Extended Abstract Parsing:
Constraint-based Analysis of Two-staged Programs†

Soonho Kong
soon@ropas.snu.ac.kr

23 Apr 2010
ROPAS Show & Tell

† This is a joint-work with Wontae Choi and Prof. Kwangkeun Yi

Chapter 1

Review: Abstract Parsing

Two-Staged Program

Two-Staged
Program

$Eval : Pgm \rightarrow Code$

1+2

Code

Two-Staged Program

Two-Staged
Program

$Eval : Pgm \rightarrow Code$

1+

Code

Two-Staged Program

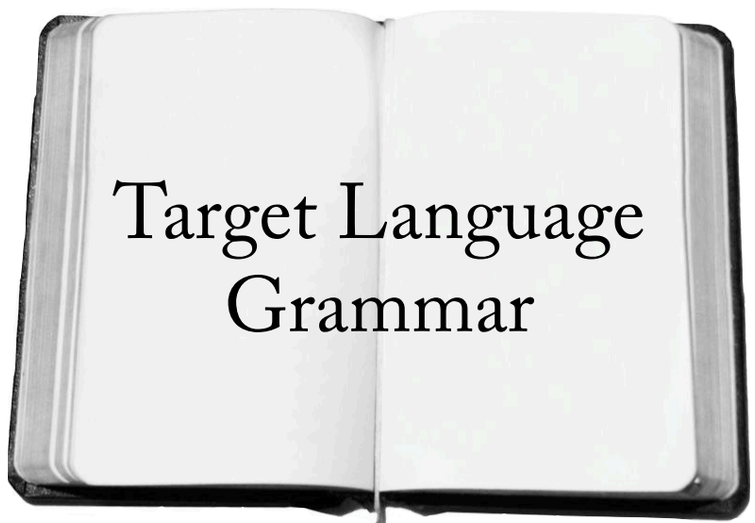
Two-Staged
Program

$Eval : Pgm \rightarrow Code$

X

1+

Code

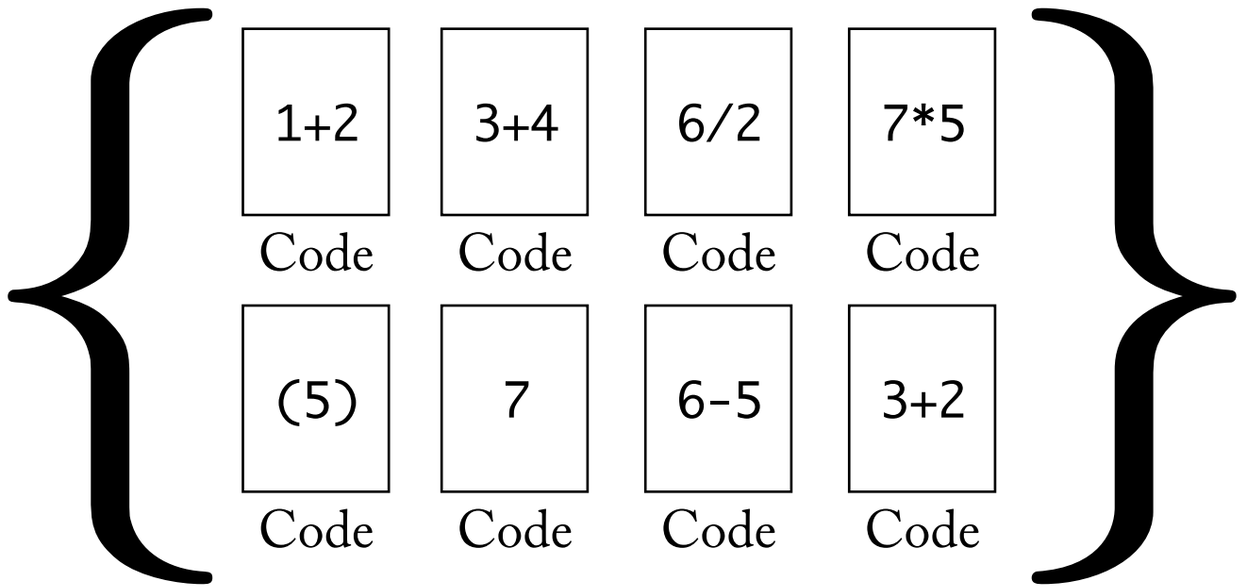


Goal:

Statically Checking the Syntax of Generated Code

Two-Staged Program

$$CollectingEval : Pgm \rightarrow 2^{Code}$$

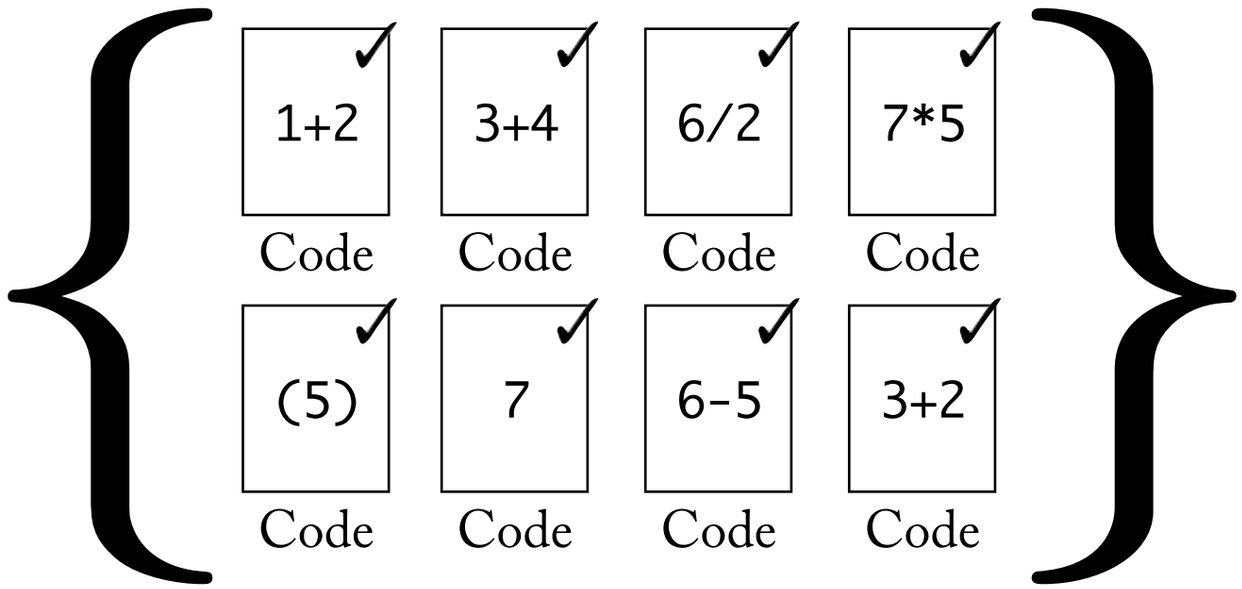


Goal:

Statically Checking the Syntax of Generated Code

Two-Staged Program

$$CollectingEval : Pgm \rightarrow 2^{Code}$$



Goal:

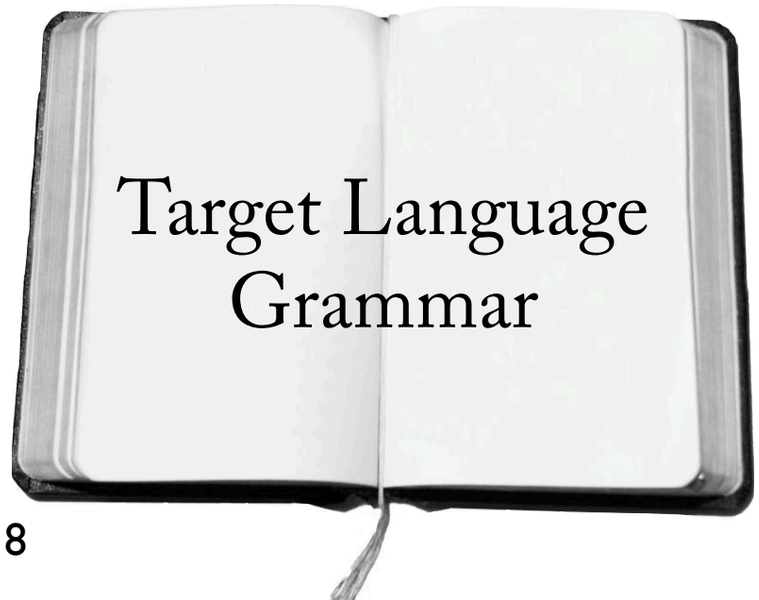
Statically Checking the Syntax of Generated Code

✓Safe!

Two-Staged Program

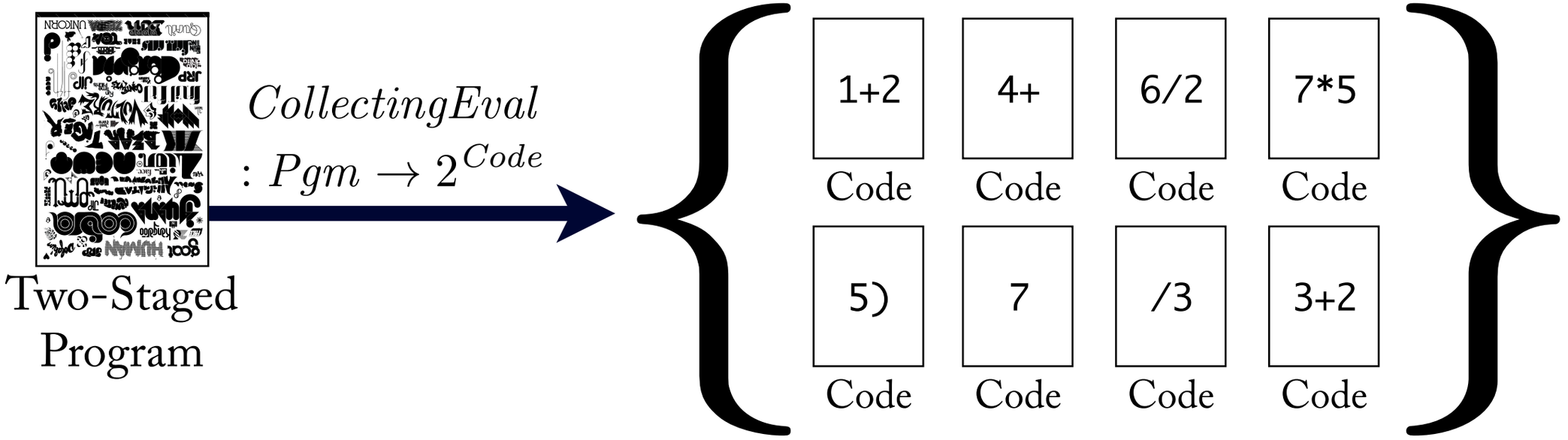
$$CollectingEval : Pgm \rightarrow 2^{Code}$$

{ 1+2 ✓ Code, 3+4 ✓ Code, 6/2 ✓ Code, 7*5 ✓ Code,
(5) ✓ Code, 7 ✓ Code, 6-5 ✓ Code, 3+2 ✓ Code }



Goal:

Statically Checking the Syntax of Generated Code

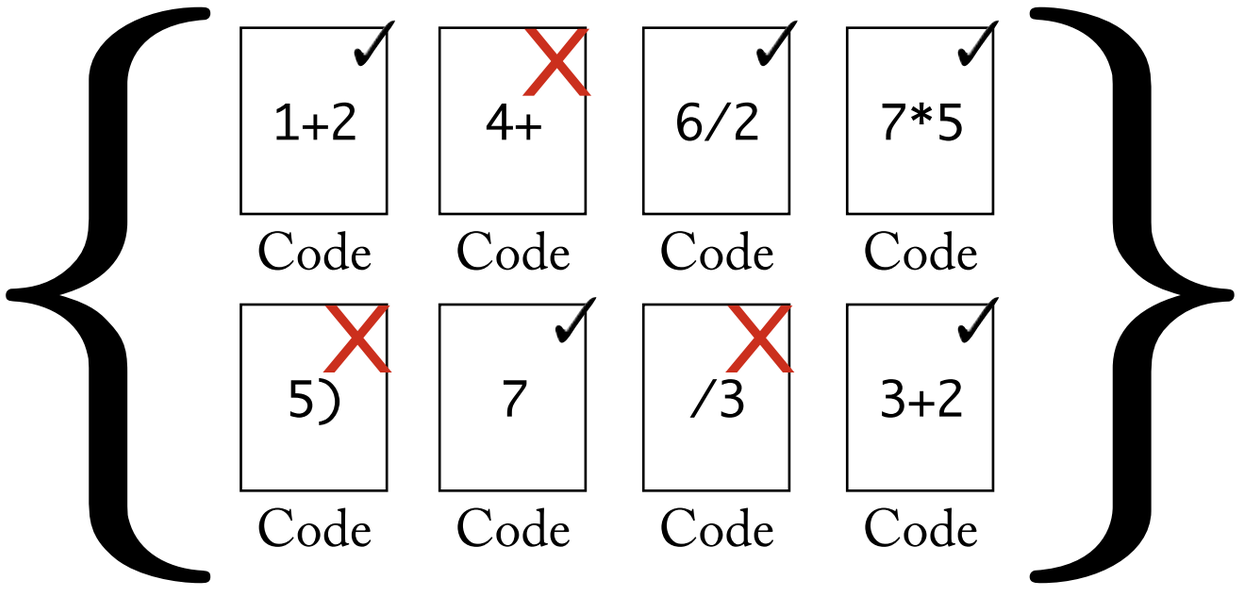


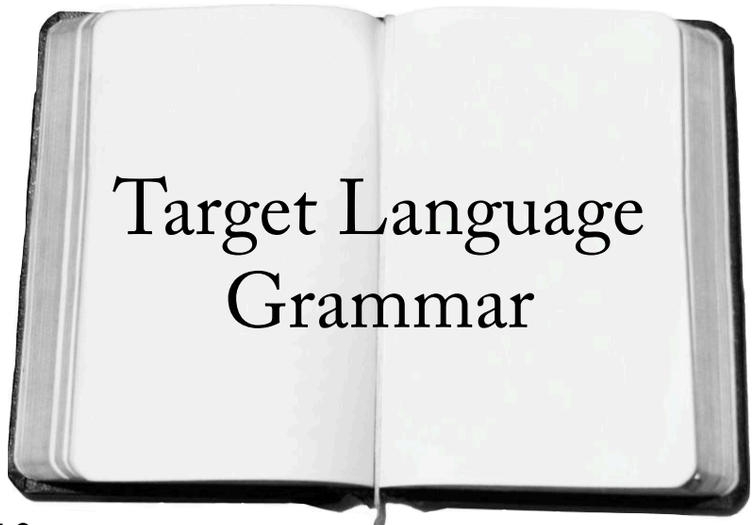
Goal:

Statically Checking the Syntax of Generated Code

Two-Staged Program

$$CollectingEval : Pgm \rightarrow 2^{Code}$$





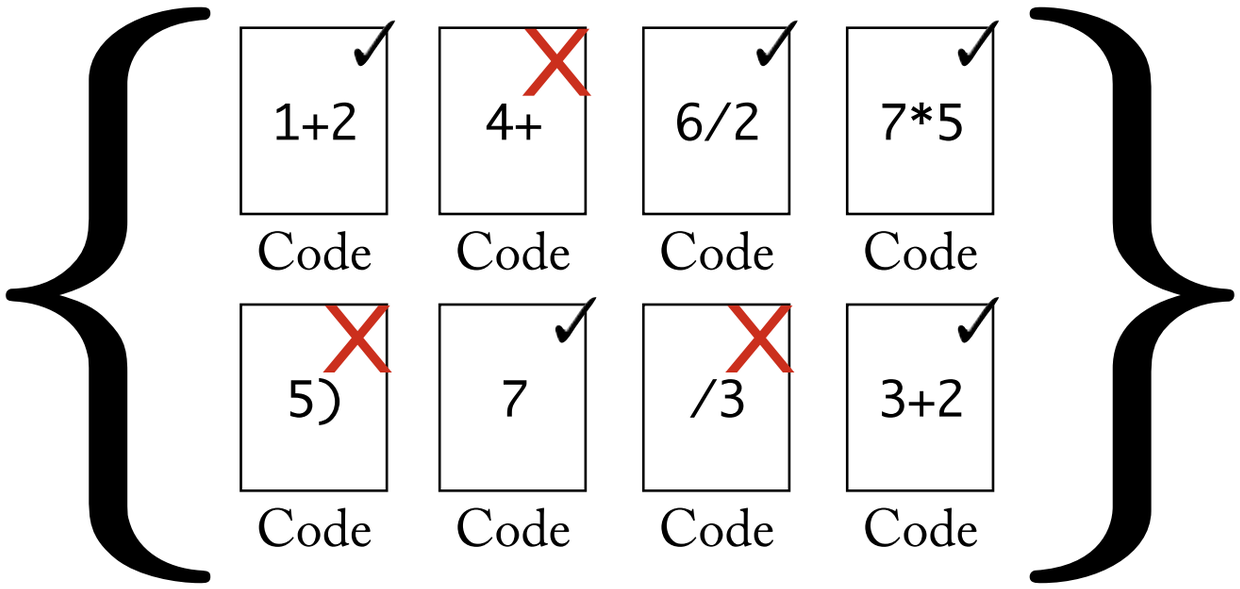
Goal:

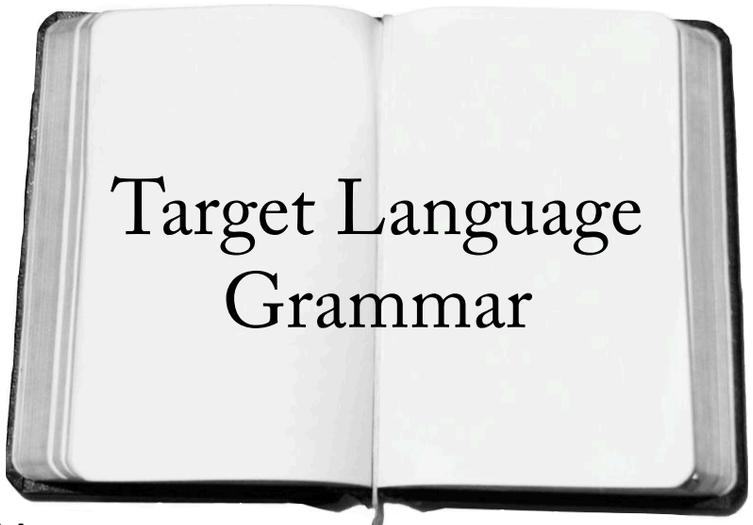
Statically Checking the Syntax of Generated Code

✗ Unsafe!

Two-Staged Program

$$CollectingEval : Pgm \rightarrow 2^{Code}$$





Problem:

CollectingEval is Infeasible

?

Two-Staged Program

$$CollectingEval : Pgm \rightarrow 2^{Code}$$

{ 1+2 Code, 4+ Code, 6/2 Code, 7*5 Code, 5) Code, 7 Code, • • • • • }

Solution:

Use Abstraction

Two-Staged
Program

$$\widehat{Eval} : Pgm \to \widehat{D}$$

d

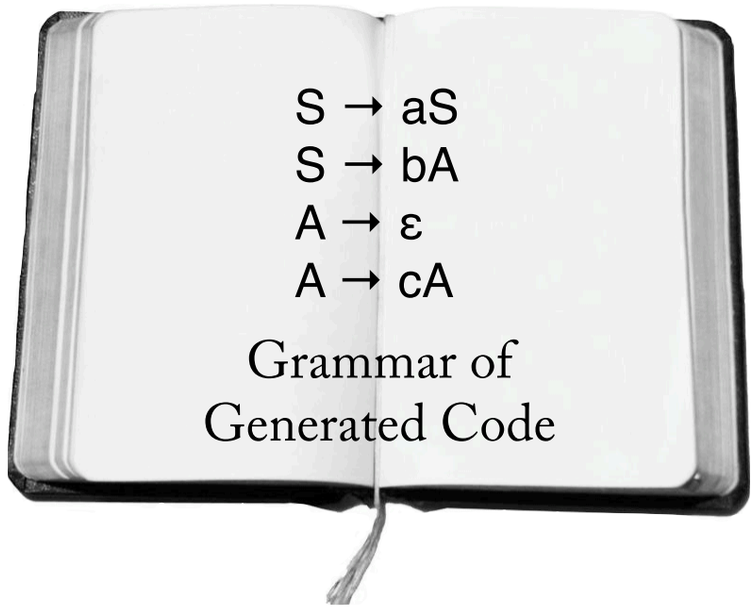
Abstraction
for Set of Code

Previous Works:

Abstraction into Grammar

Two-Staged Program

$\widehat{Eval}$
$: Pgm \to \widehat{D}$



$\ln \circ \ulcorner$

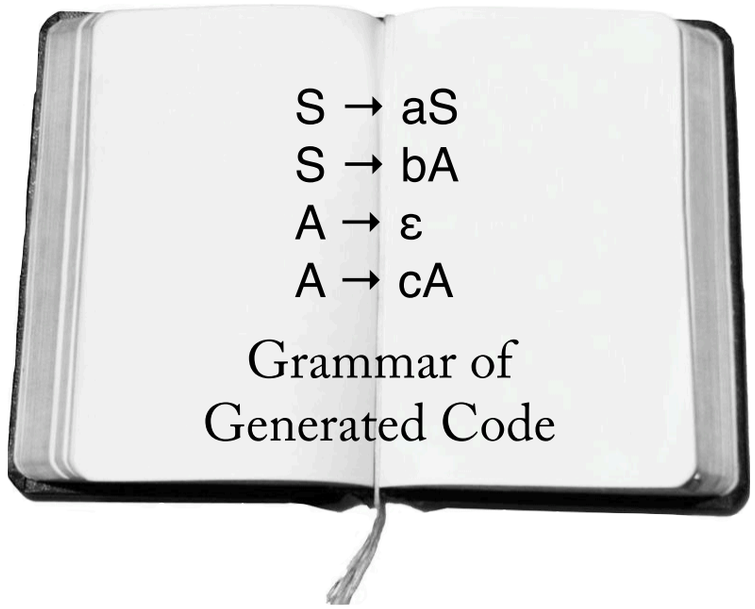
Previous Works:

Abstraction into Grammar

Two-Staged Program

$$\widehat{Eval} : Pgm \to \widehat{D}$$

CFG



$\in \cap \cdot \supset$

Previous Works:

Abstraction into Grammar

$$\text{CFG} \overset{?}{\subseteq} \text{CFG}$$

Undecidable!

Previous Works:

Abstraction into Grammar

$$\text{RG} \overset{?}{\subset} \text{CFG}$$

Decidable

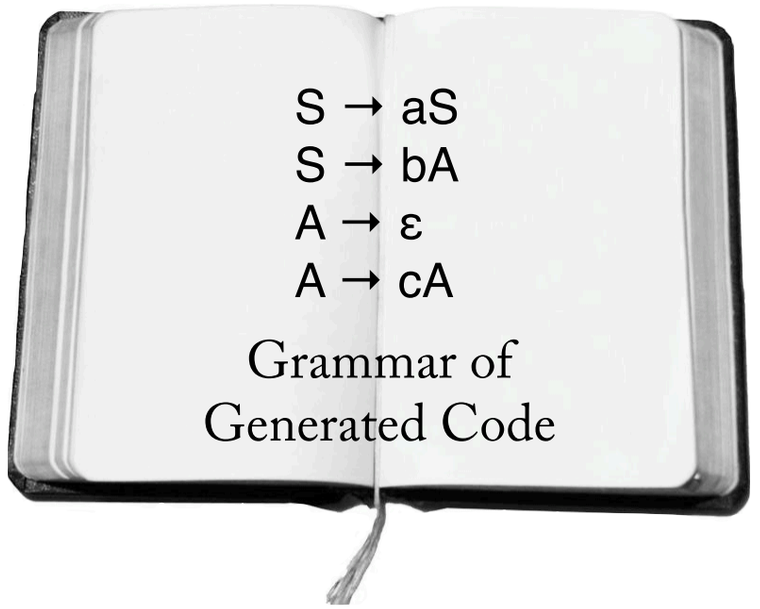
Previous Works:

Aske Simon Christensen, Anders Mller, and Michael I. Schwartzbach.
"Precise analysis of string expressions." SAS'03

Two-Staged
Program

$\widehat{Eval}$
$: Pgm \to \widehat{D}$

CFG => RG



$\ln \cdot \circ$

Previous Works:

Yasuhiko Minamide. "*Static approximation of dynamically generated web pages*." WWW'05

Two-Staged Program

$$\widehat{Eval} : Pgm \to \widehat{D}$$

CFG

S → aS
S → bA
A → ε
A → cA

Grammar of Generated Code

$\cap$

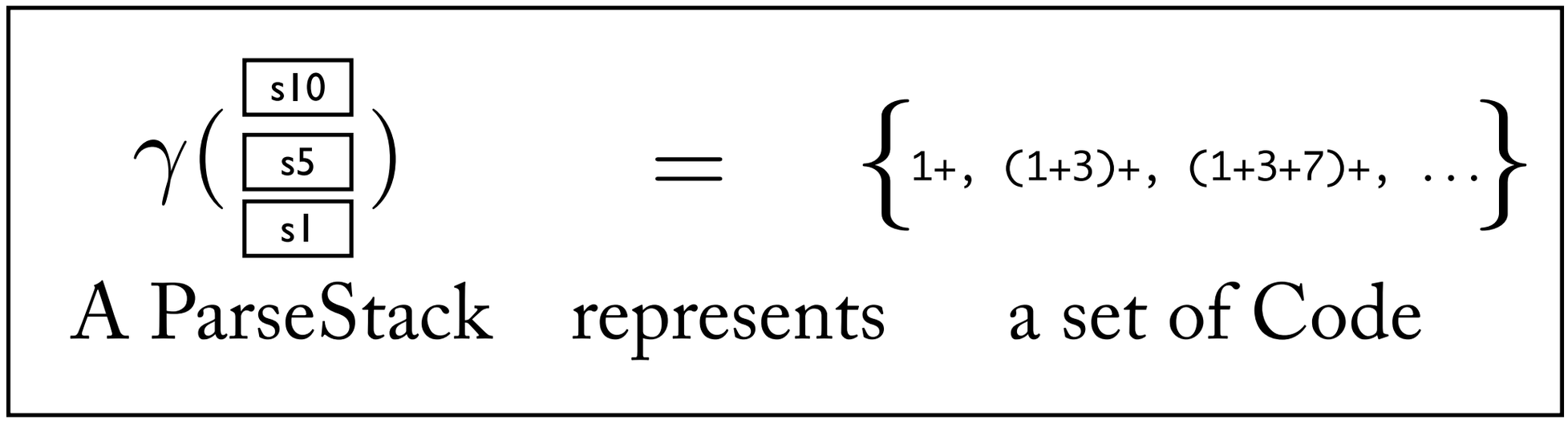
CFG=>RG

Target Language Grammar

Abstract Parsing

Abstraction using ParseStack

Key idea #1

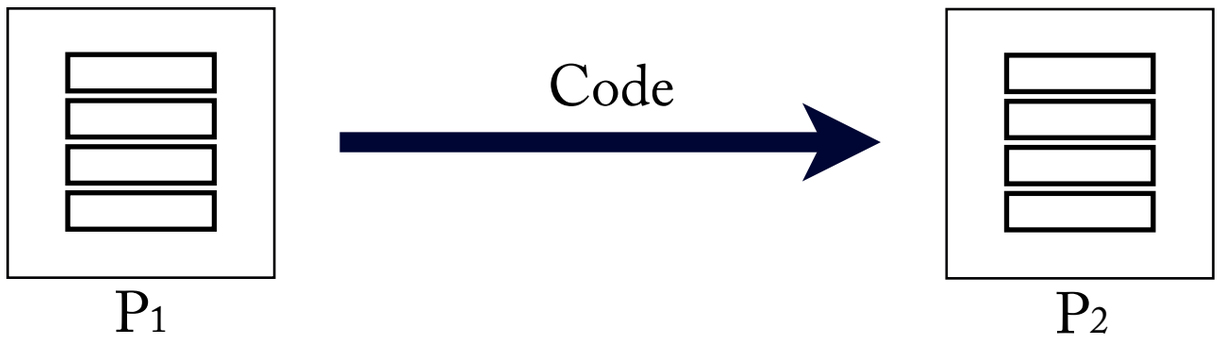


A ParseStack represents a set of Code

Abstract Parsing

Abstraction using ParseStack

Key idea #2



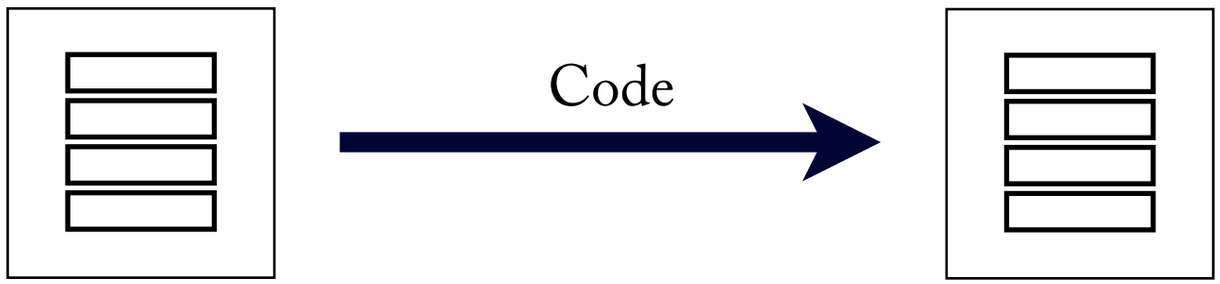
Code as ParseStack Transition Function

$$Code \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} P \to P$$

Abstract Parsing

Abstraction using ParseStack

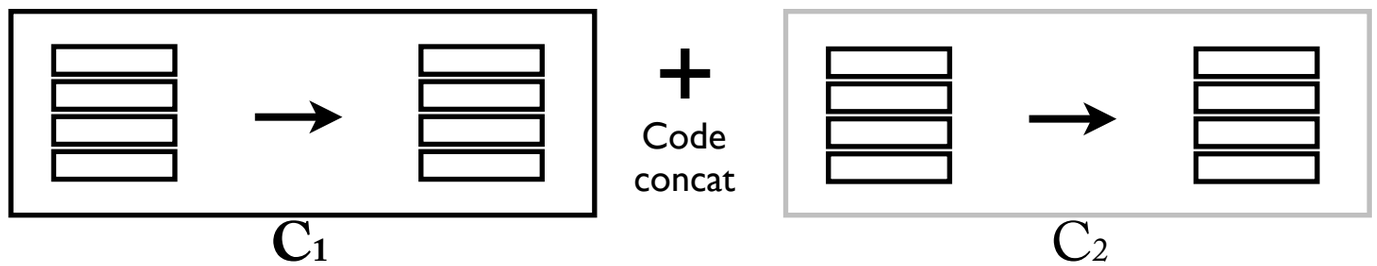
Key idea #2



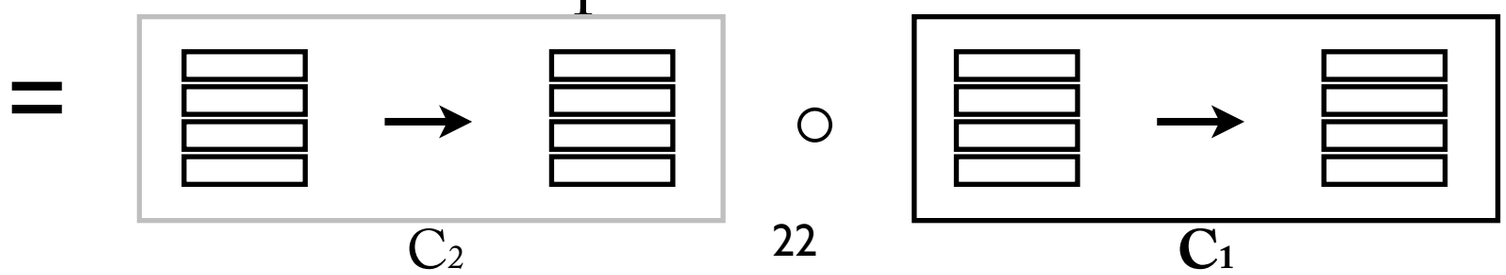
Code as ParseStack Transition Function

$$Code \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} P \to P$$

Code concatenation



Function composition.



Abstract Parsing

Abstraction using ParseStack

Abstraction Step

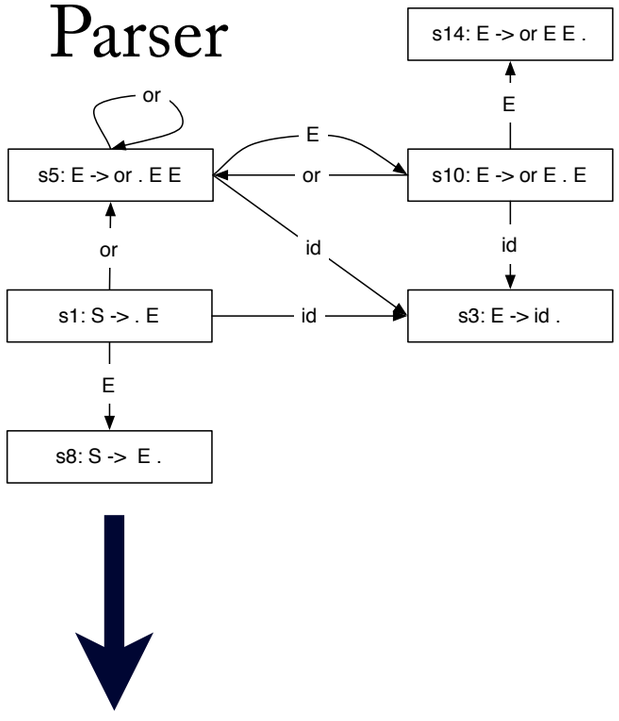
$$2^{Code} \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \boxed{2^{P \to P}} \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \boxed{2^{P} \to 2^{P}}$$

Abstract Parsing

$$AbstractParsing$$
$$: Pgm \to (2^{P} \to 2^{P})$$

Abstract Parsing

Abstraction into ParseStack



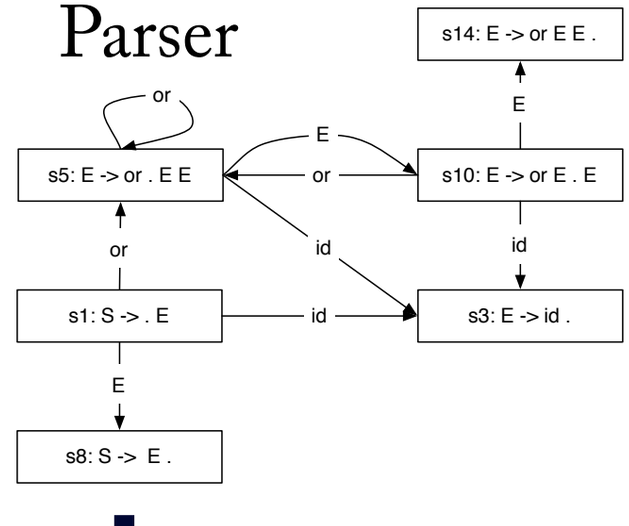
Two-Staged
Program

$$AbstractParsing : Pgm \rightarrow (2^P \rightarrow 2^P)$$

$$f$$

Abstract Parsing

Abstraction into ParseStack



Two-Staged
Program

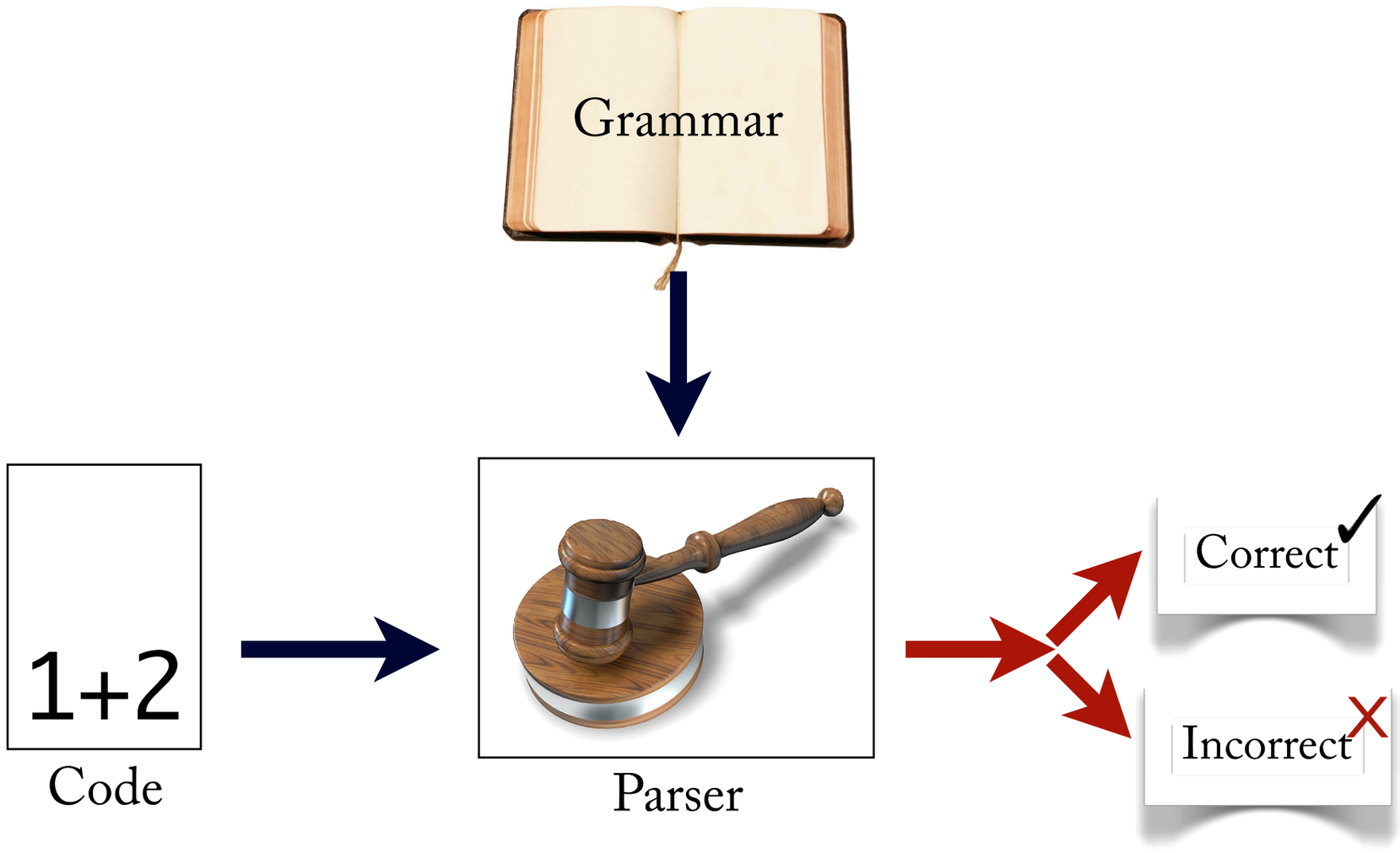
$$AbstractParsing : Pgm \rightarrow (2^P \rightarrow 2^P)$$

$$f\{\underset{\text{Initial ParseStack}}{\boxed{s1}}\} = \{\underset{\text{Accepting ParseStack}}{\boxed{s8}\ \boxed{s1}\ \checkmark}, \underset{\text{Non-Accepting ParseStack}}{\boxed{s10}\ \boxed{s5}\ \boxed{s1}\ \times}\}$$

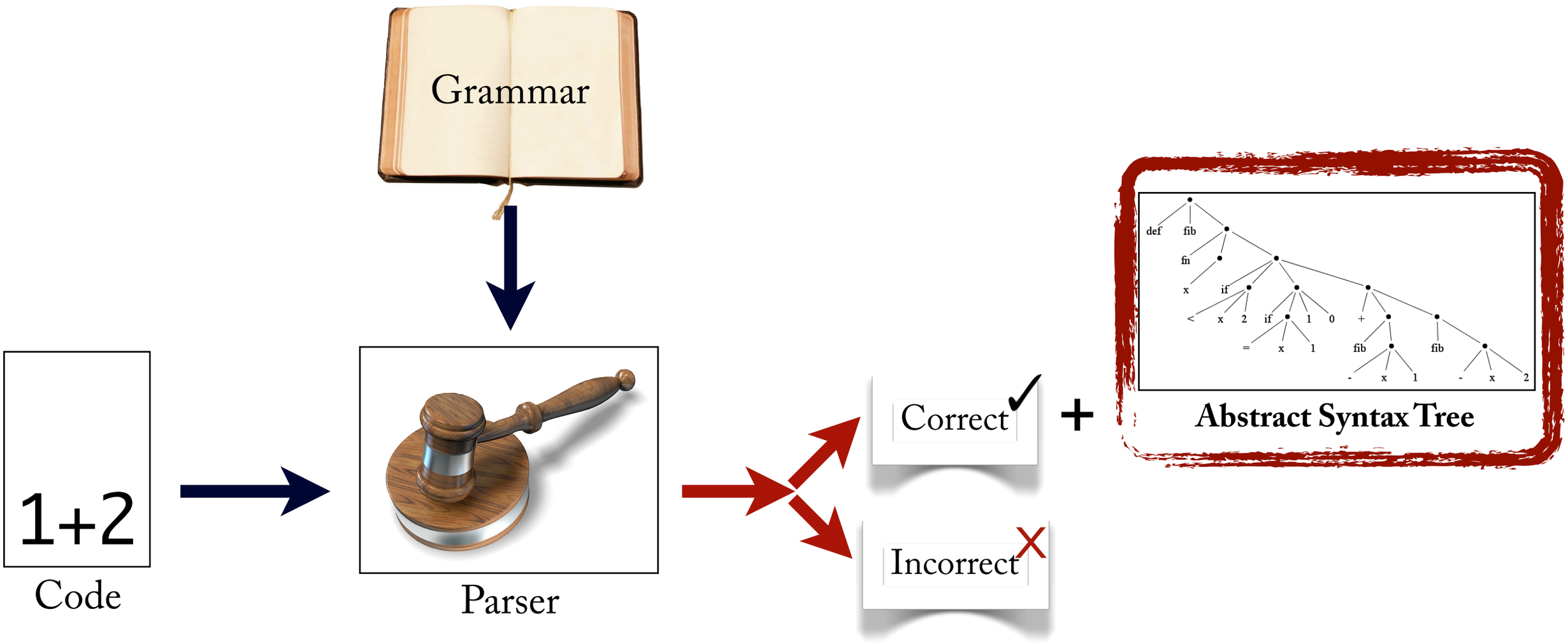
Chapter II

Extended Abstract Parsing:
Constraint-based Analysis of Generated Code

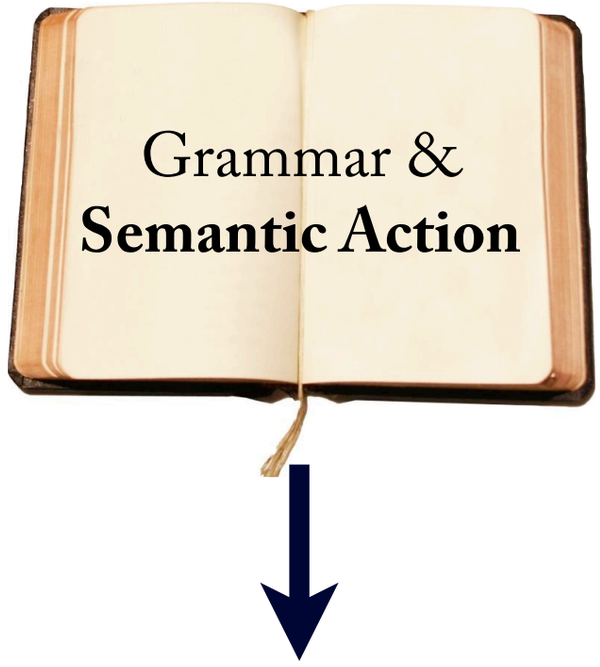
Parsing in Abstract Parsing: “Decision Procedure”



Parsing in Modern Compiler: "AST Builder"



Extended Abstract Parsing (Big Picture)



1+2

Code

Extended Abstract Parser

Correct ✓ + $\{C_1, C_2, \ldots, C_n\}$

Incorrect ✗

Set of Constraints

Solve!

Abstraction Plan

Abstract Parsing

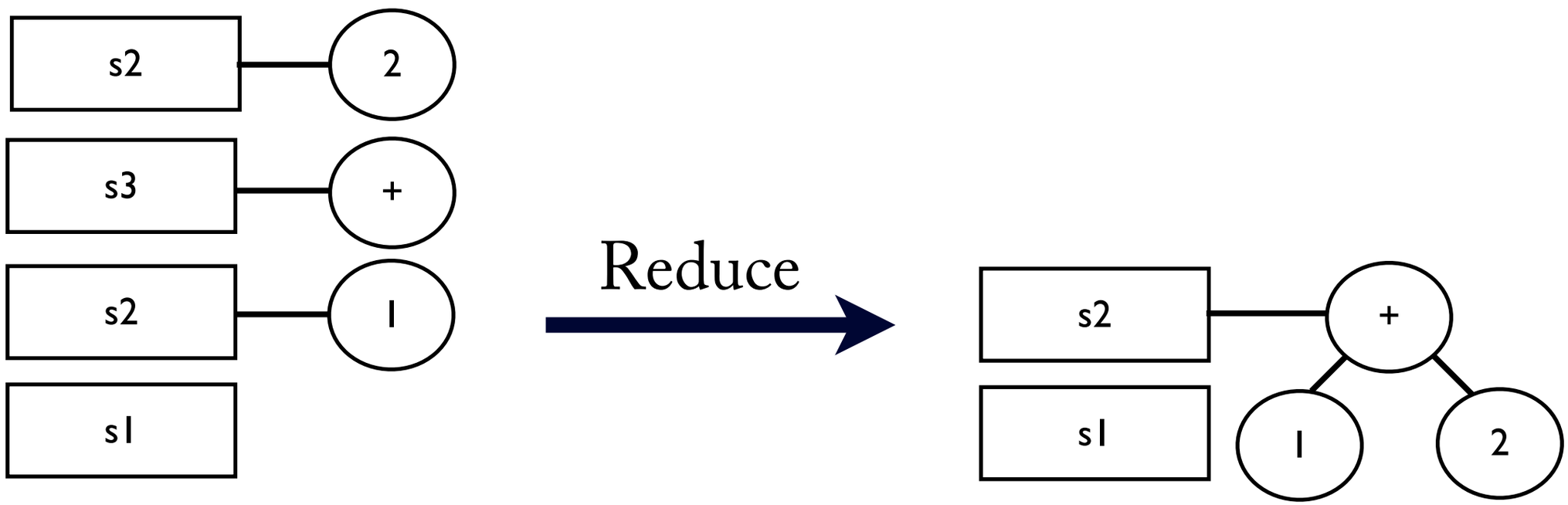
$$2^{Code} \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} 2^{P \to P} \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} 2^{P} \to 2^{P}$$

Extended Abstract Parsing

$$2^{Code} \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \boxed{2^{P_{AST} \to P_{AST}}} \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \boxed{2^{P_C \to P_C}} \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \boxed{2^{P_C} \to 2^{P_C}}$$

P_{AST} = ParseStack with AST

: List of (ParsingState, Corresponding AST)



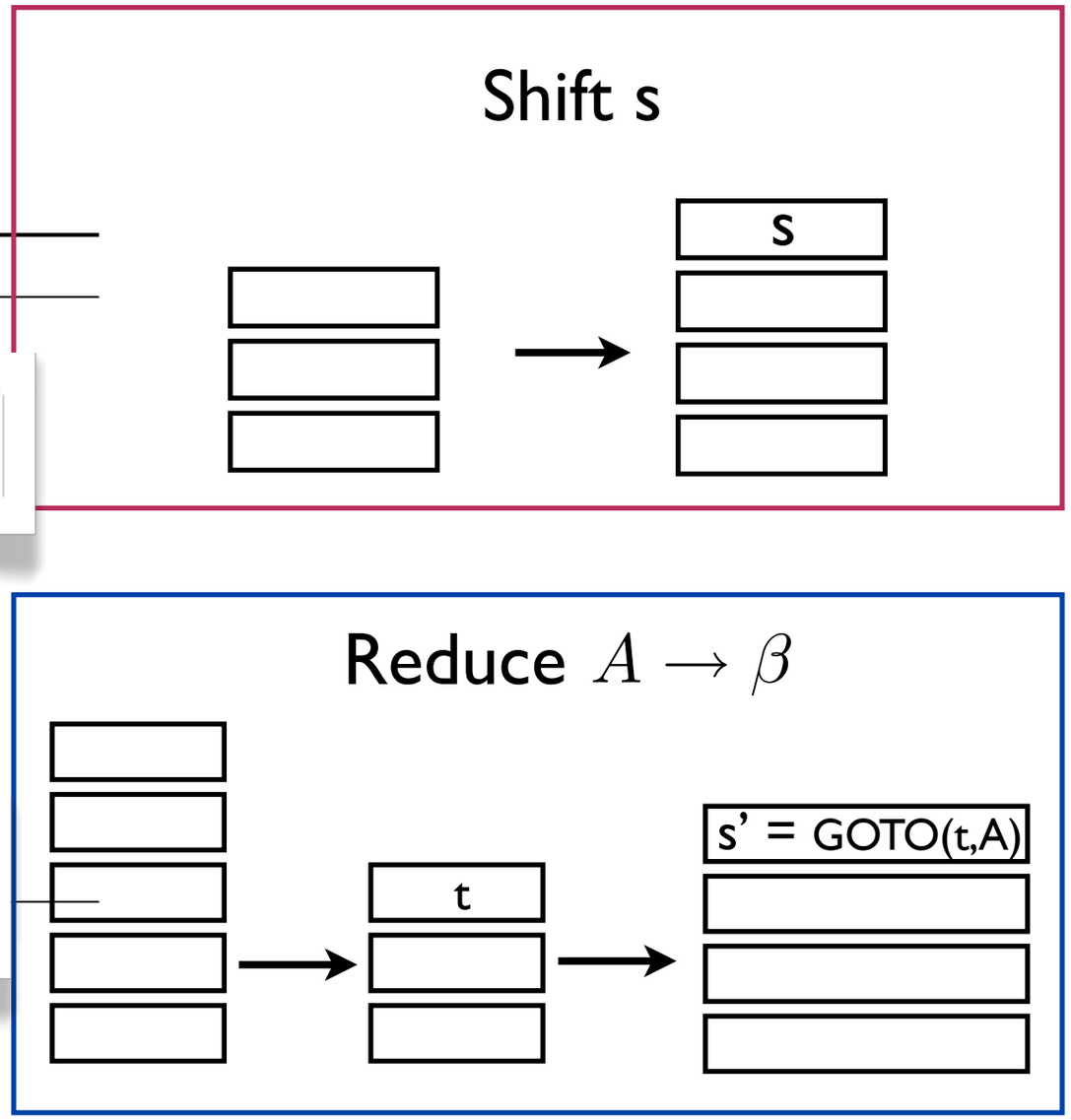
P_{AST} = ParseStack with AST

Algorithm 1 *parse_action* algorithm

```
1:  procedure parse_action(p, t)
2:      s_top ← the state on top of stack p
3:      if ACTION[s_top, t] = shift s then
4:          push s onto the stack p
5:          return p
6:      else if ACTION[s_top, t] = reduce A → β then
7:          pop |β| symbol off the stack p
8:          s_top ← the state on top of stack p
9:          push GOTO[s_top, A] onto the stack p
10:         return parse_action(p, t)
11:     end if
12: end procedure
```

Add LeafNode

Compose NewNode

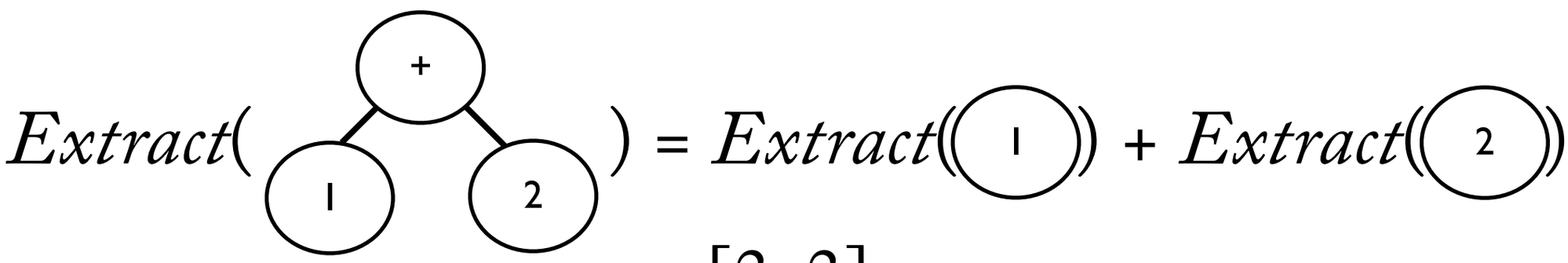


P_C = ParseStack with Constraint

: List of (ParsingState, Corresponding Constraint)

1. Need $Extract : AST \to C$
2. *Extract* has to be **compositional**, which means a constraint of an AST should be a composition of constraints of subASTs.

<u>example</u>



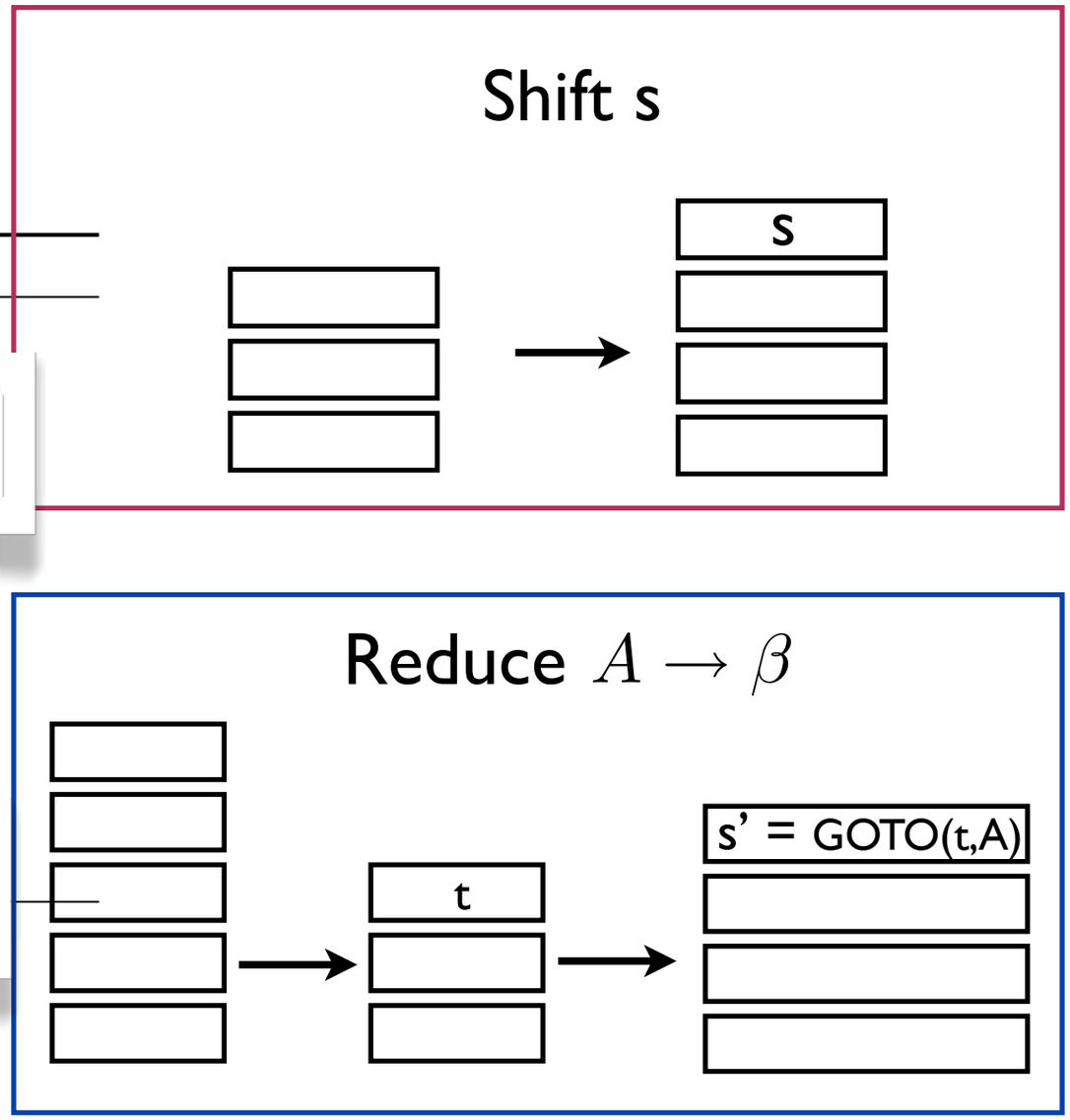
P_C = ParseStack with Constraint

Algorithm 1 *parse_action* algorithm

```
1:  procedure parse_action(p, t)
2:      s_top ← the state on top of stack p
3:      if ACTION[s_top, t] = shift s then
4:          push s onto the stack p
5:          return p
6:      else if ACTION[s_top, t] = reduce A → β then
7:          pop |β| symbol off the stack p
8:          s_top ← the state on top of stack p
9:          push GOTO[s_top, A] onto the stack p
10:         return parse_action(p, t)
11:     end if
12: end procedure
```

Add Constraint

Compose New Constraint



Compositional Constraints

- Uninitialized Local Variables Analysis
- Interval Value Analysis
- Simple Type Inference

Work in Progress

- Formalize Extended Abstract Parsing

$$2^{Code} \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \boxed{2^{P_{AST} \to P_{AST}}} \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \boxed{2^{P_C \to P_C}} \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \boxed{2^{P_C} \to 2^{P_C}}$$

- Formalize Instantiation of the Three Examples
  - Uninitialized Local Variables Analysis
  - Interval Value Analysis
  - Simple Type Inference

Thank You