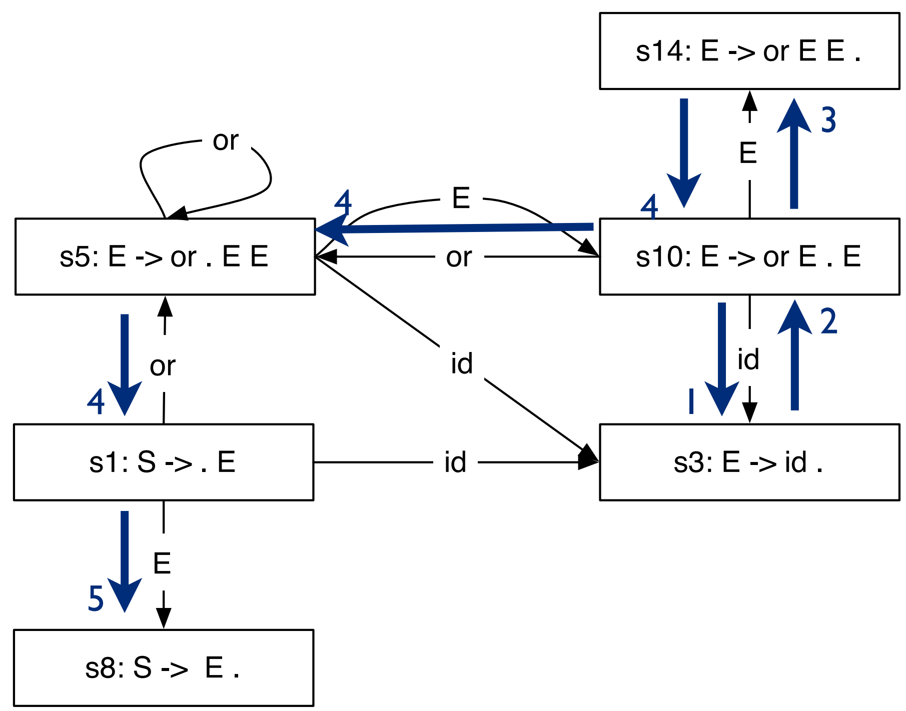
Parsing with Semantic Action



Example: Parsing "or a. b"

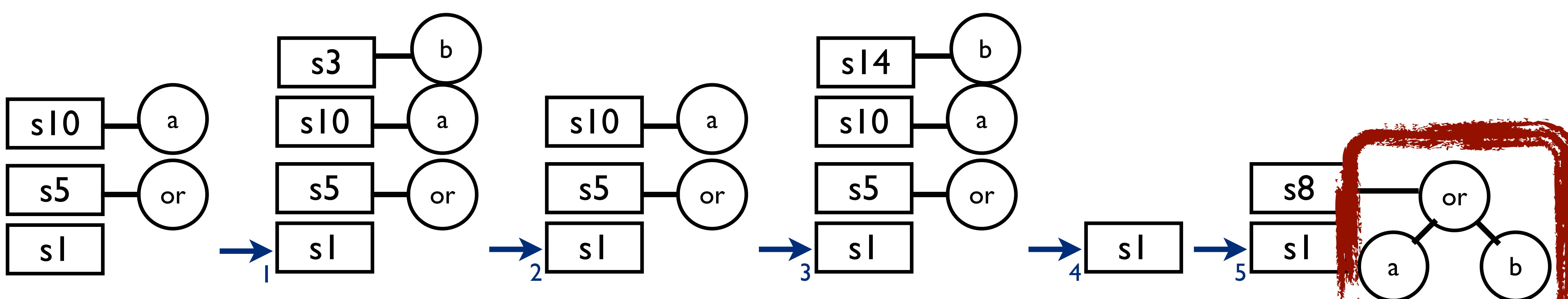


Example: Parsing “or a. b”



Semantic Action

```
E: OR e e { OR($2, $3) }
```



Shift “b” Reduce “E->id” Reduce “E->or E E”

Observation

- Semantic action allows us to construct AST
- Is is possible to construct other thing?

Observation

- Semantic action allows us to construct AST
- Is is possible to construct other thing?

Yes!

Observation

- Semantic action allows us to construct AST
- Is is possible to construct other thing?

Yes!

“as long as it is **compositionally** constructive”

Observation

- Semantic action allows us to construct AST
- Is is possible to construct other thing?

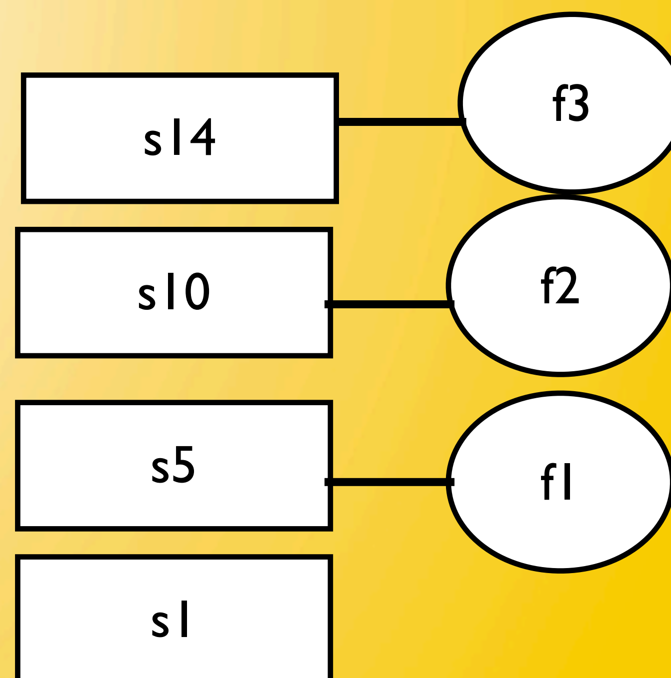
Yes!

“as long as i

Semantics!

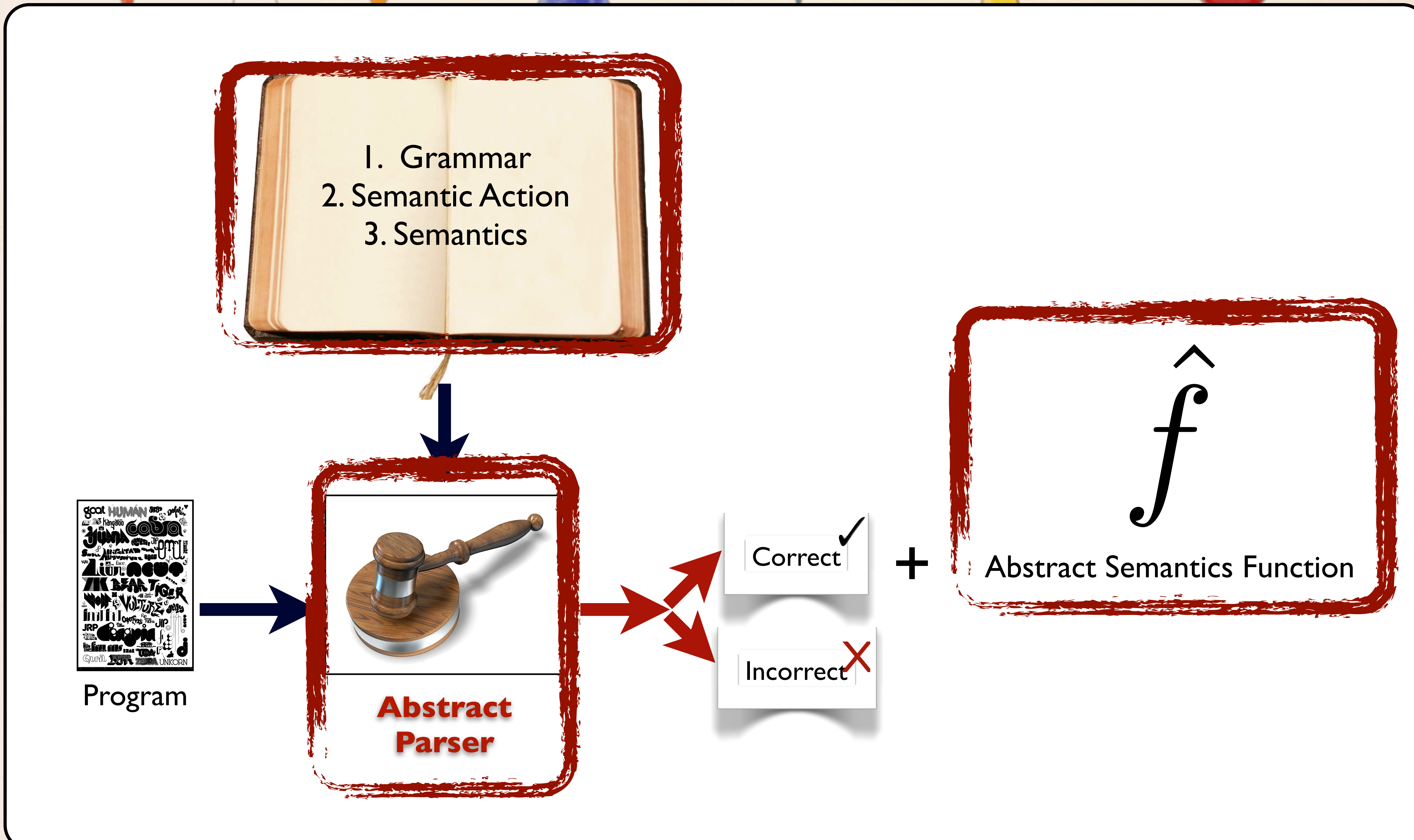
ly constructive”

IDEA

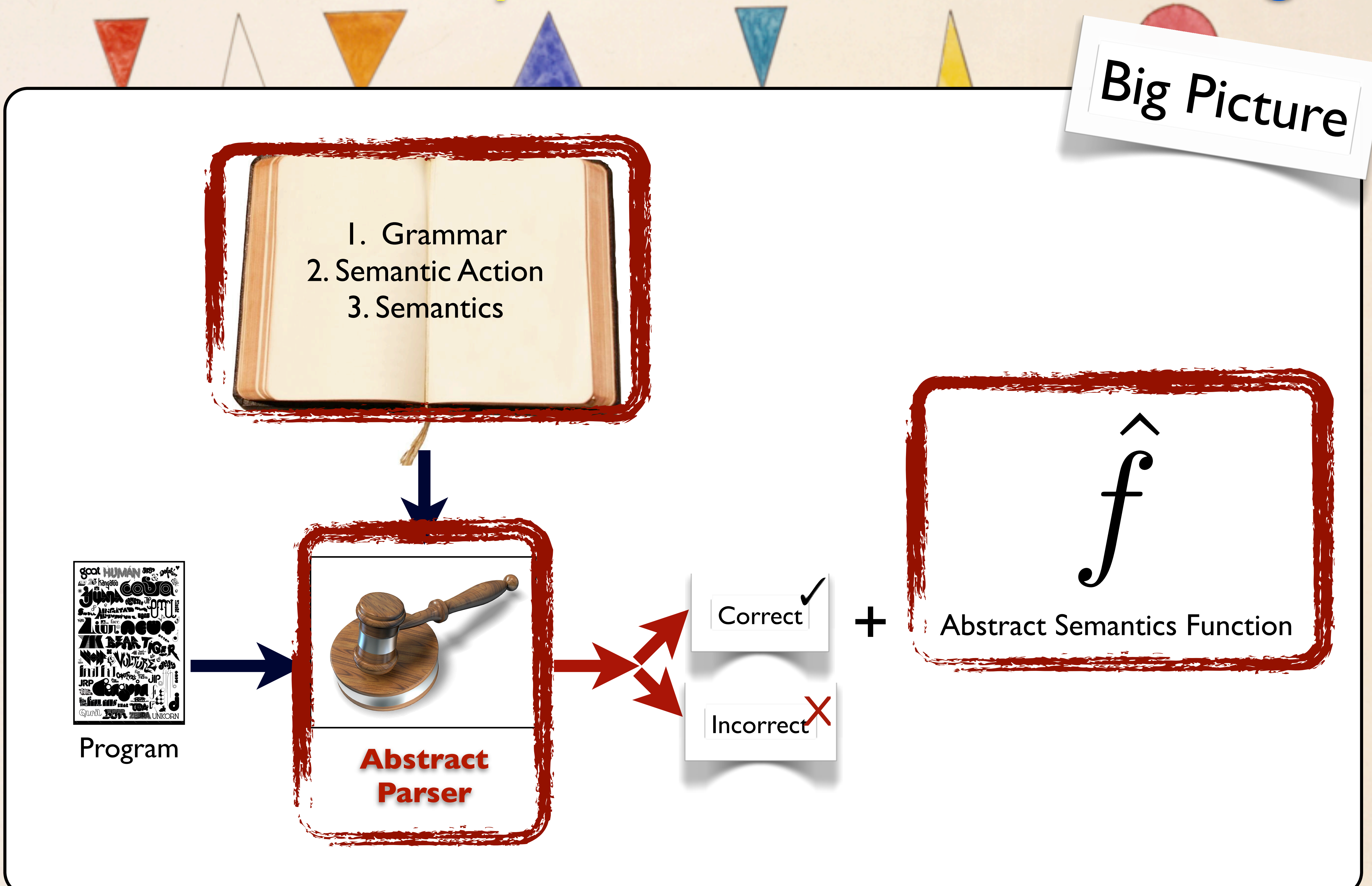


"**Augment** parse state with **semantic function**!"

Semantic Analysis with Abstract Parsing



Semantic Analysis with Abstract Parsing



Example

Unused Variable Analysis

$$
\begin{aligned}
\llbracket \texttt{or}\ e_1\ e_2 \rrbracket &= \llbracket e_1 \rrbracket \cap \llbracket e_2 \rrbracket \\
\llbracket \texttt{let}\ x\ e_1\ e_2 \rrbracket &= \llbracket e_1 \rrbracket \cap \llbracket e_2 \rrbracket \\
\llbracket \texttt{x} \rrbracket &= U - \{x\} \\
\llbracket \texttt{re}\ \texttt{x}\ e_1\ e_2\ e_3 \rrbracket &= \llbracket e_1 \rrbracket \cap \llbracket e_2 \rrbracket \cap \llbracket e_3 \rrbracket
\end{aligned}
$$

Example

Unused Variable Analysis

```
let p1 ‘let . x . a
let p2 ‘let . y . b
       ‘,p1 . ,p2 . or . x . x


=> let x a
   let y b
       or  x  x
```

Example

Unused Variable Analysis

```
let p1 ‘let . x . a
let p2 ‘let . y . b
       ‘,p1 . ,p2 . or . x . x

=> let x a
   let y b
       or  x  x
```

$$U = \{\mathtt{x}, \mathtt{y}, \mathtt{a}, \mathtt{b}\}$$

$$\llbracket \mathtt{or}\ e_1\ e_2 \rrbracket = \llbracket e_1 \rrbracket \cap \llbracket e_2 \rrbracket$$
$$\llbracket \mathtt{let}\ x\ e_1\ e_2 \rrbracket = \llbracket e_1 \rrbracket \cap \llbracket e_2 \rrbracket$$
$$\llbracket \mathtt{x} \rrbracket = U - \{x\}$$
$$\llbracket \mathtt{re}\ \mathtt{x}\ e_1\ e_2\ e_3 \rrbracket = \llbracket e_1 \rrbracket \cap \llbracket e_2 \rrbracket \cap \llbracket e_3 \rrbracket$$

Conclusion

Naoki Kobayashi

Q: Possible to check
semantic property?

Soonho Kong

Wontae Choi

Conclusion

Naoki Kobayashi

Q: Possible to check semantic property?

A: Yes, we can!

Soonho Kong

Wontae Choi

Conclusion

- **Semantic analysis** is possible with **abstract parsing**
- By augmenting parse stack
  - Semantic function for each parse state
- Work in Progress
  - Formalize the idea
  - Find out more examples


Thank you