PCC Framework for Program-Generators

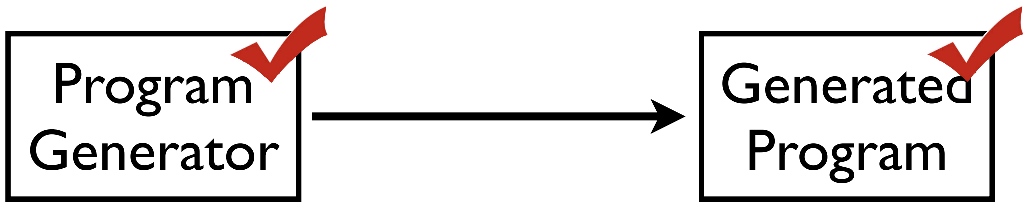
Soonho Kong   Wontae Choi   Kwangkeun Yi

Programming Research Lab.
Seoul National University
{soon,wtchoi,kwang}@ropas.snu.ac.kr

15 August 2009
Workshop on Proof-Carrying Code and Software Certification

Introduction

- Program-Generators
  - Need to ensure the safe execution of *the generated programs* as well as the generator itself.



  - Safety properties of the generated programs are efficiently expressed by *the grammar* **G**.
    e.g. *"generated programs should not have nested loops"*
  - **Question**:
    "Do the generated programs conform to the safety grammar **G**?"

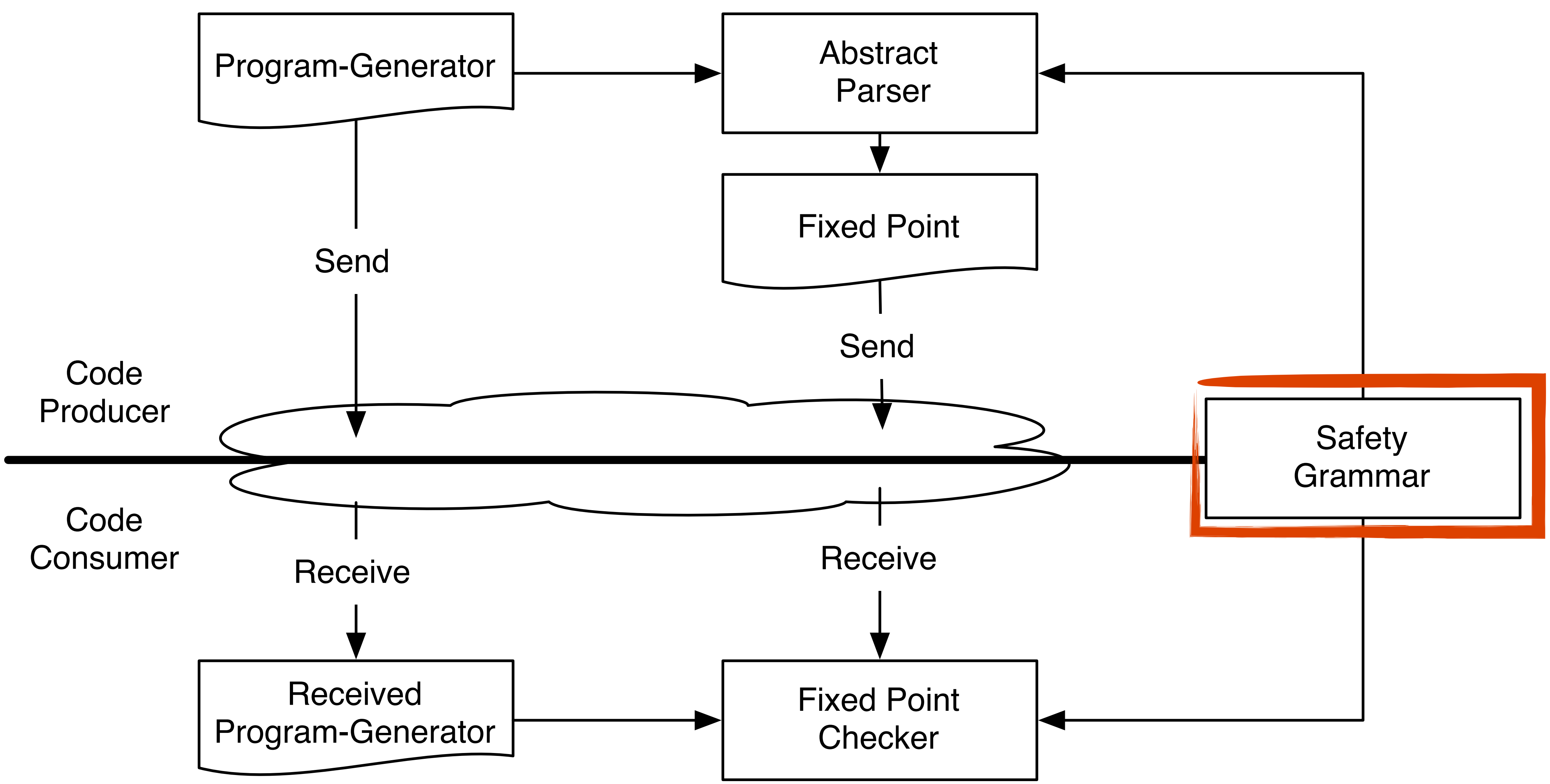
Introduction

- Abstract Parsing
  - Powerful static string analysis technique presented by Doh, Kim, and Schmidt[1]
  - Determine whether the strings generated in the program conform to the given grammar **G**.
  - Use LR parser as a component
  - Formalized and parameterized in the abstract interpretation framework by Kong, Choi and Yi[2]

---

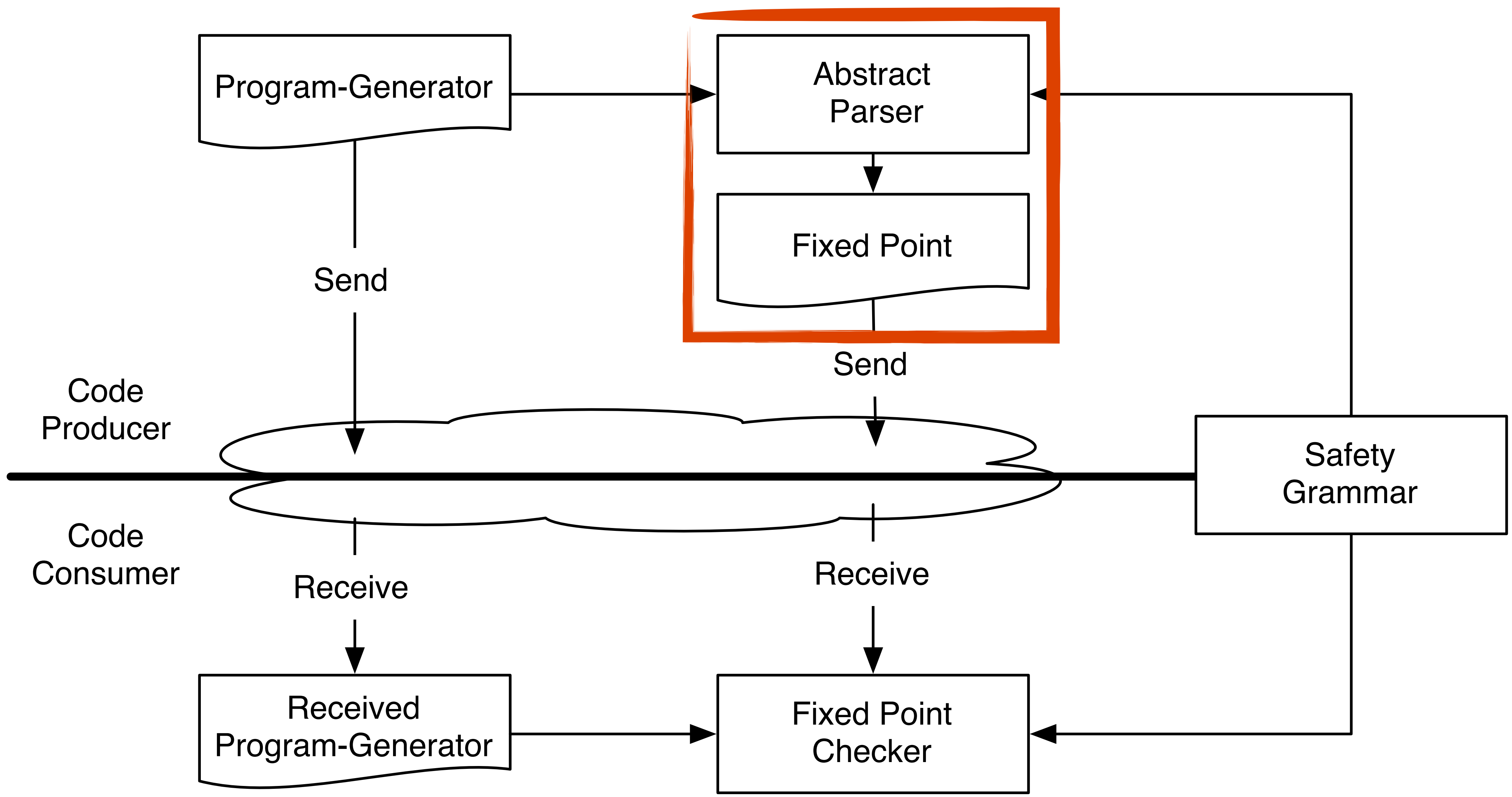
1. Kyung-Goo Doh, Hyunha Kim, and David Schmidt. *"Abstract parsing: static analysis of dynamically generated string output using LR-parsing technology."* In Proceeeding of the International Static Analysis Symposium, 2009.
2. Soonho Kong, Wontae Choi, Kwangkeun Yi. *"Abstract Parsing for Two-staged Languages with Concatenation*" International Conference on Generative Programming and Component Engineering, (to appear), 2009.

Big Picture
PCC Framework for Program-Generators



Safety grammar is shared between code producer and consumer.

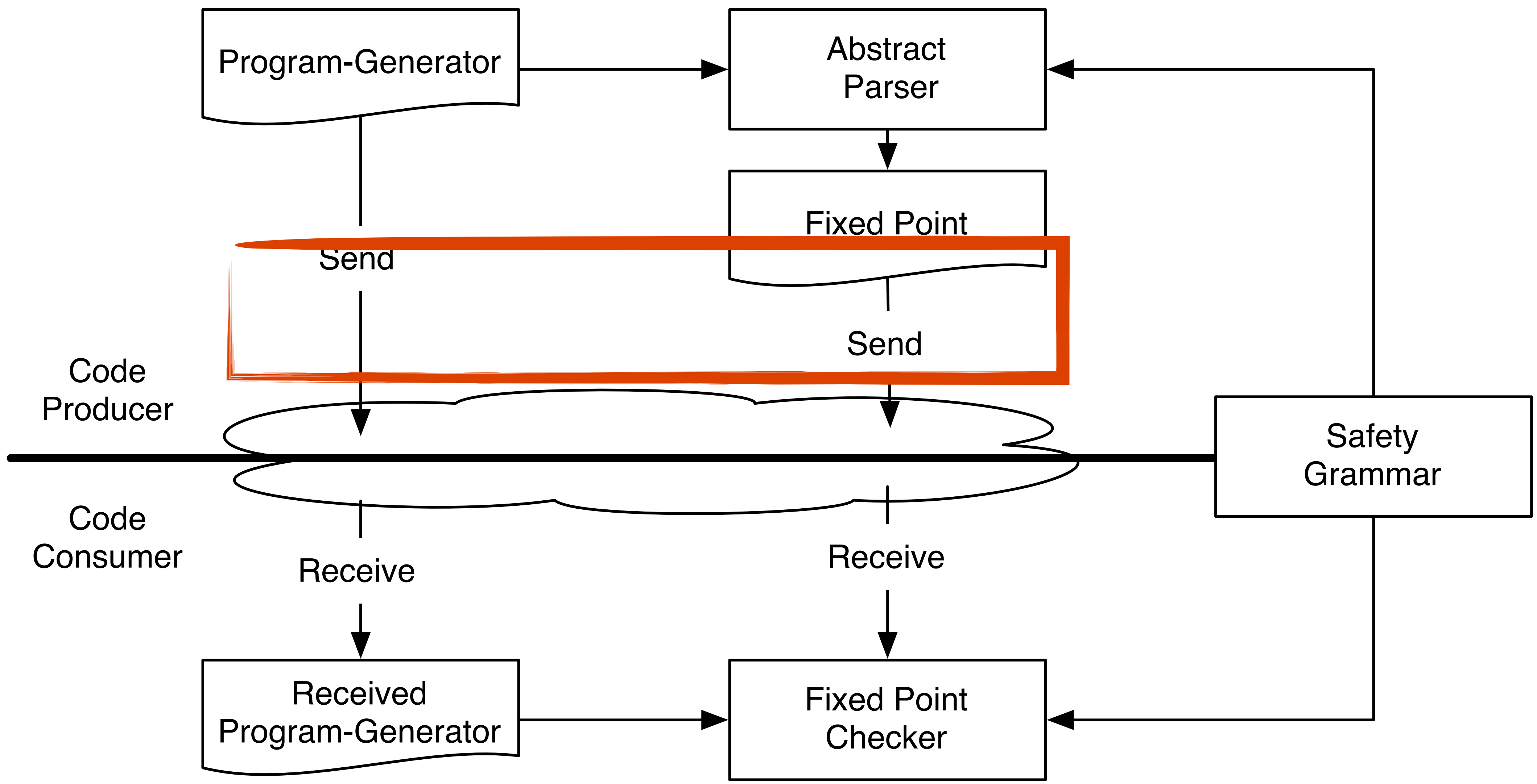
Big Picture
PCC Framework for Program-Generators



In code producer side, abstract parser computes fixed-point solution for the given program-generator.

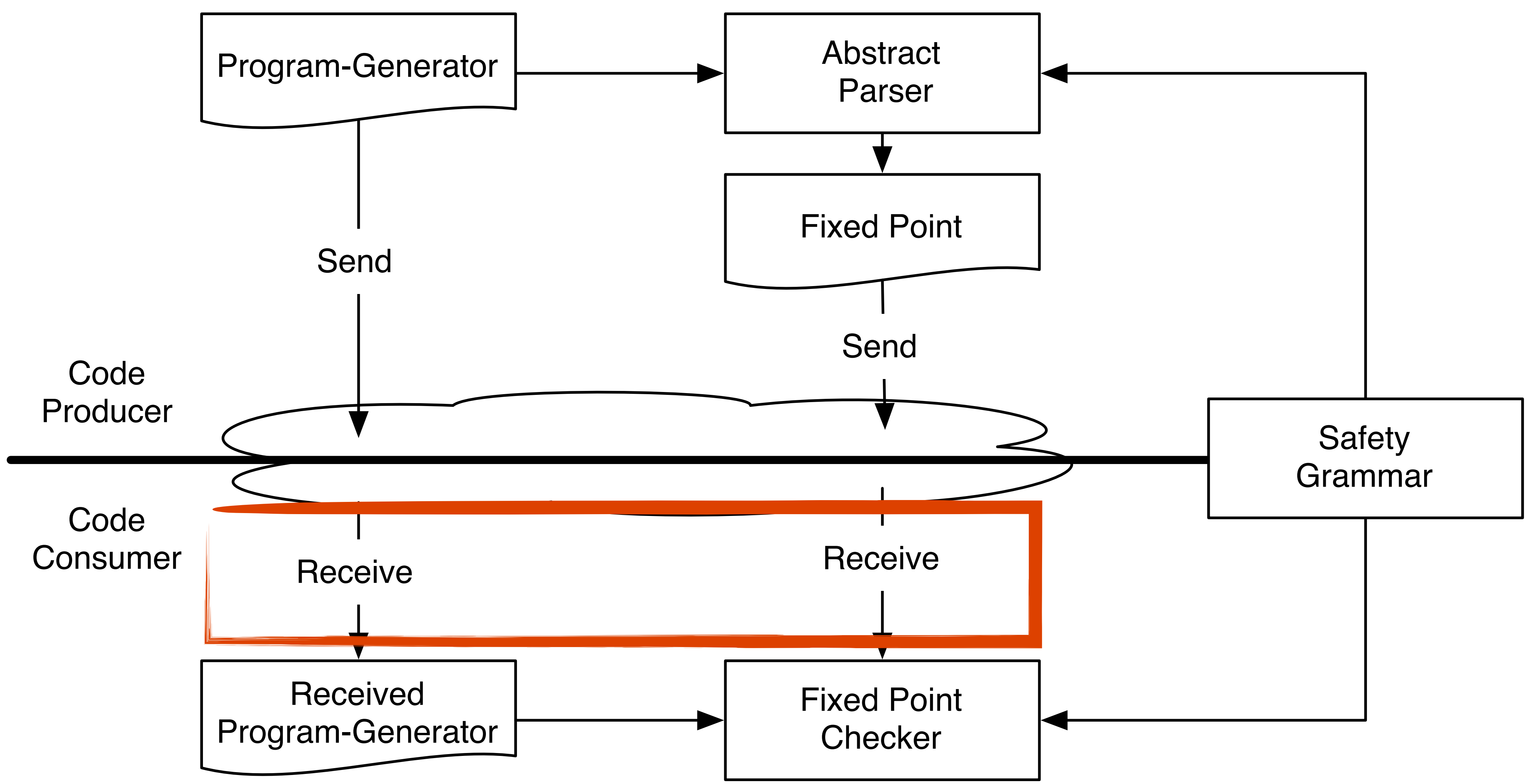
Big Picture

PCC Framework for Program-Generators



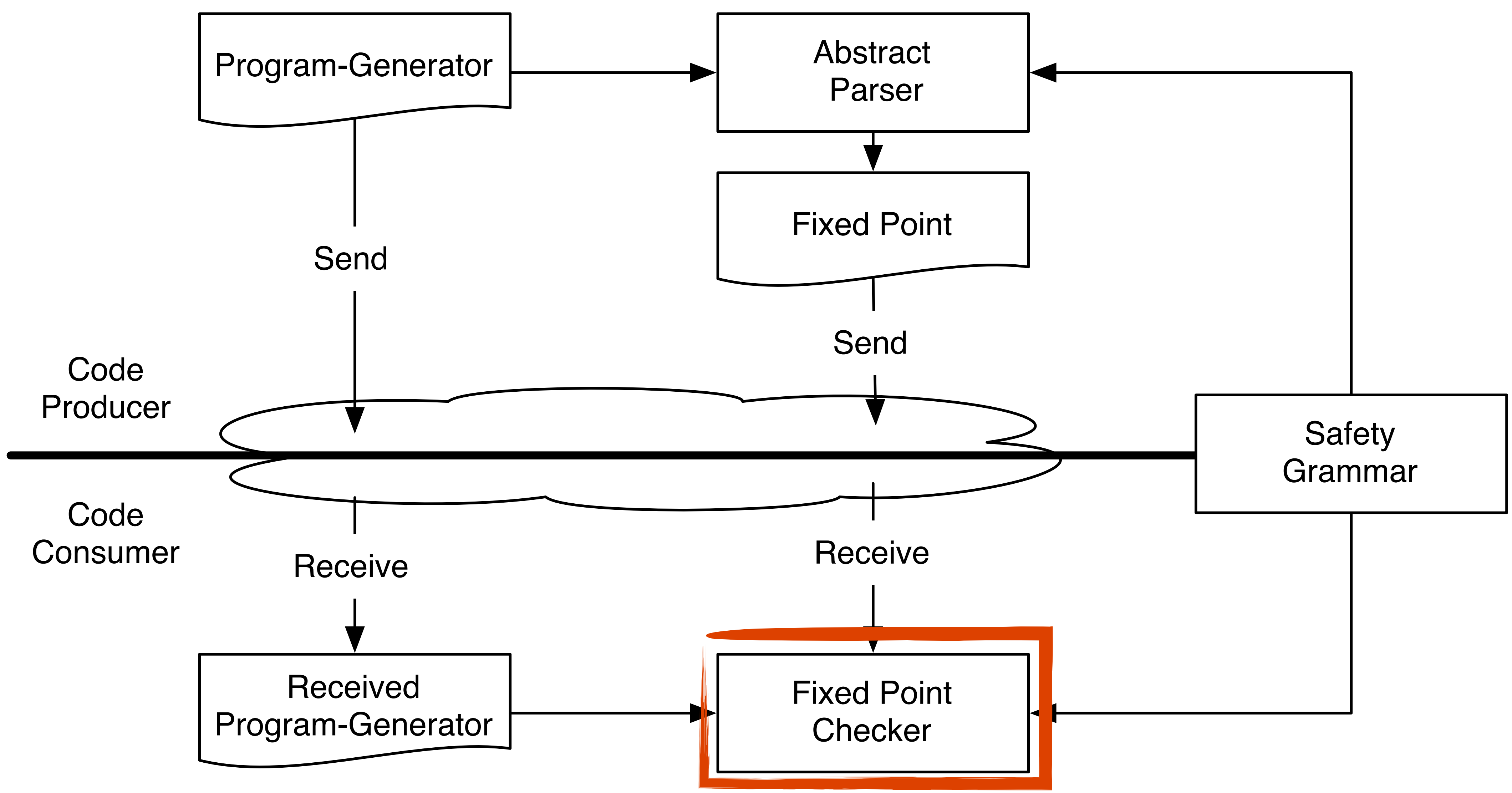
Code producer sends the program-generator with the computed fixed-point solution.

Big Picture
PCC Framework for Program-Generators



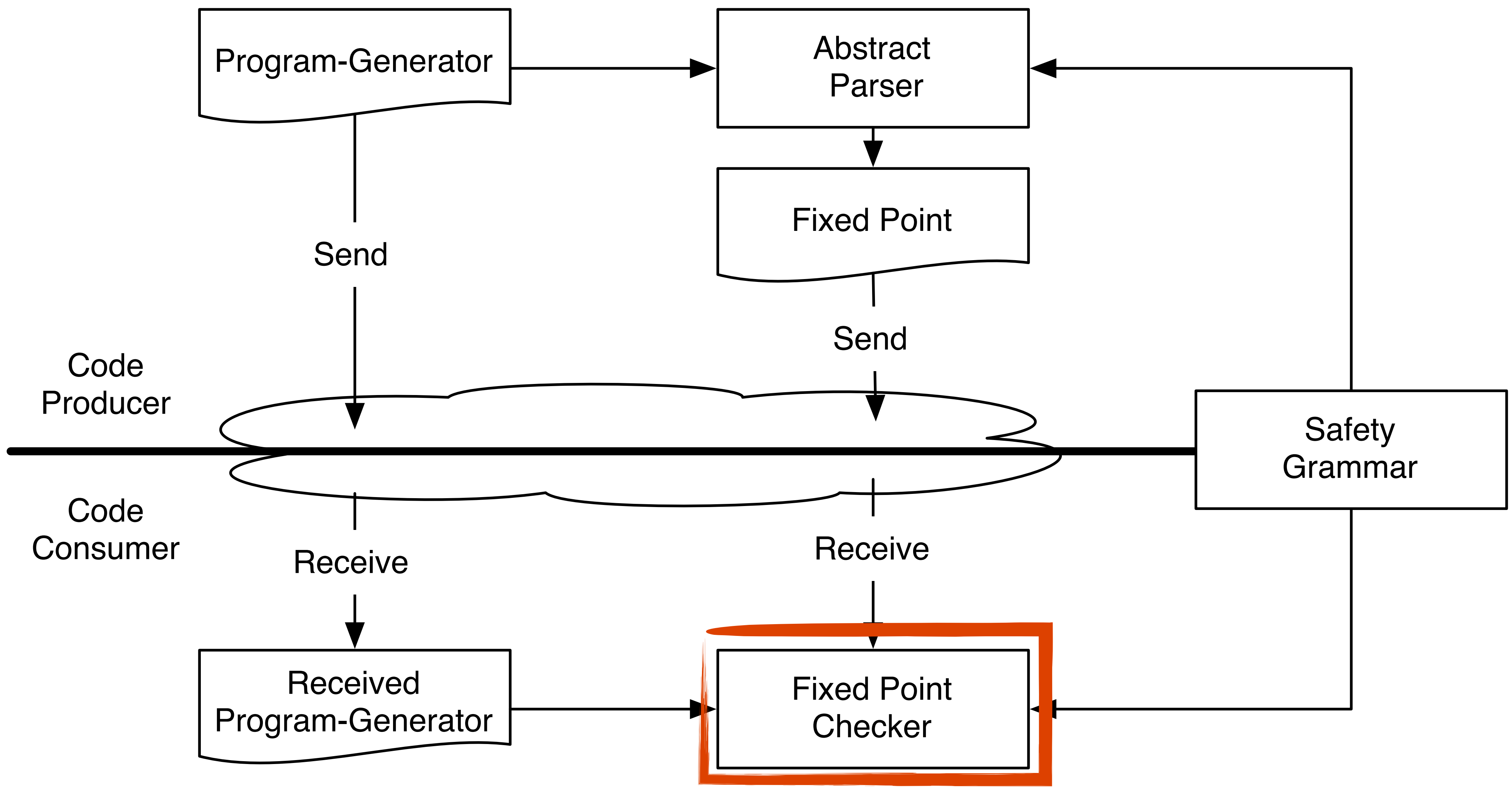
Code consumer receives an untrusted program-generator and an accompanied fixed-point solution.

Big Picture
PCC Framework for Program-Generators



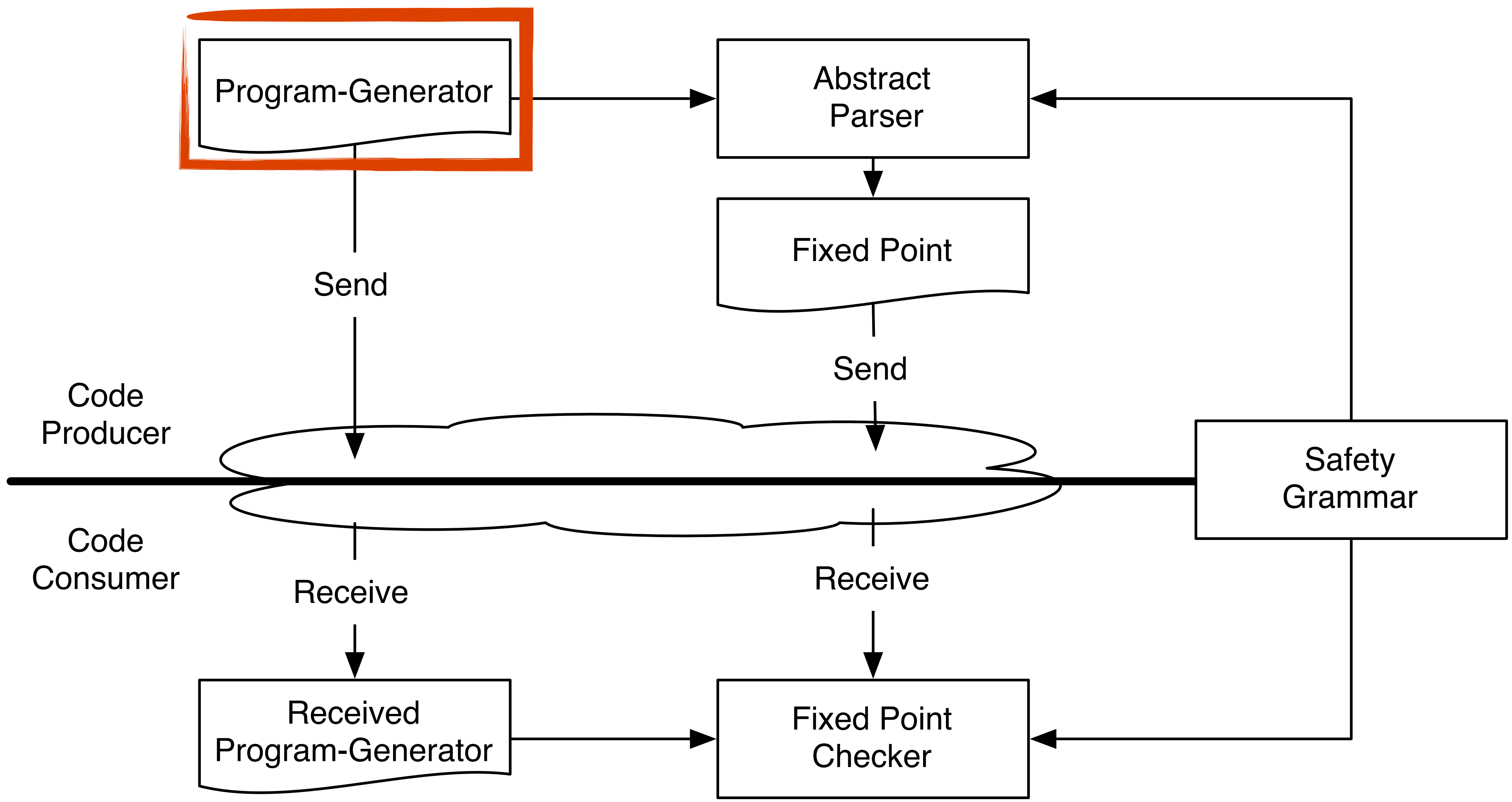
In code consumer side, the checker validates that the received fixed-point is indeed the solution for the received program-generator.

Big Picture
PCC Framework for Program-Generators



If the checker validates it successfully,
the code consumer is ready to execute the received program-generator.

Program-Generator



Language for Program-Generators

- Tow-staged language with concatenation

Syntax

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

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

Language for Program-Generators

Example

x is initialized with “a”

```
re x `a
   `( . ,x . )
   let y
       `or . a
       `,y . ,x

=>
```

Language for Program-Generators

Example

Loop body is not executed.

```
re x `a
  `( . ,x . )
  let y
    `or . a
    `,y . ,x

=>
```

x is initialized with “a”

y is initialized with “or a”

Language for Program-Generators

Example

Loop body is not executed.

```
re x `a
   `( . ,x . )
   let y
      `or . a
      `,y . ,x

=>
```

x is initialized with “a”

y is initialized with “or a”

Value is “or a a”

Language for Program-Generators

Example

Loop body is not executed.

```
re x `a
   `( . ,x . )
   let y
      `or . a
      `,y . ,x

=>  or a a        (if loop body is not executed)
```

x is initialized with “a”

y is initialized with “or a”

Value is “or a a”

Language for Program-Generators

Example

x is initialized with "a"

```
re x `a
  `( . ,x . )
  let y
     `or . a
     `,y . ,x

=>  or a a
```

Language for Program-Generators

Example

```
re x `a
   `( . ,x . )
   let y
       `or . a
       `,y . ,x

=>  or a a
```

x is initialized with “a”

x is “(a)”

Loop body is executed once

Language for Program-Generators

Example

```
re x `a
   `( . ,x . )
   let y
      `or . a
      `,y . ,x

=>  or a a
```

x is initialized with "a"

x is "(a)"

y is initialized with "or a"

Loop body is executed once

Language for Program-Generators

Example

Loop body is executed once

```
re x `a
   `( . ,x . )
   let y
       `or . a
       `,y . ,x

=>  or a a
```

x is initialized with “a”

x is “(a)”

y is initialized with “or a”

Value is “or a (a)”

Language for Program-Generators

Example

Loop body is executed once

```
re x `a
   `( . ,x . )
   let y
       `or . a
       `,y . ,x

=>  or a a
    or a (a)     (if loop body is executed once)
```

x is initialized with “a”

x is “(a)”

y is initialized with “or a”

Value is “or a (a)”

Language for Program-Generators

Example

x is initialized with "a"

```
re x `a
   `( . ,x . )
   let y
       `or . a
       `,y . ,x

=>  or a a
    or a (a)
```

Language for Program-Generators

Example

Loop body is executed once

x is initialized with "a"

x is "(a)"

```
re x `a
   `( . ,x . )
   let y
      `or . a
      `,y . ,x

=>  or a a
    or a (a)
```

Language for Program-Generators

Example

Loop body is executed twice

```
re x `a
   `( . ,x . )
   let y
       `or . a
       `,y . ,x

=>  or a a
    or a (a)
```

x is initialized with “a”

x is “((a))”

Language for Program-Generators

Example

Loop body is executed twice

```
re x `a
   `( . ,x . )
   let y
       `or . a
       `,y . ,x

=>  or a a
    or a (a)
```

x is initialized with "a"

x is "((a))"

y is initialized with "or a"

Language for Program-Generators

Example

Loop body is executed twice

```
re x `a
   `( . ,x . )
   let y
       `or . a
       `,y . ,x

=>  or a a
    or a (a)
```

x is initialized with “a”

x is “((a))”

y is initialized with “or a”

Value is “or a ((a))”

Language for Program-Generators

Example

Loop body is executed twice

```
re x `a
   `( . ,x . )
   let y
      `or . a
      `,y . ,x

=>  or a a
    or a (a)
    or a ((a))          (if loop body is executed twice)
```

x is initialized with "a"

x is "((a))"

y is initialized with "or a"

Value is "or a ((a))"

Language
for Program-Generators

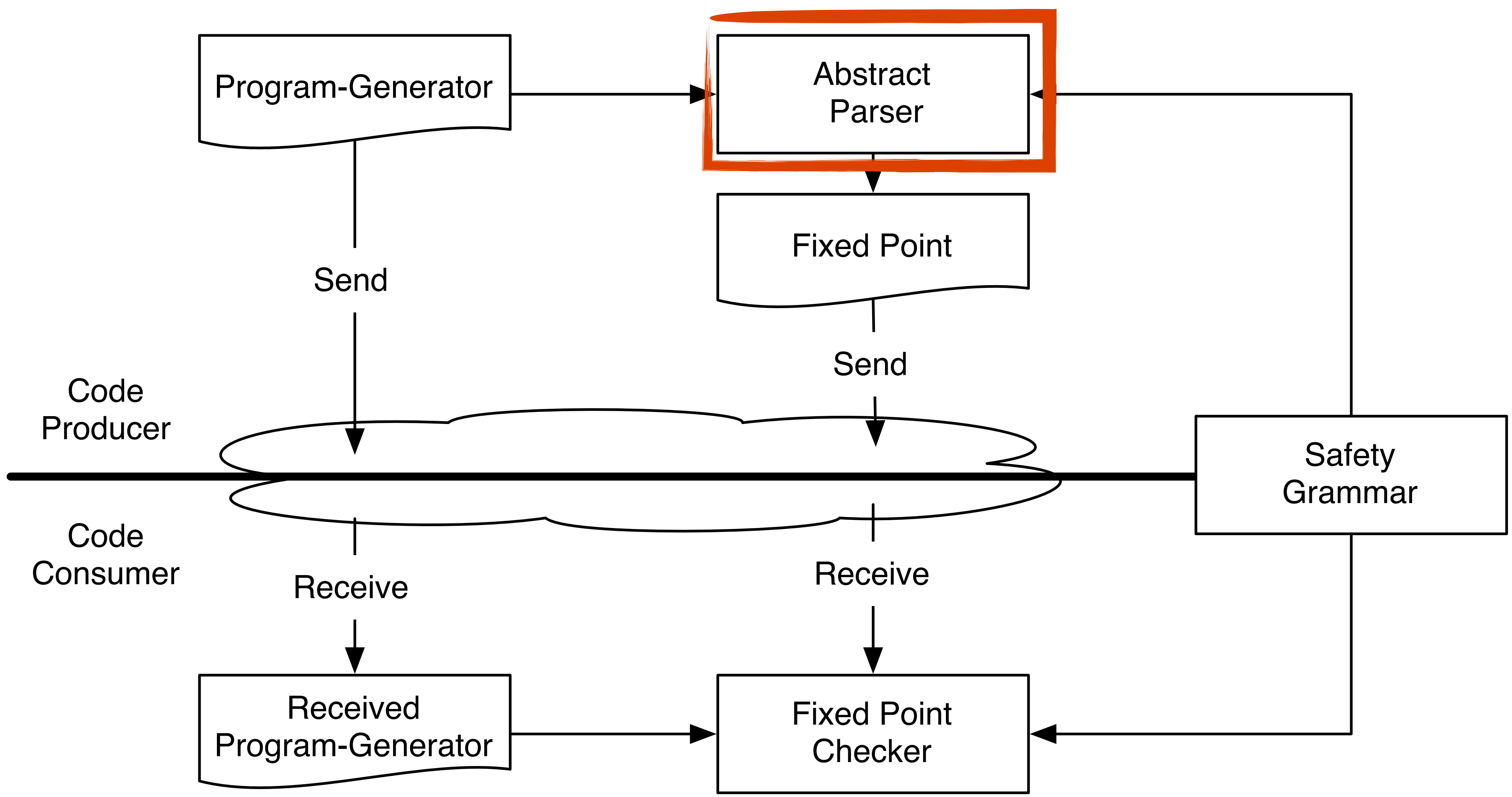
Example

```
re x `a
   `( . ,x . )
   let y
       `or . a
       `,y . ,x

=>  or a a
    or a (a)
    or a ((a))
    or a (((a)))
    .
    .
    .
```

Note that only one of them is taken as a value of this program in execution.

Abstract Parsing



Abstract Parsing

- Instead of executing the program and parsing the result,

$$[\![e]\!]^0\Sigma = \{c_1, c_2, \ldots, c_n\} \quad parse(c_i) = O/X$$

- Define abstract semantics using parse stack and execute the program on it.

$$[\![\hat{e}]\!]^0\Sigma\{p_{init}\} = \{p_1, p_2, \ldots, p_n\}$$

Over-approximation of the parsing result of all the generated programs

Abstract Parsing

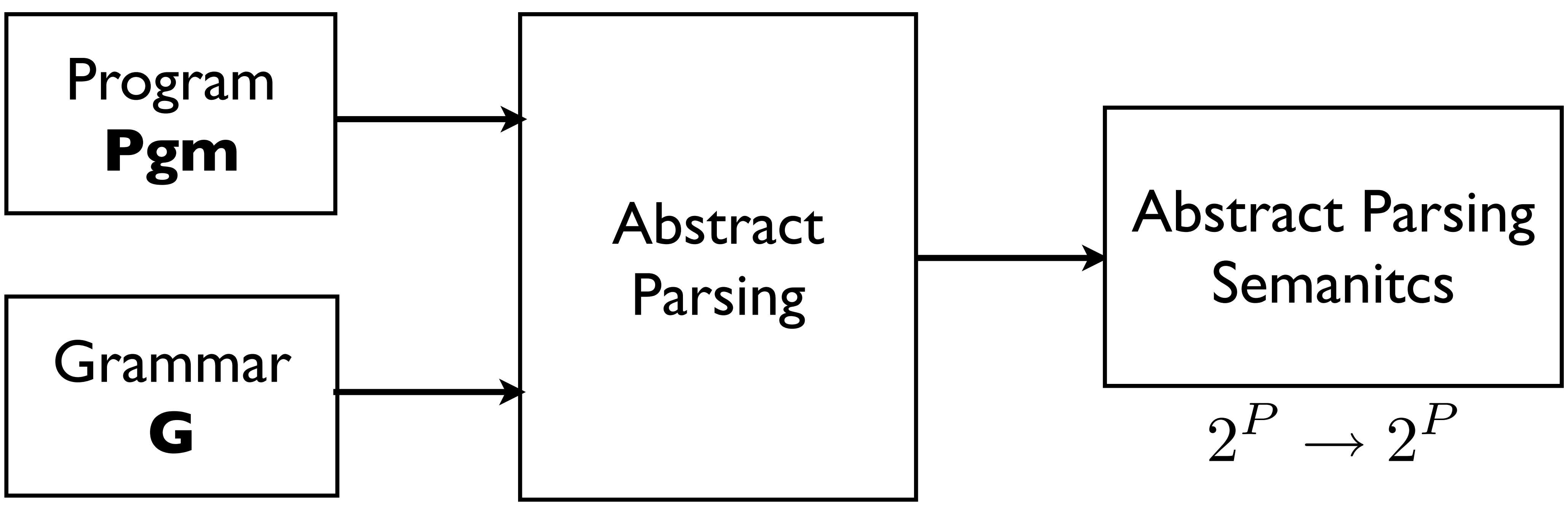
- Q: What should be the abstract value for Code c?
- A: Parse Stack Transition Function

`or`		`a`		`or a`
...		...		...
$s_1 \mapsto s_1 s_5$	+	$s_1 s_5 \mapsto s_1 s_5 s_{10}$	=>	$s_1 \mapsto s_1 s_5 s_{10}$
...		...		

Code concatenation => Function Composition

Abstract Parsing

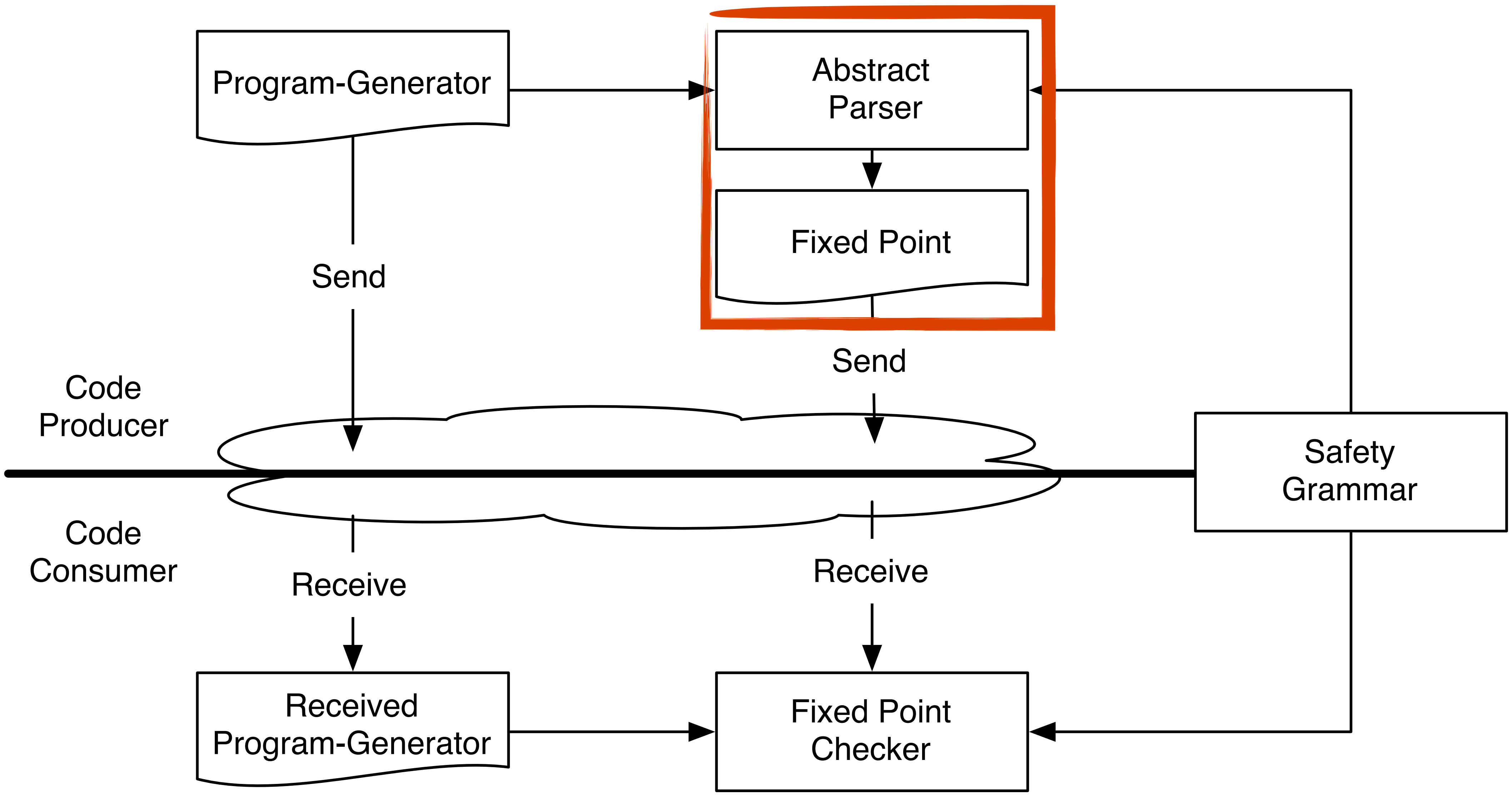
- Abstract parsing semantics of the program **Pgm** is used to determine whether generated programs conform to the grammar **G**.
- If $AbstractParsing(Pgm, G)(\{P_{init}\}) = \{P_{acc}\}$, then we can conclude that generated programs conform to the grammar **G**. Otherwise not.



Abstract Parsing in PCC Framework

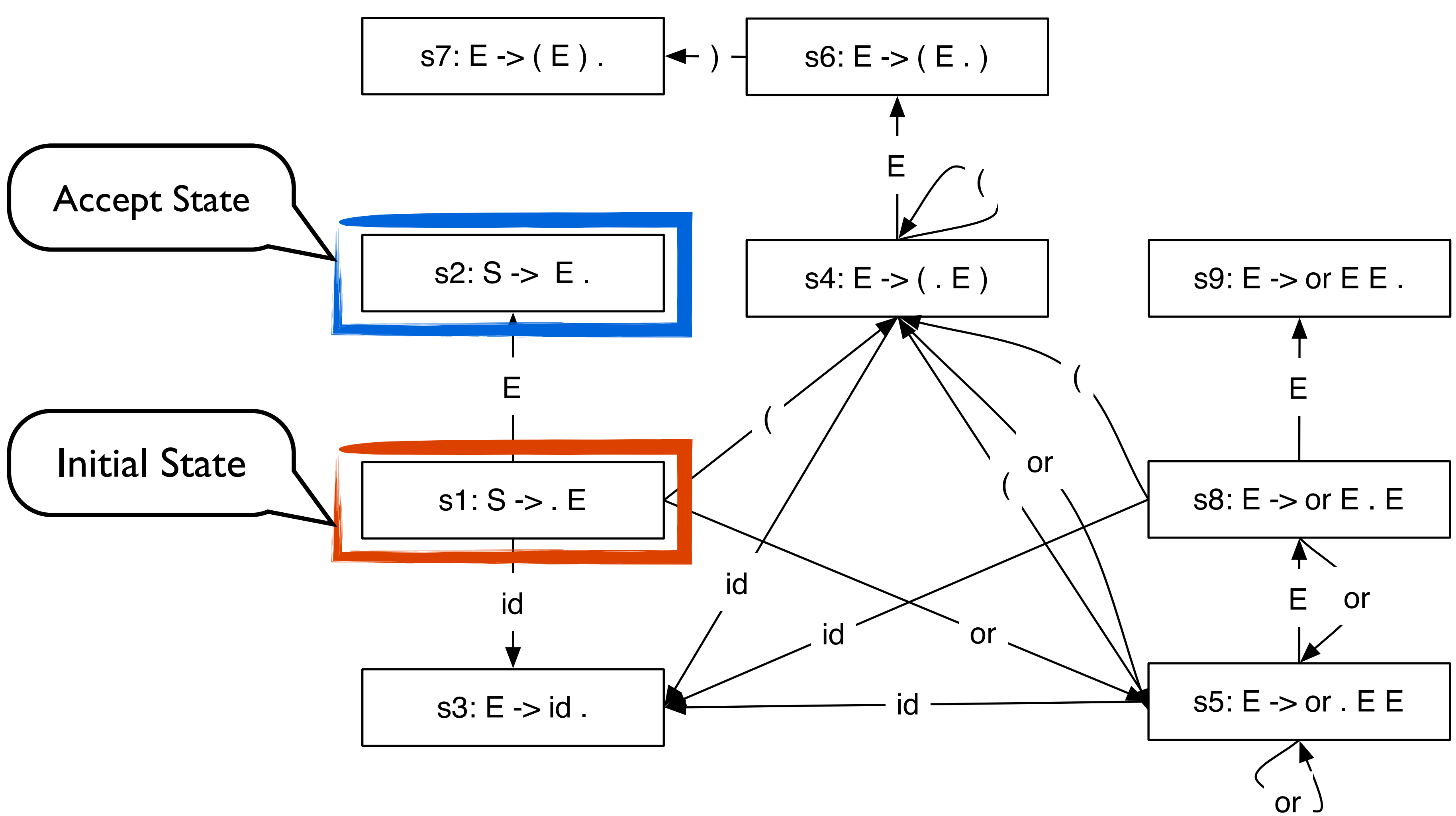
- Need *abstract parsing semantics* to certify the program.
- Semantic equations are derived from the program directly.
- Loop is the only component to require fixed-point computation.
- Certificate in our framework:
  the fixed-point solution for every loop in the program.

Certificate Generation



Certificate Generation with Example

- Safety Grammar $E \rightarrow id \mid (E) \mid \texttt{let}\ id\ E\ E \mid \texttt{or}\ E\ E$
  1) syntactically correct, 2) contain no loops



Part of the LR(0) parsing controller for the safety grammar

Certificate Generation with Example

Example Program

```
    re x `a
       `( . ,x . )
       let y
           `or . a
           `,y . ,x
```

Semantic Equation

Certificate Generation with Example

Example Program

```
re x `a
  `( . ,x . )
  let y
    `or . a
    `,y . ,x
```

Semantic Equation

$$P = X \circ Y$$

Certificate Generation with Example

Example Program

```
re x `a
  `( . ,x . )
  let y
    `or . a
    ,y   ,x
```

Semantic Equation

$$P = X \circ Y$$

Certificate Generation with Example

Example Program

```
re x `a
   `( . ,x . )
   let y
       `or . a
       `,y . ,x
```

Semantic Equation

$$P = X \circ Y$$

$$Y = \lambda P.PA(PA(P, \texttt{or}), \texttt{a})$$

ParseAction : Parse Stack x Token -> Parse Stack
Component of generated LR Parser

Certificate Generation with Example

Example Program

```
re x `a
   `( . ,x . )
   let y
       `or . a
       `,y . ,x
```

Semantic Equation

$$P = X \circ Y$$

$$Y = \lambda P.PA(PA(P, \texttt{or}), \texttt{a})$$

Certificate Generation with Example

Example Program

```
re x `a
   `( . ,x . )
   let y
       `or . a
       `,y . ,x
```

Semantic Equation

$$P = X \circ Y$$

$$Y = \lambda P.PA(PA(P, \texttt{or}), \texttt{a})$$

$$T = \mathit{fix}\,\lambda T.\lambda s.(PA(s, \texttt{a}) \cup \lambda P.PA(P, \texttt{)}) \circ \lambda P.\mathit{reduce}(T(\mathit{top}(P))@\mathit{tail}(P)) \circ \lambda P.PA(P, \texttt{(})s)$$

$$X = \lambda P.\mathit{reduce}(T(\mathit{top}(P))@\mathit{tail}(P))$$

Certificate Generation with Example

Example Program

```
    re x `a
      `( . ,x . )
      let y
        `or . a
        `,y . ,x
```

Semantic Equation

$$P = X \circ Y$$

$$Y = \lambda P.PA(PA(P, \texttt{or}), \texttt{a})$$

$$T = \mathit{fix} \lambda T.\lambda s.(PA(s, \texttt{a})) \cup \lambda P.PA(P, \texttt{)}) \circ \lambda P.\mathit{reduce}(T(\mathit{top}(P))@\mathit{tail}(P)) \circ \lambda P.PA(P, \texttt{(})s)$$

$$X = \lambda P.\mathit{reduce}(T(\mathit{top}(P))@\mathit{tail}(P))$$

Certificate Generation with Example

Example Program

```
re x `a
   `( . ,x . )
   let y
       `or . a
       `,y . ,x
```

Semantic Equation

$$P = X \circ Y$$

$$Y = \lambda P.PA(PA(P, \texttt{or}), \texttt{a})$$

$$T = \mathit{fix}\lambda T.\lambda s.(PA(s, \texttt{a}) \cup$$

$$\lambda P.PA(P, \texttt{)}) \circ \lambda P.\mathit{reduce}(T(\mathit{top}(P))@\mathit{tail}(P)) \circ \lambda P.PA(P, \texttt{(})s)$$

$$X = \lambda P.\mathit{reduce}(T(\mathit{top}(P))@\mathit{tail}(P))$$

Certificate Generation with Example

Example Program

```
re x `a
   `( . ,x . )
   let y
       `or . a
       `,y . ,x
```

Semantic Equation

$$P = X \circ Y$$

$$Y = \lambda P.PA(PA(P, \texttt{or}), \texttt{a})$$

$$T = \mathit{fix}\lambda T.\lambda s.(PA(s, \texttt{a}) \cup$$

$$\lambda P.PA(P, \texttt{)}) \circ \lambda P.\mathit{reduce}(T(\mathit{top}(P))@\mathit{tail}(P)) \circ \lambda P.PA(P, \texttt{(})s)$$

$$X = \lambda P.\mathit{reduce}(T(\mathit{top}(P))@\mathit{tail}(P))$$

Certificate Generation with Example

Example Program

```
re x `a
   `( . ,x . )
   let y
      `or . a
      `,y . ,x
```

Semantic Equation

$$P = X \circ Y$$

$$Y = \lambda P.PA(PA(P, \texttt{or}), \texttt{a})$$

$$T = \mathit{fix}\lambda T.\lambda s.(PA(s, \texttt{a}) \cup \lambda P.PA(P, \texttt{)}) \circ \lambda P.\mathit{reduce}(T(\mathit{top}(P))@\mathit{tail}(P)) \circ \lambda P.PA(P, \texttt{(})s)$$

$$X = \lambda P.\mathit{reduce}(T(\mathit{top}(P))@\mathit{tail}(P))$$

Certificate Generation with Example

Example Program

```
re x `a
   `( . ,x  )
   let y
       `or . a
       `,y . ,x
```

Semantic Equation

$$P = X \circ Y$$

$$Y = \lambda P.PA(PA(P, \texttt{or}), \texttt{a})$$

$$T = \mathit{fix}\,\lambda T.\lambda s.(PA(s, \texttt{a}) \cup$$
$$\lambda P.PA(P, \texttt{)}) \circ \lambda P.\mathit{reduce}(T(\mathit{top}(P))@\mathit{tail}(P)) \circ \lambda P.PA(P, \texttt{(})s)$$

$$X = \lambda P.\mathit{reduce}(T(\mathit{top}(P))@\mathit{tail}(P))$$

Certificate Generation with Example

Example Program

```
re x `a
  `( . ,x . )
  let y
    `or . a
    `,y . ,x
```

Semantic Equation

$$P = X \circ Y$$

$$Y = \lambda P.PA(PA(P, \texttt{or}), \texttt{a})$$

$$T = \mathit{fix}\,\lambda T.\lambda s.(PA(s, \texttt{a}) \cup \lambda P.PA(P, \texttt{)}) \circ \lambda P.\mathit{reduce}(T(\mathit{top}(P))@\mathit{tail}(P)) \circ \lambda P.PA(P, \texttt{(})s)$$

$$X = \lambda P.\mathit{reduce}(T(\mathit{top}(P))@\mathit{tail}(P))$$

T : Parse State -> Set of Parse Stack
Record the difference of the loop input and output.

Certificate Generation with Example

Example Program

```
re x `a
   `( . ,x . )
   let y
      `or . a
      `,y . ,x
```

Semantic Equation

$$
\begin{aligned}
P &= X \circ Y \\
Y &= \lambda P.PA(PA(P, \texttt{or}), \texttt{a}) \\
T &= \mathit{fix}\,\lambda T.\lambda s.(PA(s, \texttt{a}) \cup \\
&\qquad \lambda P.PA(P, \texttt{)})) \circ \lambda P.\mathit{reduce}(T(\mathit{top}(P))@\mathit{tail}(P)) \circ \lambda P.PA(P, \texttt{(})s) \\
X &= \lambda P.\mathit{reduce}(T(\mathit{top}(P))@\mathit{tail}(P))
\end{aligned}
$$

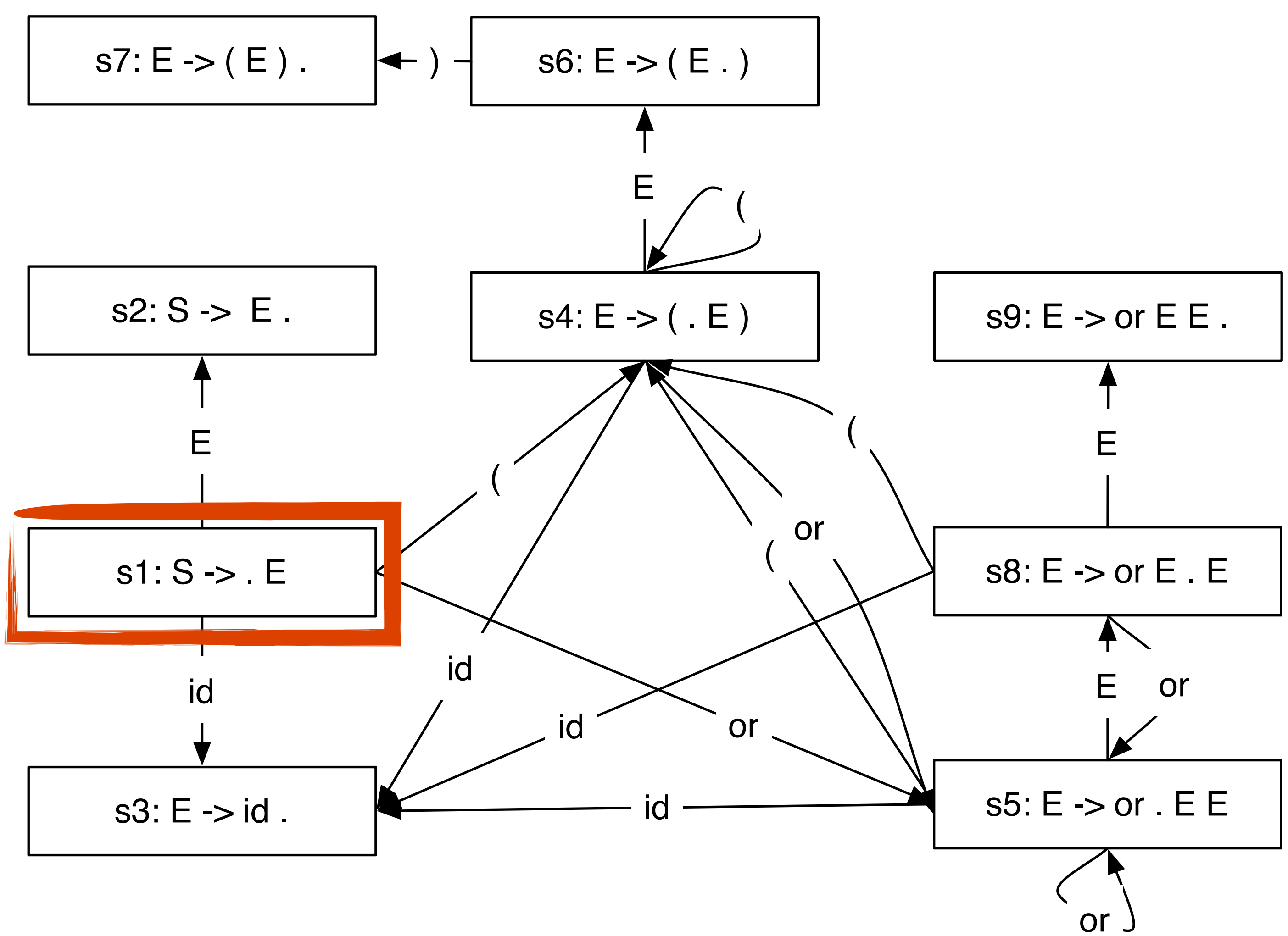
Need to compute fixed-point T!

Certificate Generation with Example

Semantic Equation

$$P = X \circ Y$$
$$Y = \lambda P.PA(PA(P, \texttt{or}), \texttt{a})$$

$$P(s_1) = X \circ Y(s_1)$$
$$= X \circ PA(PA(s_1, \texttt{or}), \texttt{a})$$



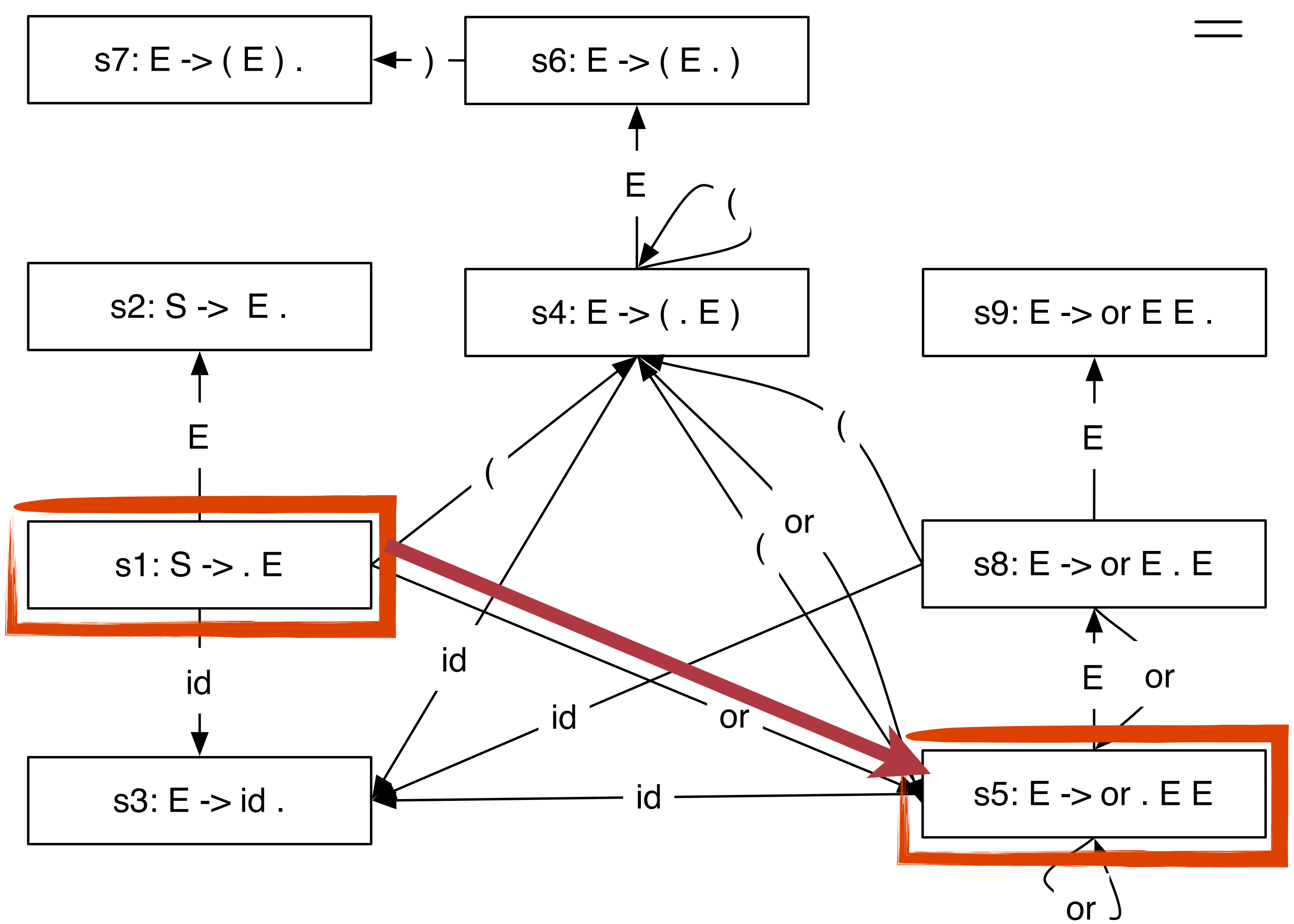
Part of the LR(0) parsing controller for the safety grammar

Certificate Generation with Example

Semantic Equation

$$P = X \circ Y$$
$$Y = \lambda P. PA(PA(P, \texttt{or}), \texttt{a})$$

$$\begin{aligned} P(s_1) &= X \circ Y(s_1) \\ &= X \circ PA(PA(s_1, \texttt{or}), \texttt{a}) \\ &= X \circ PA(s_5 s_1, \texttt{a}) \end{aligned}$$



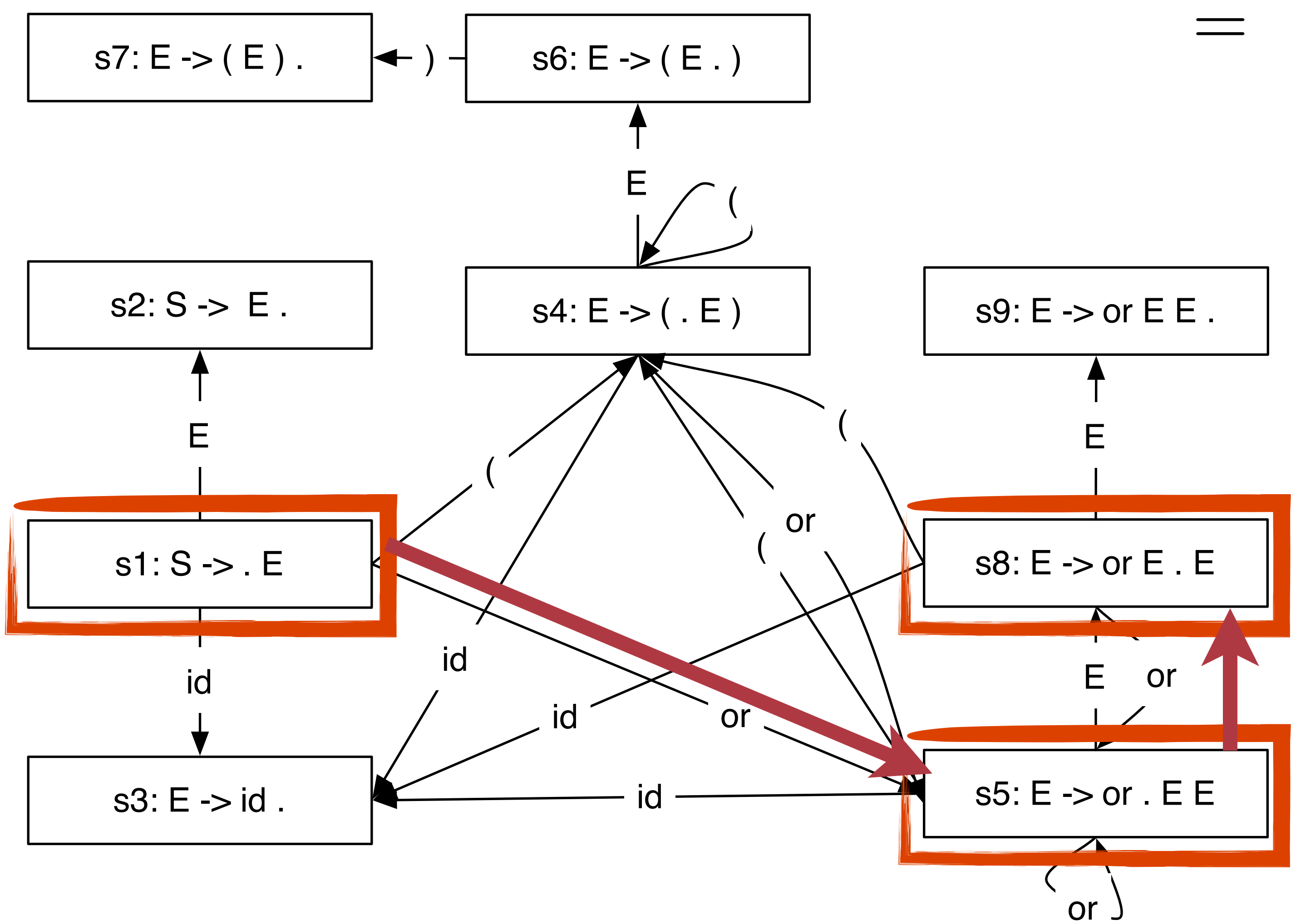
Part of the LR(0) parsing controller for the safety grammar

Certificate Generation with Example

Semantic Equation

$$P = X \circ Y$$
$$Y = \lambda P.PA(PA(P, \texttt{or}), \texttt{a})$$

$$\begin{aligned} P(s_1) &= X \circ Y(s_1) \\ &= X \circ PA(PA(s_1, \texttt{or}), \texttt{a}) \\ &= X(s_8 s_5 s_1) \end{aligned}$$



Part of the LR(0) parsing controller for the safety grammar

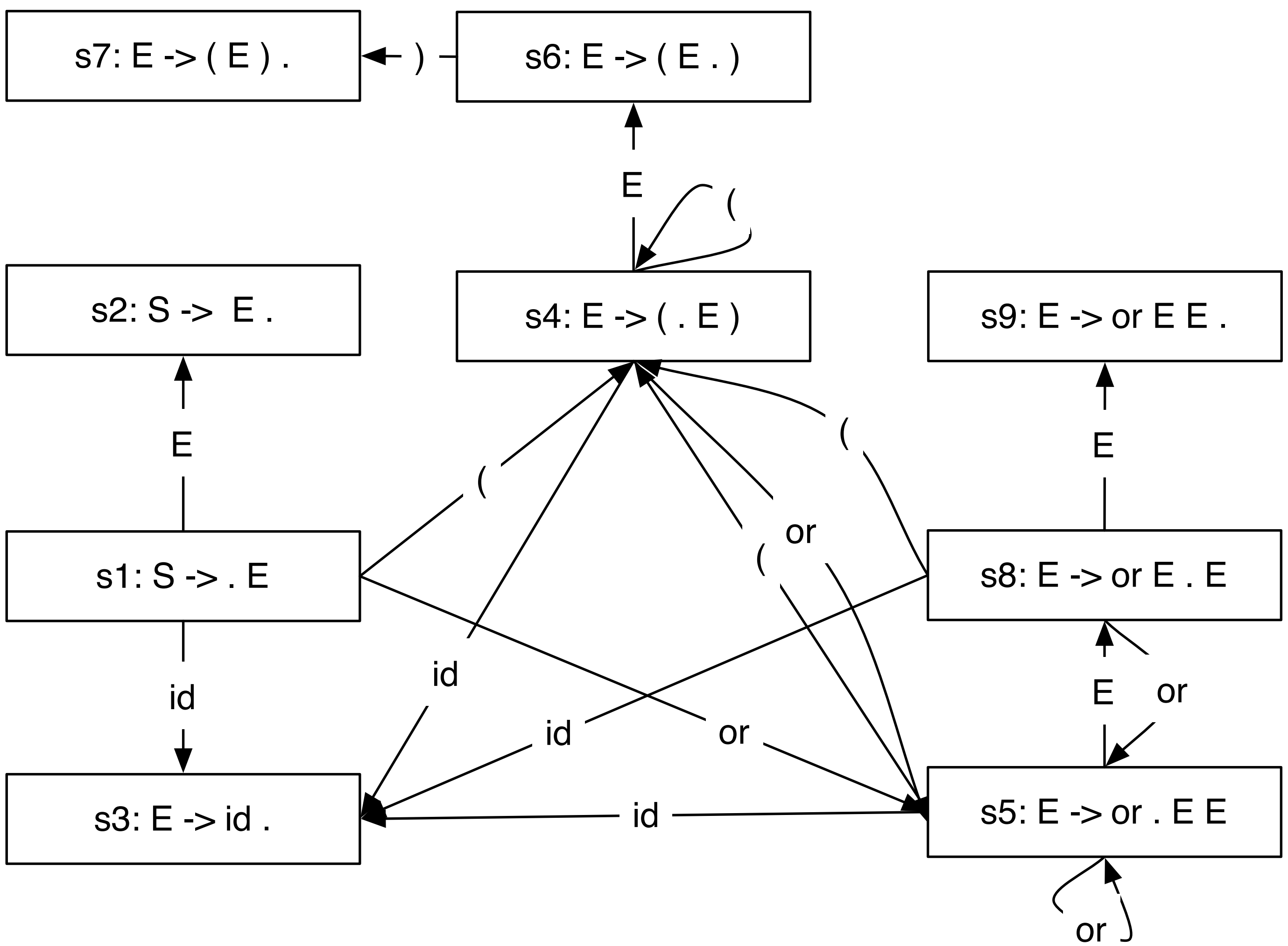
Certificate Generation with Example

Semantic Equation

$X = \lambda P.reduce(T(top(P))@tail(P))$

$X(s_8 s_5 s_1) = reduce(T(s_8)@s_5 s_1)$

We need $T(s_8)$.

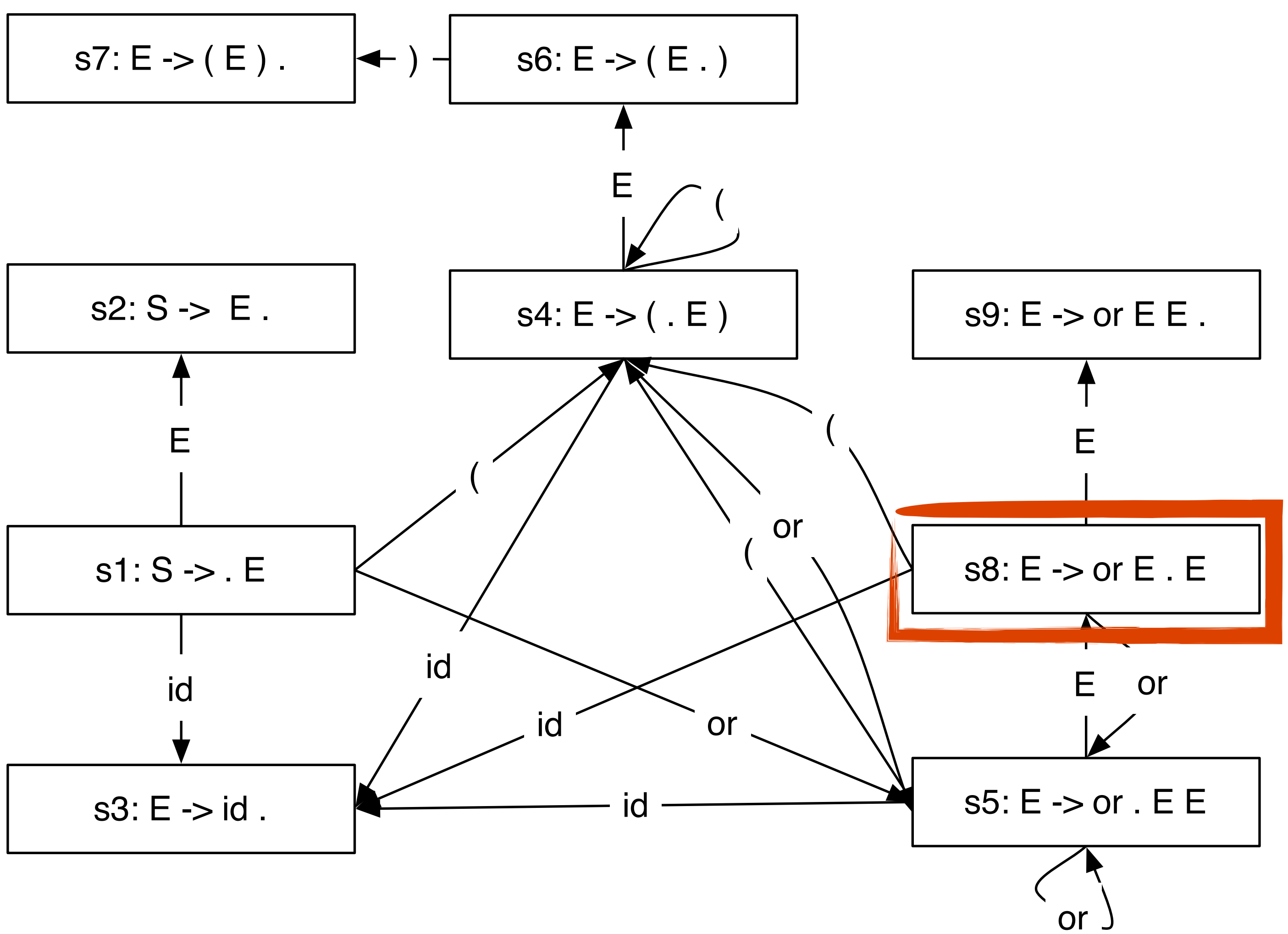


Part of the LR(0) parsing controller for the safety grammar

Certificate Generation with Example

Semantic Equation

$$T = \mathit{fix} \lambda T.\lambda s.(PA(s, \mathtt{a}) \cup$$
$$\lambda P.PA(P, \mathtt{)}) \circ \lambda P.\mathit{reduce}(T(\mathit{top}(P))@\mathit{tail}(P)) \circ \lambda P.PA(P, \mathtt{(})s)$$



$s_8 \mapsto ?$

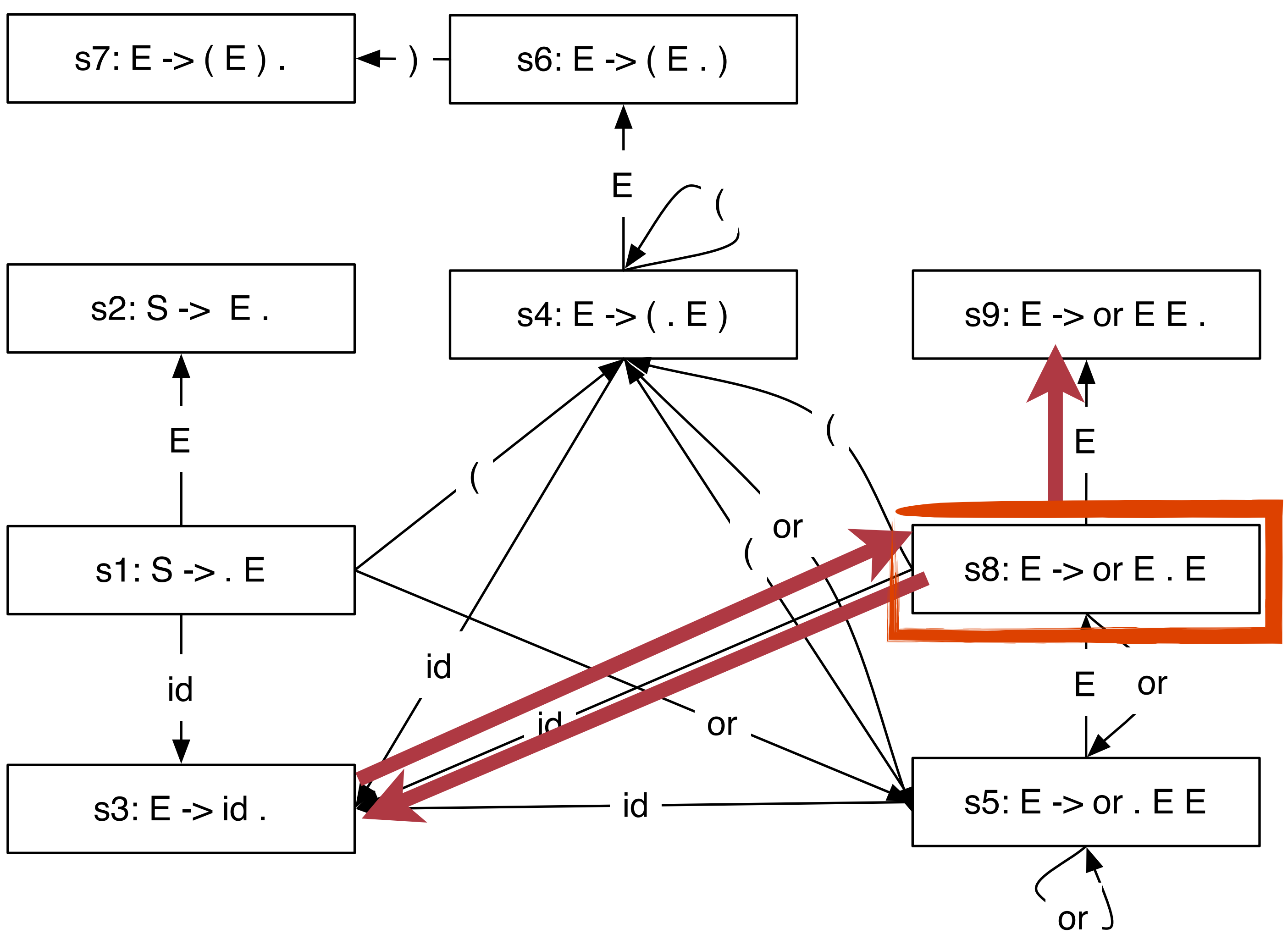
Part of the LR(0) parsing controller for the safety grammar

1st Iteration

Certificate Generation with Example

Semantic Equation

$$T = \mathit{fix}\,\lambda T.\lambda s.(PA(s, \mathtt{a}) \cup \lambda P.PA(P,)) \circ \lambda P.\mathit{reduce}(T(\mathit{top}(P))@\mathit{tail}(P)) \circ \lambda P.PA(P, ()s)$$



$s_8 \mapsto s_9 s_8 \cup$

Part of the LR(0) parsing controller for the safety grammar

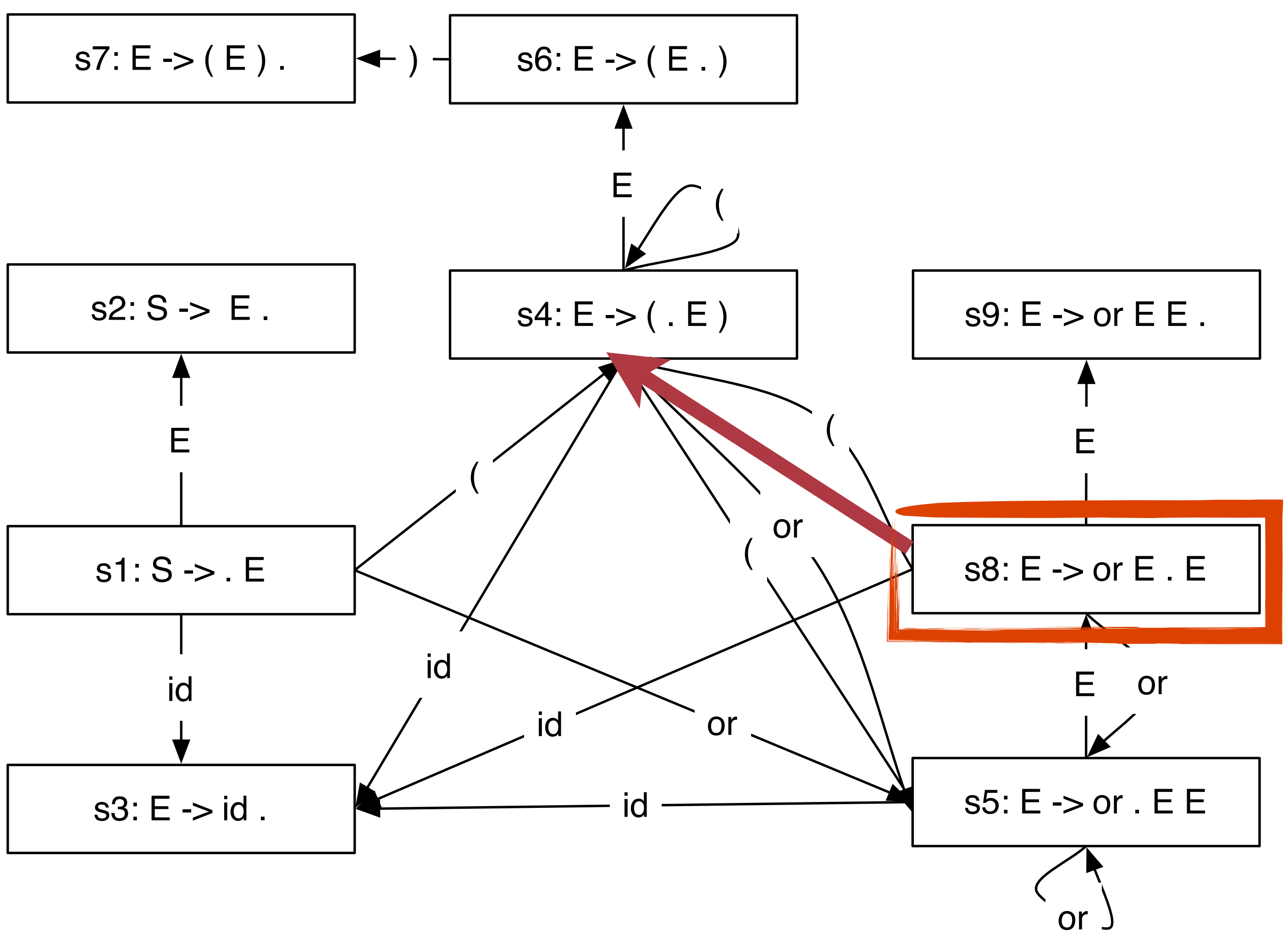
1st Iteration

Certificate Generation with Example

Semantic Equation

$$T = \mathit{fix}\lambda T.\lambda s.(PA(s, \mathtt{a}) \cup$$
$$\lambda P.PA(P, \,)\,) \circ \lambda P.\mathit{reduce}(T(\mathit{top}(P))@\mathit{tail}(P)) \circ \lambda P.PA(P, (\,)s)$$

$$s_8 \longrightarrow s_4 s_8$$



$$s_8 \mapsto s_9 s_8 \cup$$

Part of the LR(0) parsing controller for the safety grammar

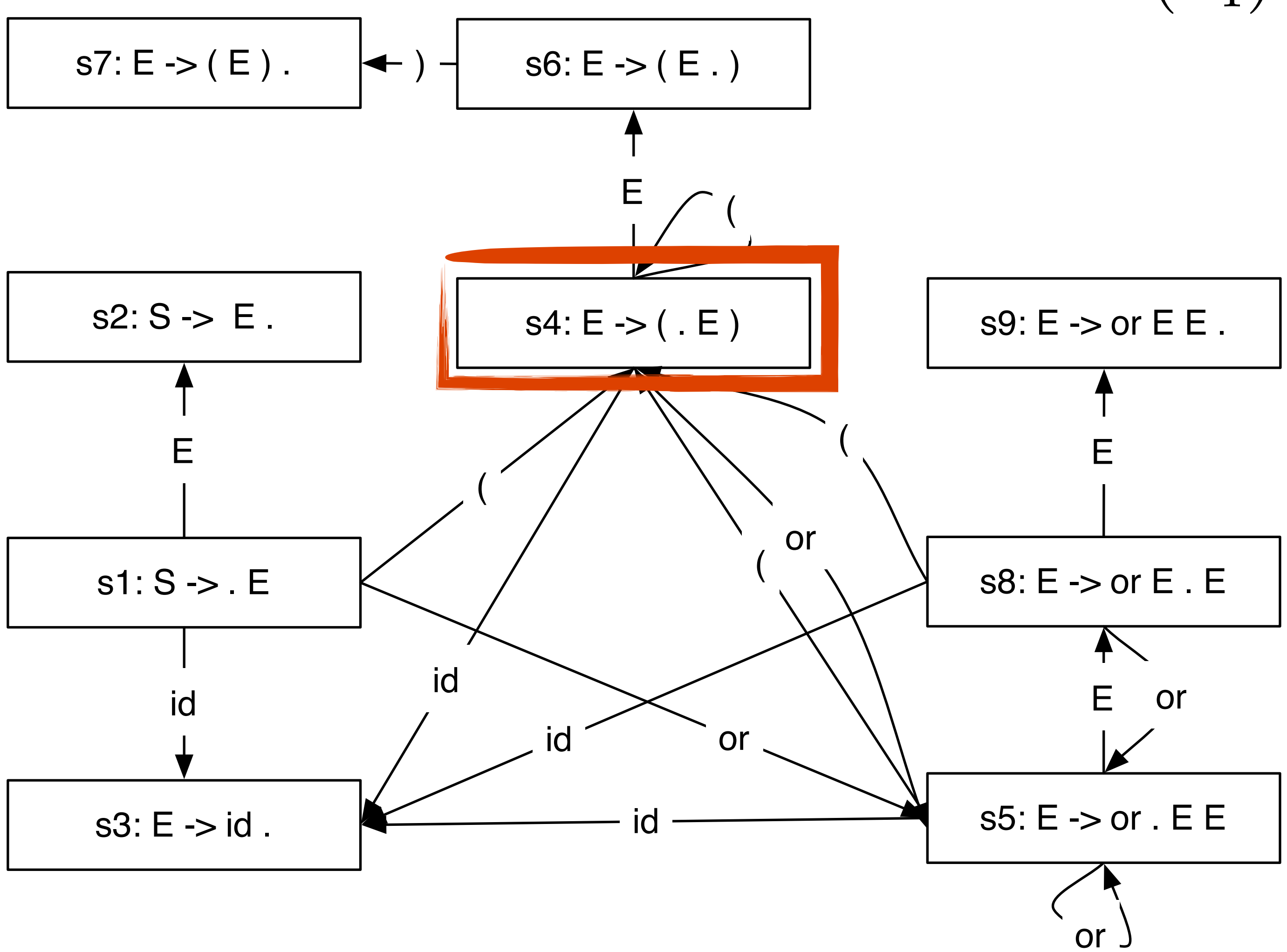
1st Iteration

Certificate Generation with Example

Semantic Equation

$$T = fix \lambda T.\lambda s.(PA(s, \mathtt{a}) \cup$$
$$\lambda P.PA(P,)) \circ \lambda P.reduce(T(top(P))@tail(P)) \circ \lambda P.PA(P, (\,)s)$$

$$T(s_4)?$$



$$s_8 \mapsto s_9 s_8 \cup$$

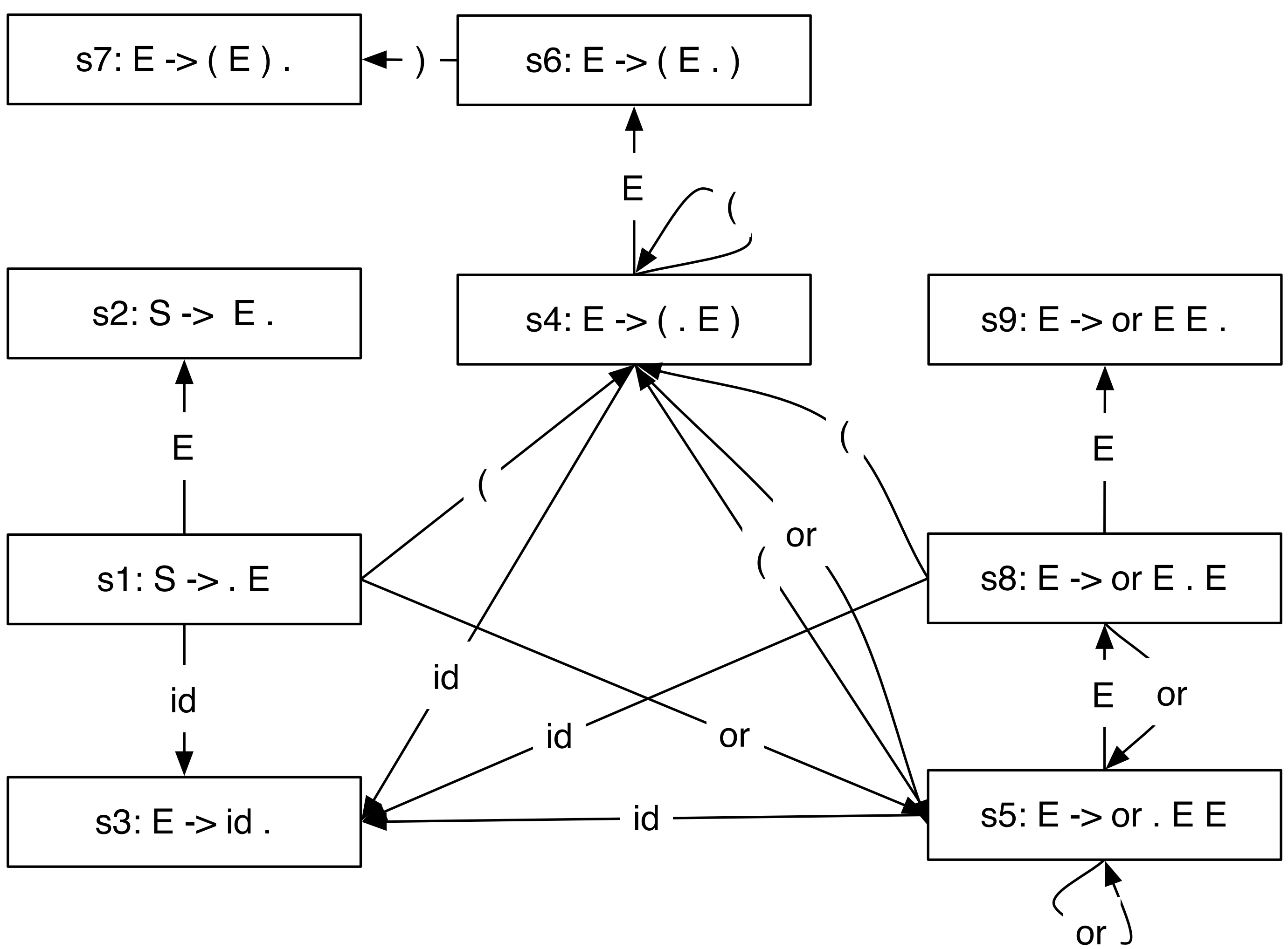
Part of the LR(0) parsing controller for the safety grammar

1st Iteration

Certificate Generation with Example

Semantic Equation

$$T = \mathit{fix}\lambda T.\lambda s.(PA(s,\mathtt{a})\cup \lambda P.PA(P,\mathtt{)}) \circ \lambda P.\mathit{reduce}(T(\mathit{top}(P))@\mathit{tail}(P)) \circ \lambda P.PA(P,\mathtt{(})s)$$



Part of the LR(0) parsing controller for the safety grammar

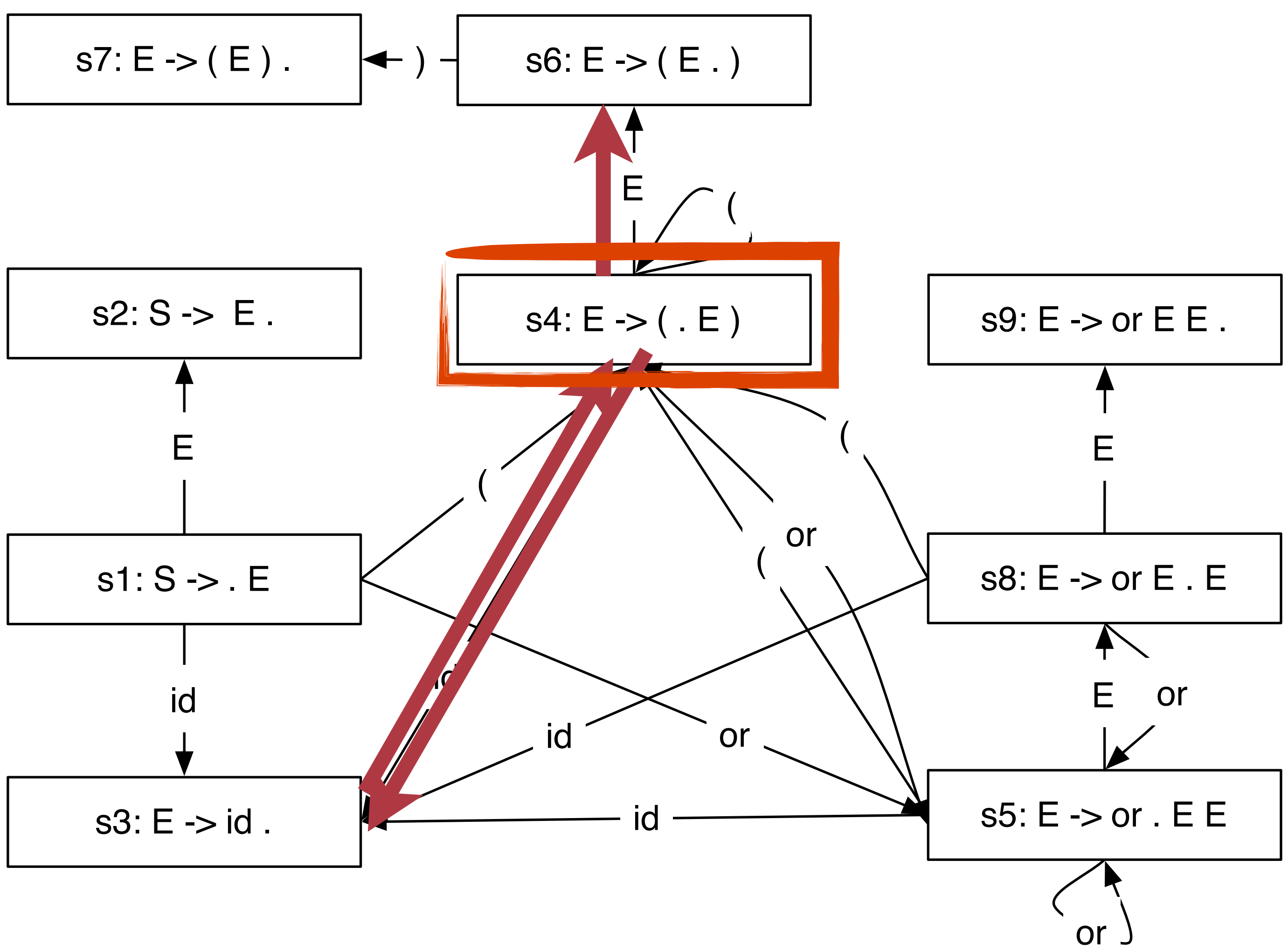
$$s_8 \mapsto s_9 s_8$$

1st Iteration

Certificate Generation with Example

Semantic Equation

$$T = fix \lambda T.\lambda s.(PA(s, \mathtt{a}) \cup \lambda P.PA(P,)) \circ \lambda P.reduce(T(top(P))@tail(P)) \circ \lambda P.PA(P, ()s)$$



Part of the LR(0) parsing controller for the safety grammar

$s_8 \mapsto s_9 s_8$

$s_4 \mapsto s_6 s_4 \cup$

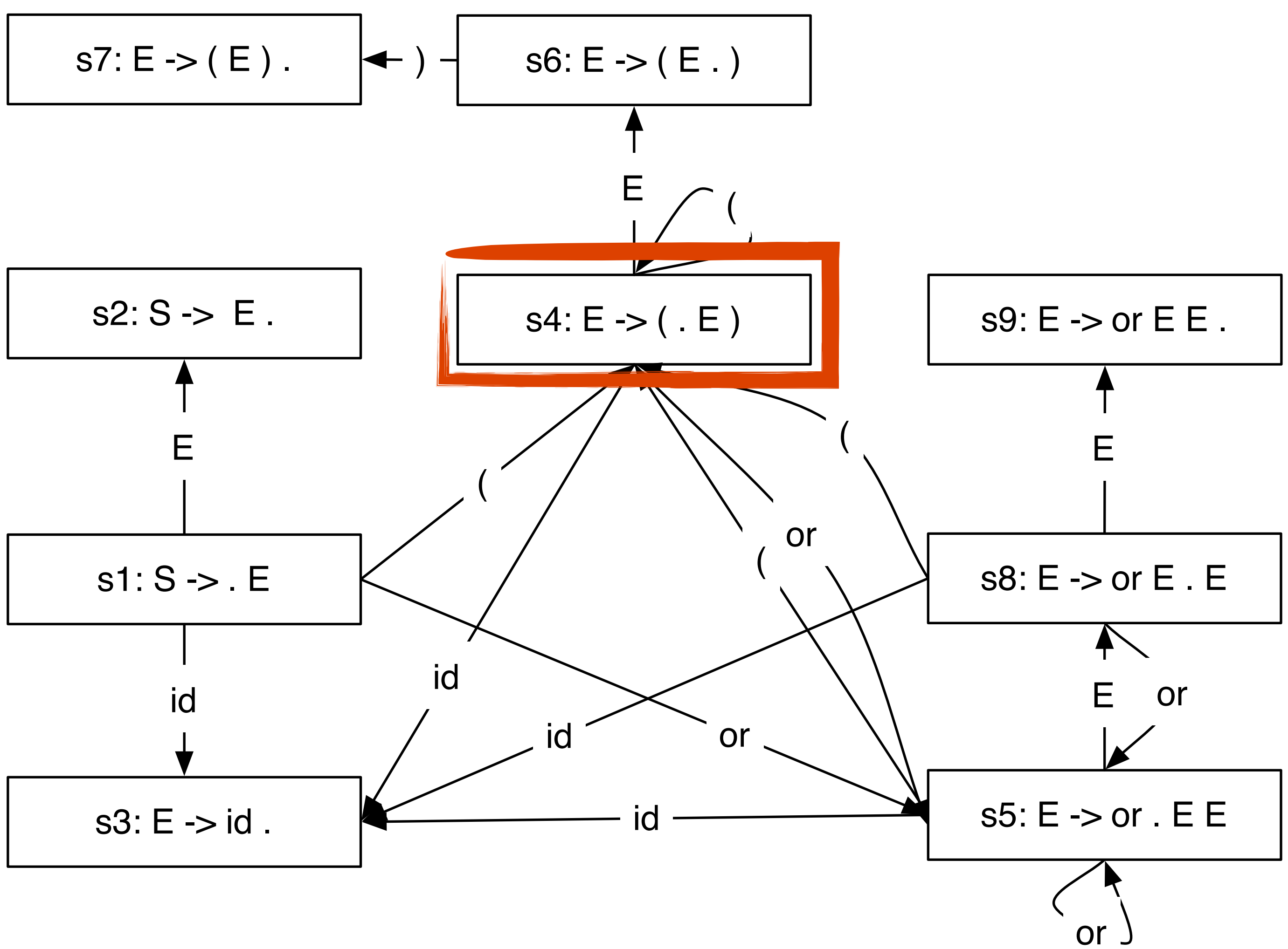
1st Iteration

Certificate Generation with Example

Semantic Equation

$$T = fix\,\lambda T.\lambda s.(PA(s, \mathsf{a}) \cup$$
$$\lambda P.PA(P,)) \circ \lambda P.reduce(T(top(P))@tail(P)) \circ \lambda P.PA(P, ()s)$$

$$s_4 \longrightarrow s_4 s_4$$



Part of the LR(0) parsing controller for the safety grammar

$$s_8 \mapsto s_9 s_8$$

$$s_4 \mapsto s_6 s_4 \cup$$

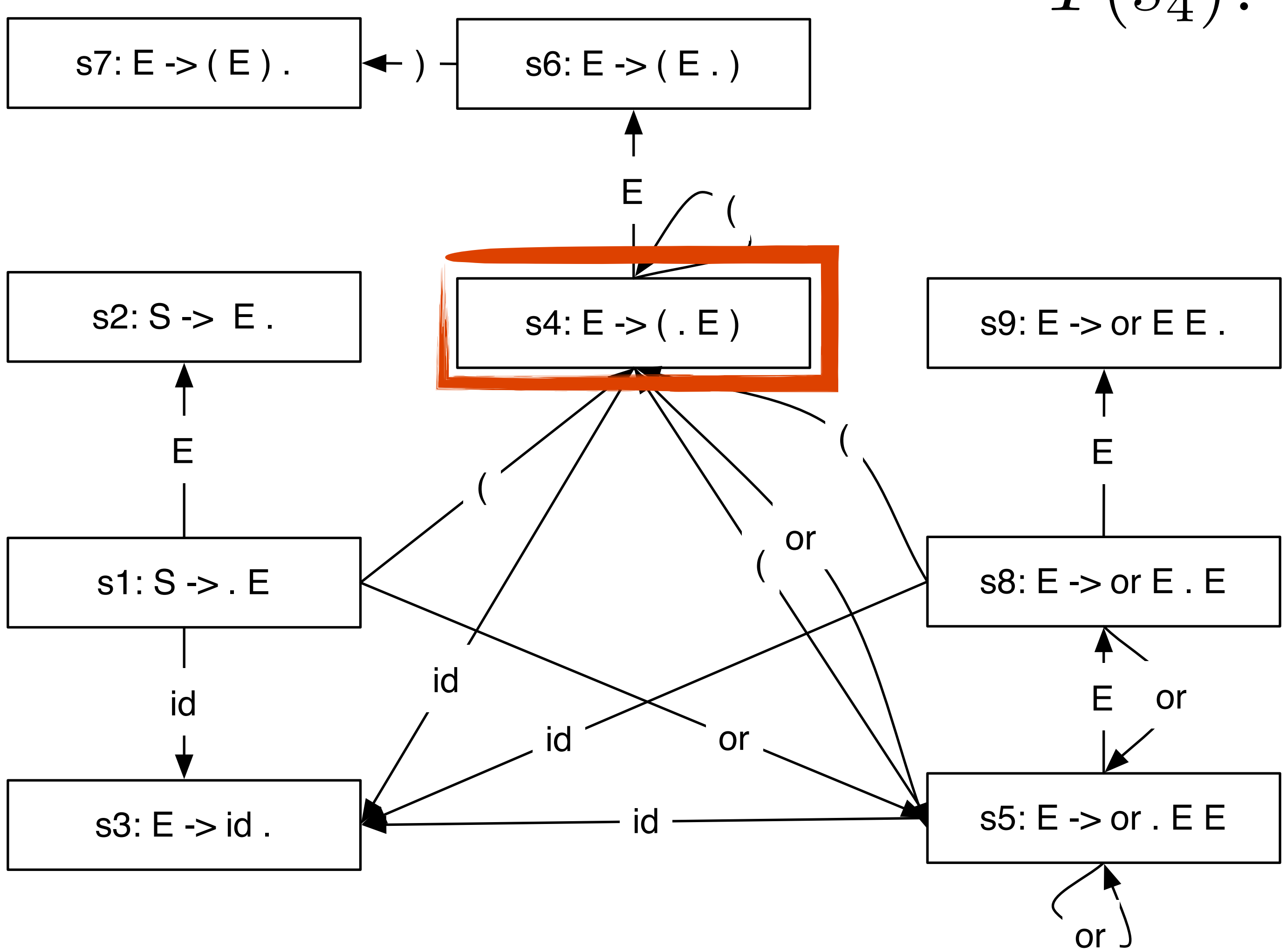
1st Iteration

Certificate Generation with Example

Semantic Equation

$$T = \mathit{fix}\,\lambda T.\lambda s.(PA(s, \mathtt{a}) \cup$$
$$\lambda P.PA(P, \mathtt{)}) \circ \lambda P.reduce(T(top(P))@tail(P)) \circ \lambda P.PA(P, \mathtt{(}\,)s)$$

$$T(s_4)?$$



$$s_8 \mapsto s_9 s_8$$

$$s_4 \mapsto s_6 s_4$$

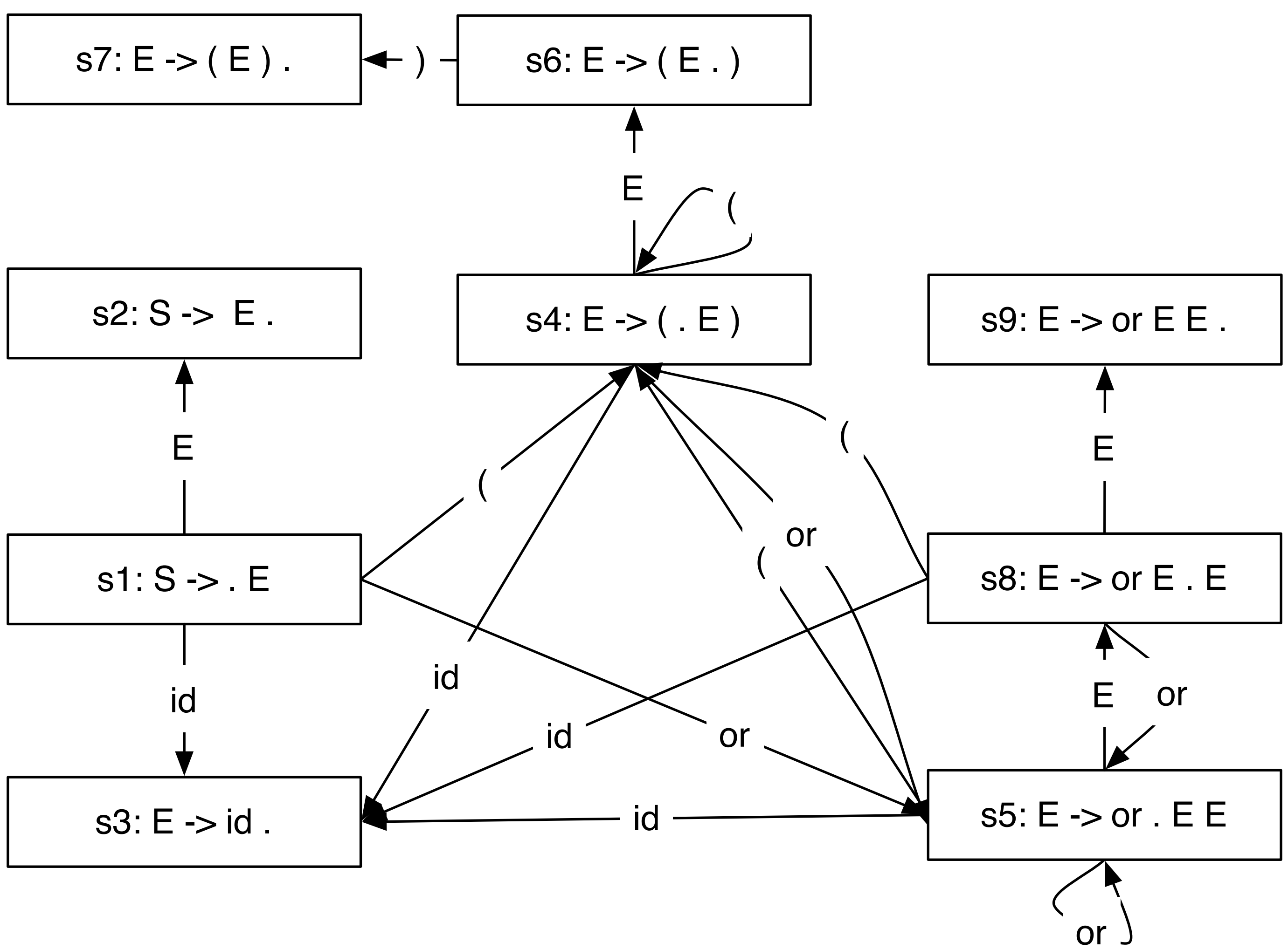
Part of the LR(0) parsing controller for the safety grammar

1st Iteration

Certificate Generation with Example

Semantic Equation

$$T = \mathit{fix}\lambda T.\lambda s.(PA(s,\mathtt{a})\cup \lambda P.PA(P,\mathtt{)})\circ \lambda P.\mathit{reduce}(T(\mathit{top}(P))@\mathit{tail}(P))\circ \lambda P.PA(P,\mathtt{(})s)$$



$$s_8 \mapsto s_9 s_8$$

$$s_4 \mapsto s_6 s_4$$

Part of the LR(0) parsing controller for the safety grammar

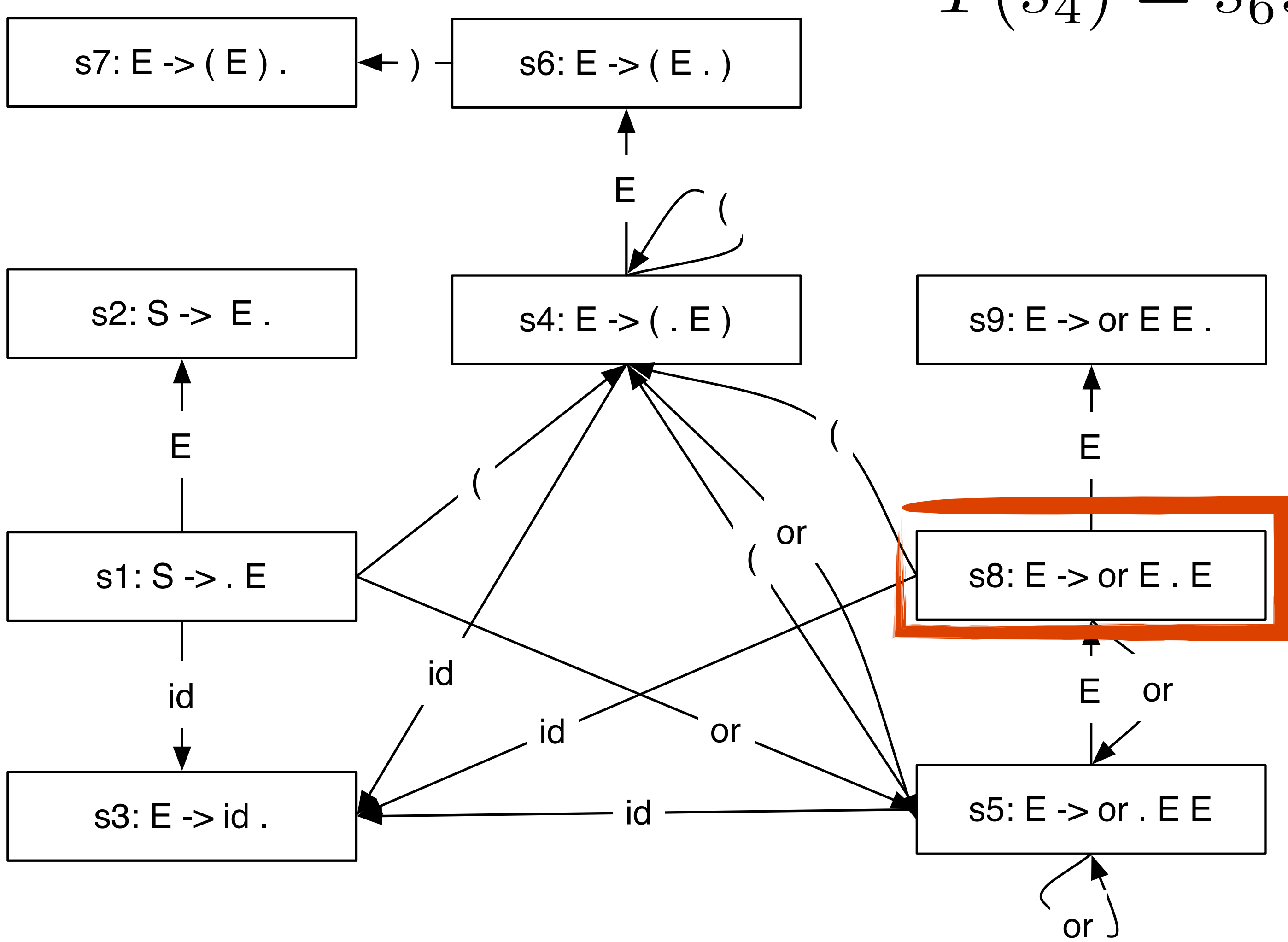
1st Iteration Done.

Certificate Generation with Example

Semantic Equation

$$T = fix \lambda T.\lambda s.(PA(s, \mathtt{a}) \cup$$
$$\lambda P.PA(P,)) \circ \lambda P.reduce(T(top(P))@\ ail(P)) \circ \lambda P.PA(P, ()s)$$

$$T(s_4) = s_6 s_4 \qquad s_8 \longrightarrow s_4 s_8$$



$$s_8 \longmapsto s_9 s_8$$

$$s_4 \longmapsto s_6 s_4$$

Part of the LR(0) parsing controller for the safety grammar

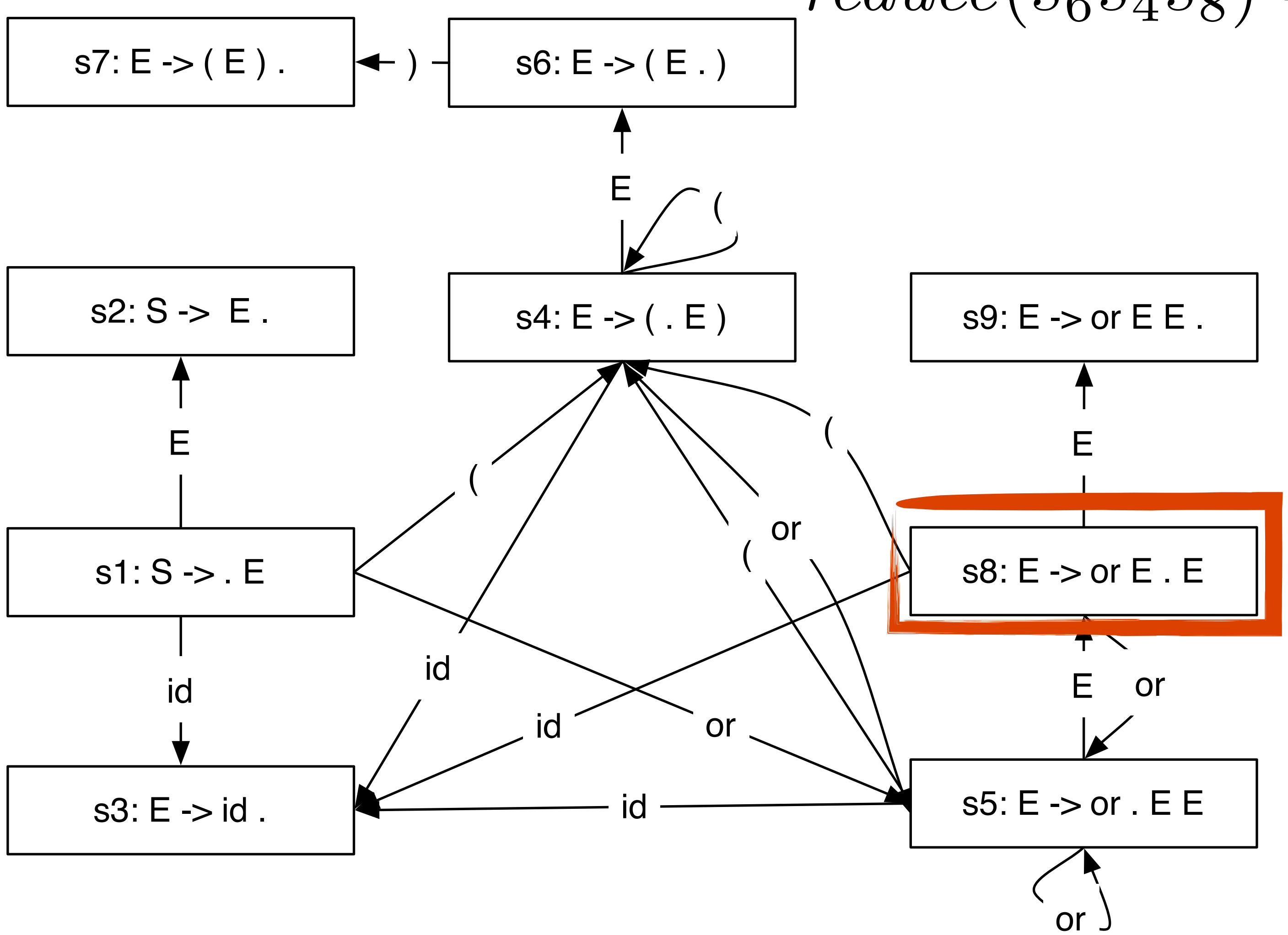
2nd Iteration

Certificate Generation with Example

Semantic Equation

$$T = \mathit{fix}\,\lambda T.\lambda s.(PA(s, \mathsf{a}) \cup$$
$$\lambda P.PA(P, \,)\,) \circ \lambda P.\mathit{reduce}(T(\mathit{top}(P))@\mathit{tail}(P)) \circ \lambda P.PA(P, (\,)s)$$

$$\mathit{reduce}(s_6 s_4 s_8) = s_6 s_4 s_8 \qquad s_8 \longrightarrow s_4 s_8$$



$$s_8 \longmapsto s_9 s_8$$

$$s_4 \longmapsto s_6 s_4$$

Part of the LR(0) parsing controller for the safety grammar

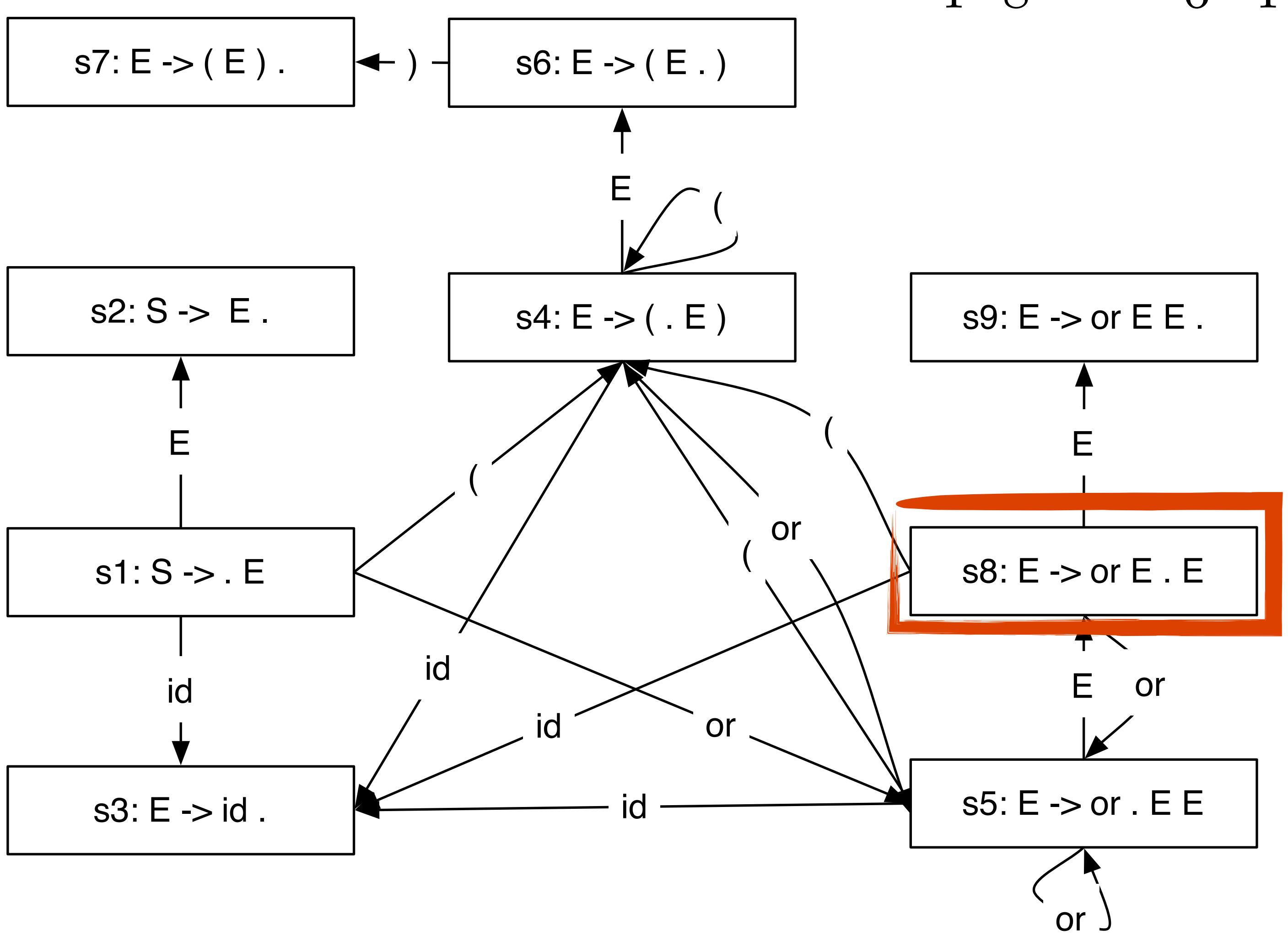
2nd Iteration

Certificate Generation with Example

Semantic Equation

$$T = fix\lambda T.\lambda s.(PA(s, \mathsf{a}) \cup$$
$$\lambda P.PA(P,)) \circ \lambda P.reduce(T(top(P))@tail(P)) \circ \lambda P.PA(P, ()s)$$

$s_4 s_8 \longrightarrow s_6 s_4 s_8$ $\quad$ $s_8 \longrightarrow s_4 s_8$



$s_8 \longmapsto s_9 s_8$

$s_4 \longmapsto s_6 s_4$

Part of the LR(0) parsing controller for the safety grammar

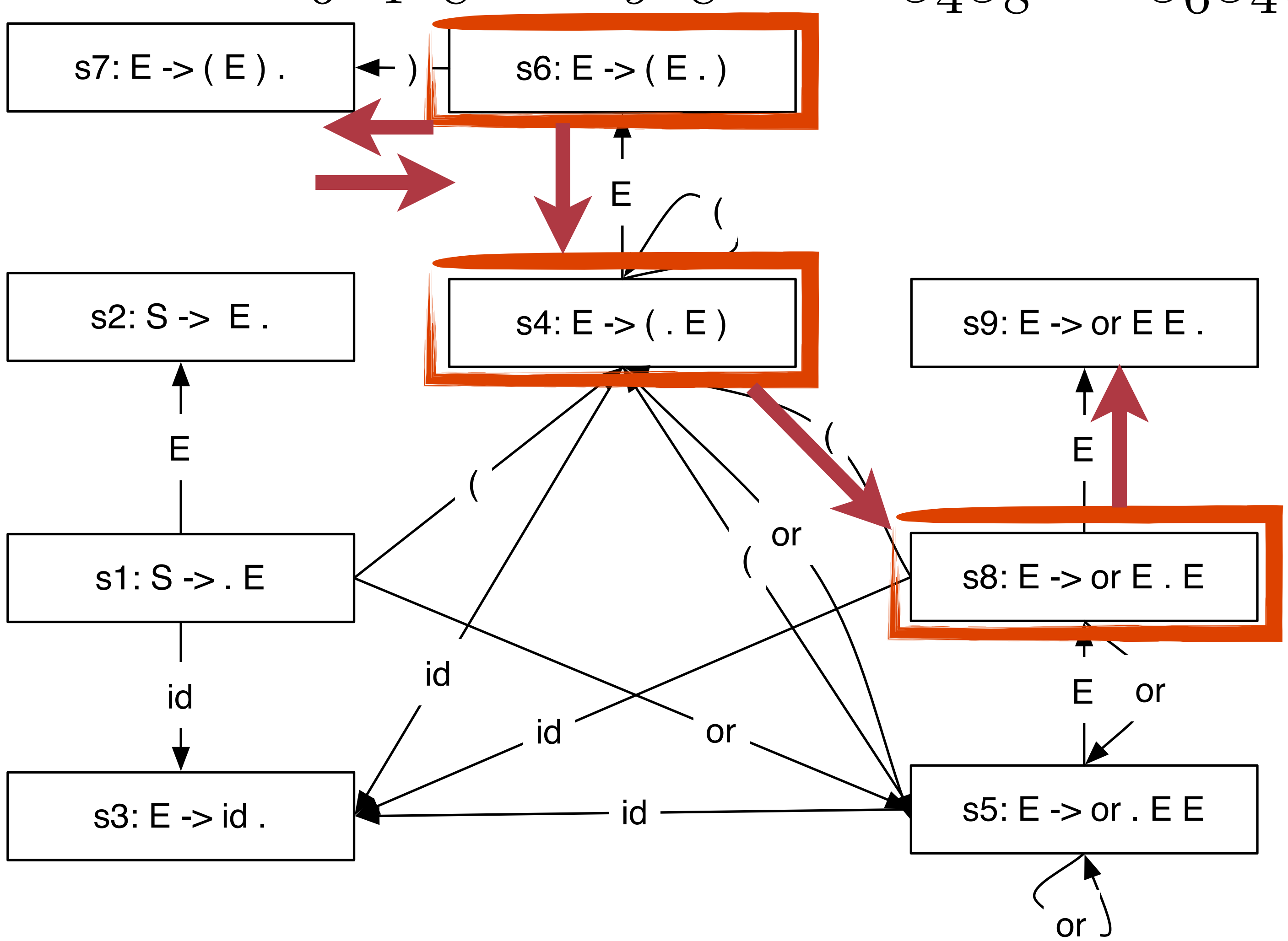
2nd Iteration

Certificate Generation with Example

Semantic Equation

$$T = fix \lambda T.\lambda s.(PA(s, \mathtt{a}) \cup$$
$$\lambda P.PA(P, \mathtt{)}) \circ \lambda P.reduce(T(top(P))@tail(P)) \circ \lambda P.PA(P, \mathtt{(})s)$$

$s_6 s_4 s_8 \longrightarrow s_9 s_8$ $\quad s_4 s_8 \longrightarrow s_6 s_4 s_8$ $\quad s_8 \longrightarrow s_4 s_8$



$s_8 \longmapsto s_9 s_8$

$s_4 \longmapsto s_6 s_4$

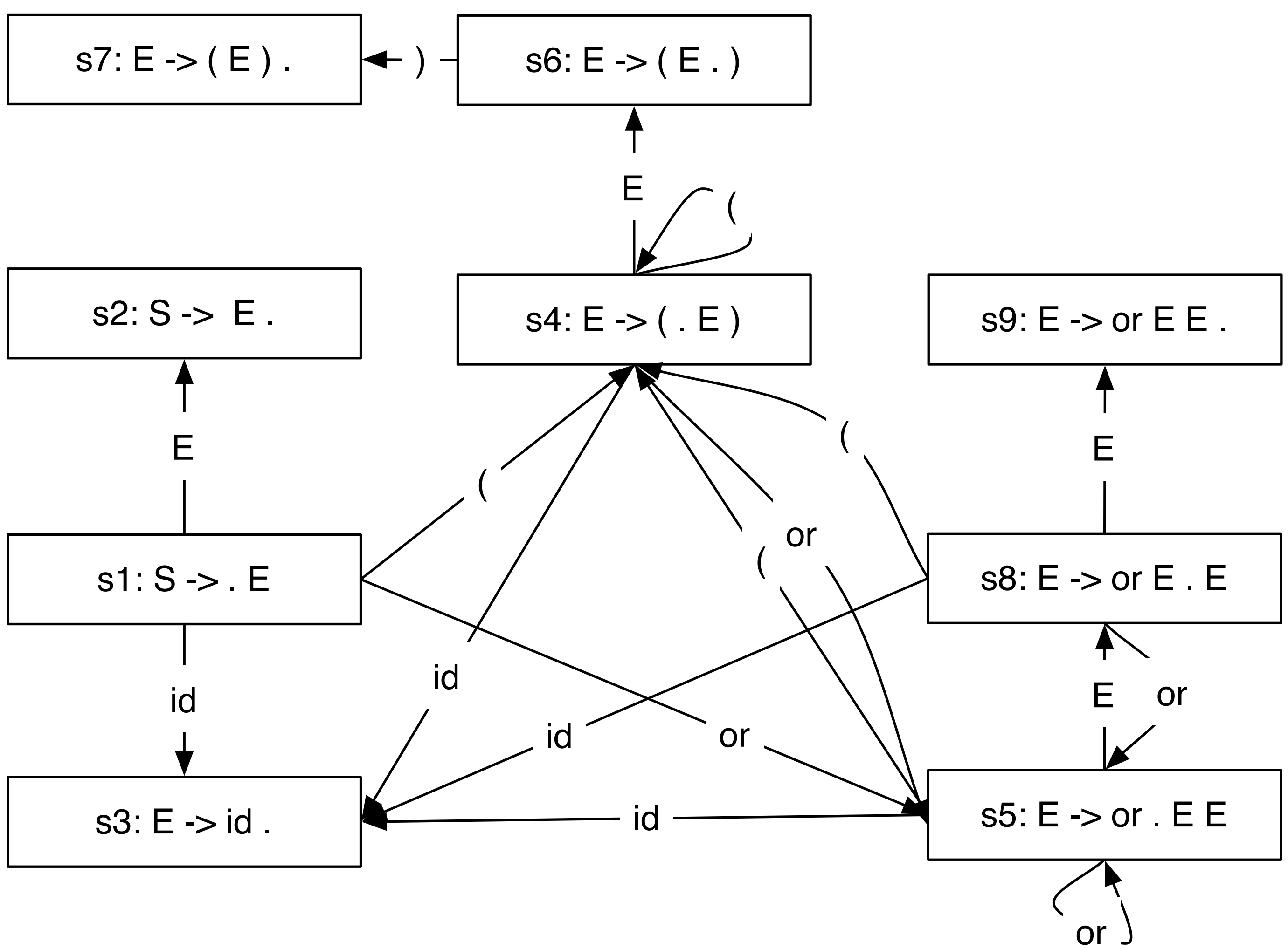
Part of the LR(0) parsing controller for the safety grammar

2nd Iteration

Certificate Generation with Example

Semantic Equation

$$T = \mathit{fix}\,\lambda T.\lambda s.(PA(s, \mathtt{a}) \cup \lambda P.PA(P, \mathtt{)}) \circ \lambda P.\mathit{reduce}(T(\mathit{top}(P))@\mathit{tail}(P)) \circ \lambda P.PA(P, \mathtt{(})s)$$



Not Changed!

$$s_8 \mapsto s_9 s_8$$

$$s_4 \mapsto s_6 s_4$$

Part of the LR(0) parsing controller for the safety grammar

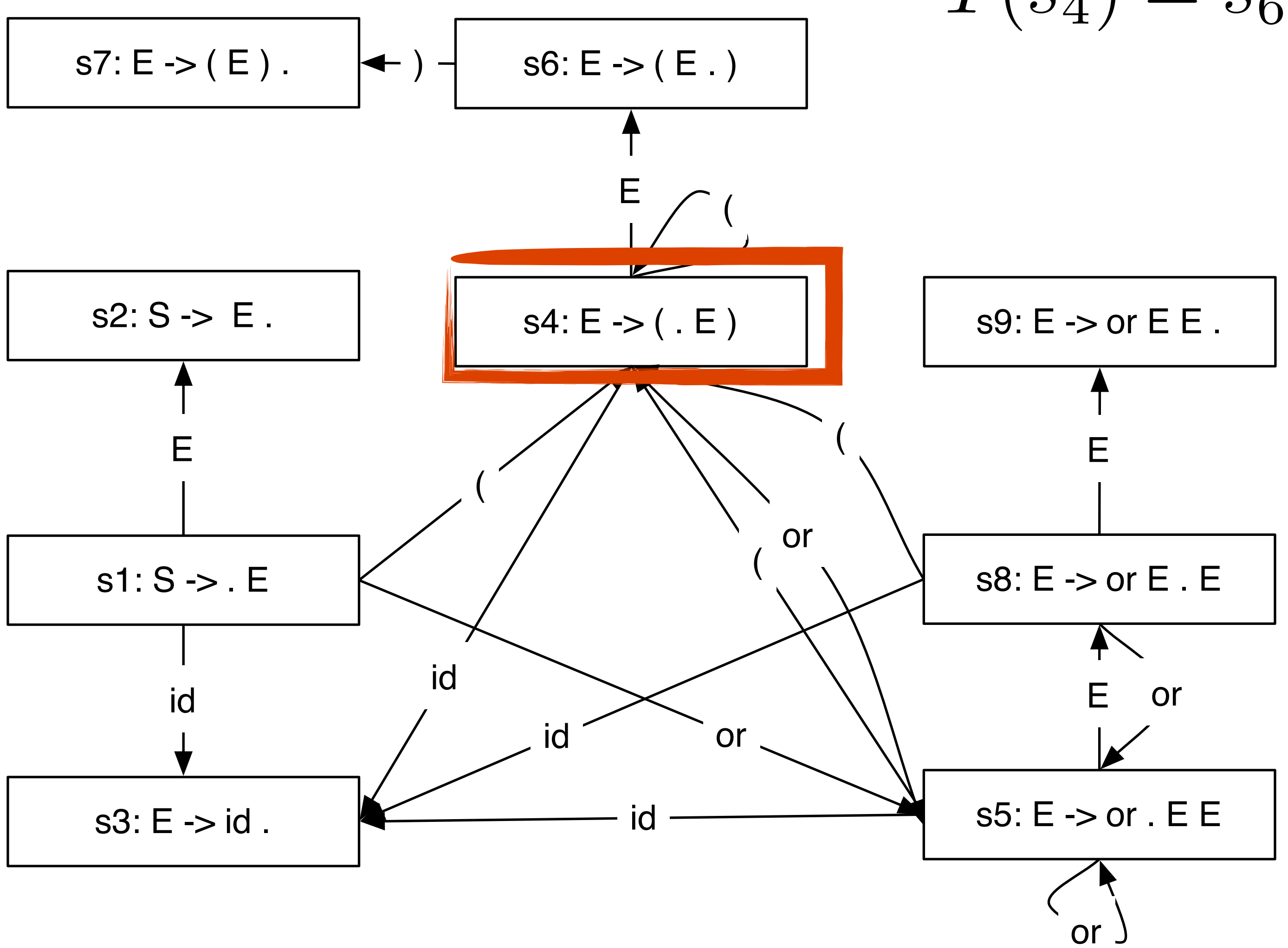
2nd Iteration

Certificate Generation with Example

Semantic Equation

$$T = \mathit{fix}\,\lambda T.\lambda s.(PA(s, \mathtt{a}) \cup$$
$$\lambda P.PA(P, \mathtt{)}) \circ \lambda P.reduce(T(top(P))@tail(P)) \circ \lambda P.PA(P, \mathtt{(})s)$$

$$T(s_4) = s_6 s_4 \qquad s_4 \longrightarrow s_4 s_4$$



$$s_8 \longmapsto s_9 s_8$$

$$s_4 \longmapsto s_6 s_4$$

Part of the LR(0) parsing controller for the safety grammar

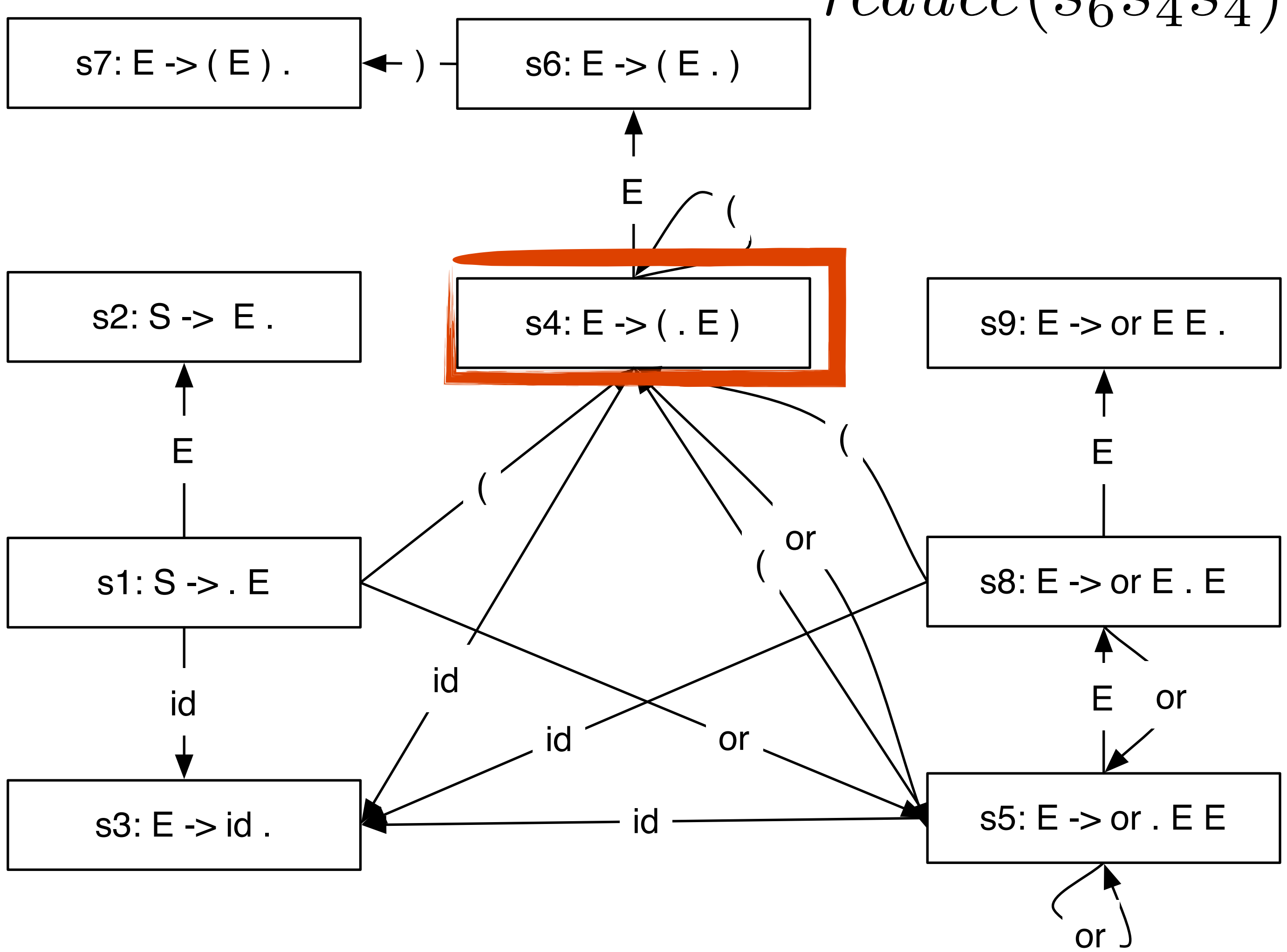
2nd Iteration

Certificate Generation with Example

Semantic Equation

$$T = \mathit{fix}\,\lambda T.\lambda s.(PA(s, \mathsf{a})\cup$$
$$\lambda P.PA(P,)) \circ \lambda P.reduce(T(top(P))@tail(P)) \circ \lambda P.PA(P, ()s)$$

$reduce(s_6 s_4 s_4) = s_6 s_4 s_4 \quad s_4 \longrightarrow s_4 s_4$



$s_8 \longmapsto s_9 s_8$

$s_4 \longmapsto s_6 s_4$

Part of the LR(0) parsing controller for the safety grammar

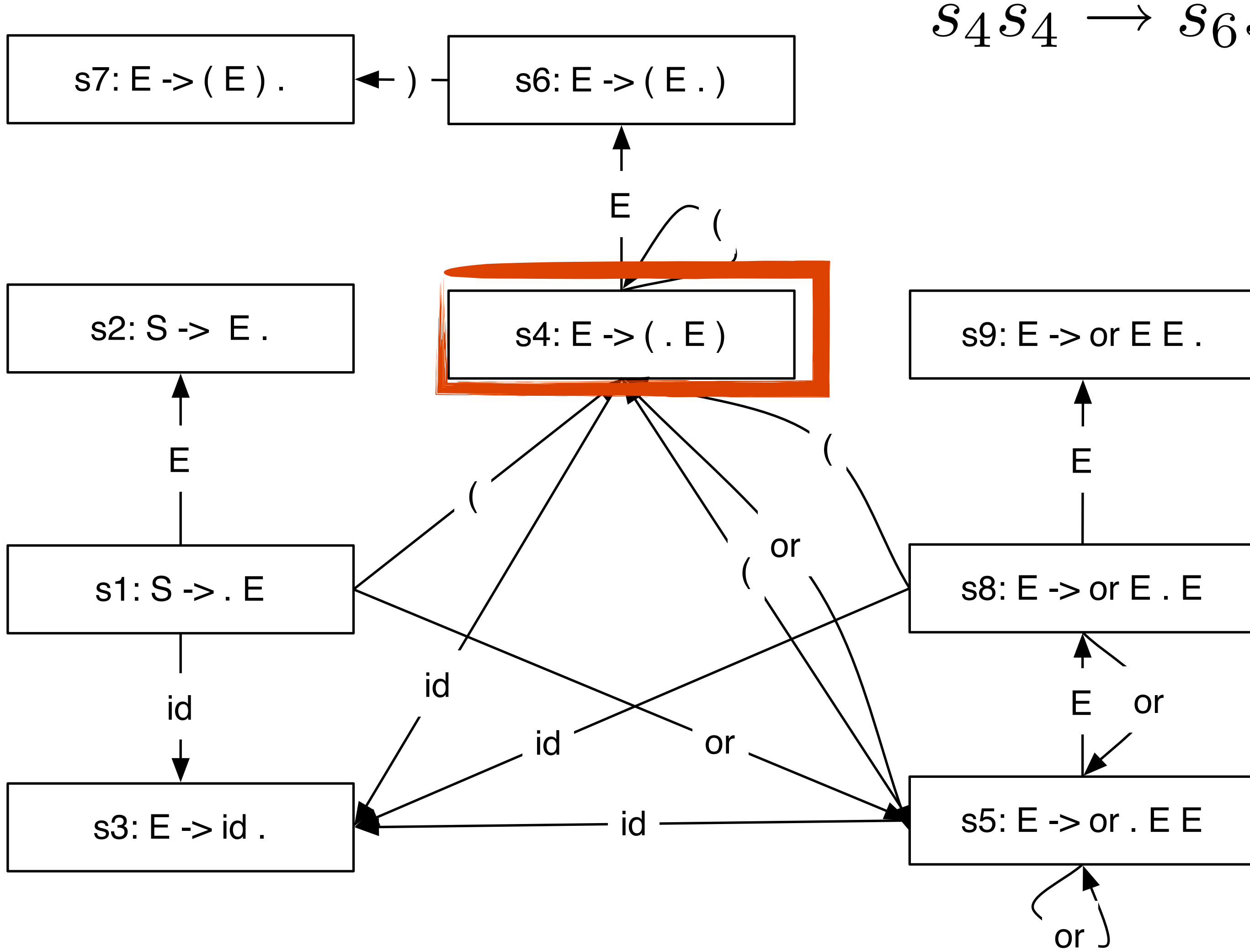
2nd Iteration

Certificate Generation with Example

Semantic Equation

$$T = fix \lambda T.\lambda s.(PA(s, \mathsf{a}) \cup$$
$$\lambda P.PA(P,)) \circ \lambda P.reduce(T(top(P))@tail(P)) \circ \lambda P.PA(P, ()s)$$

$s_4 s_4 \longrightarrow s_6 s_4 s_4$ $\quad$ $s_4 \longrightarrow s_4 s_4$



$s_8 \longmapsto s_9 s_8$

$s_4 \longmapsto s_6 s_4$

Part of the LR(0) parsing controller for the safety grammar

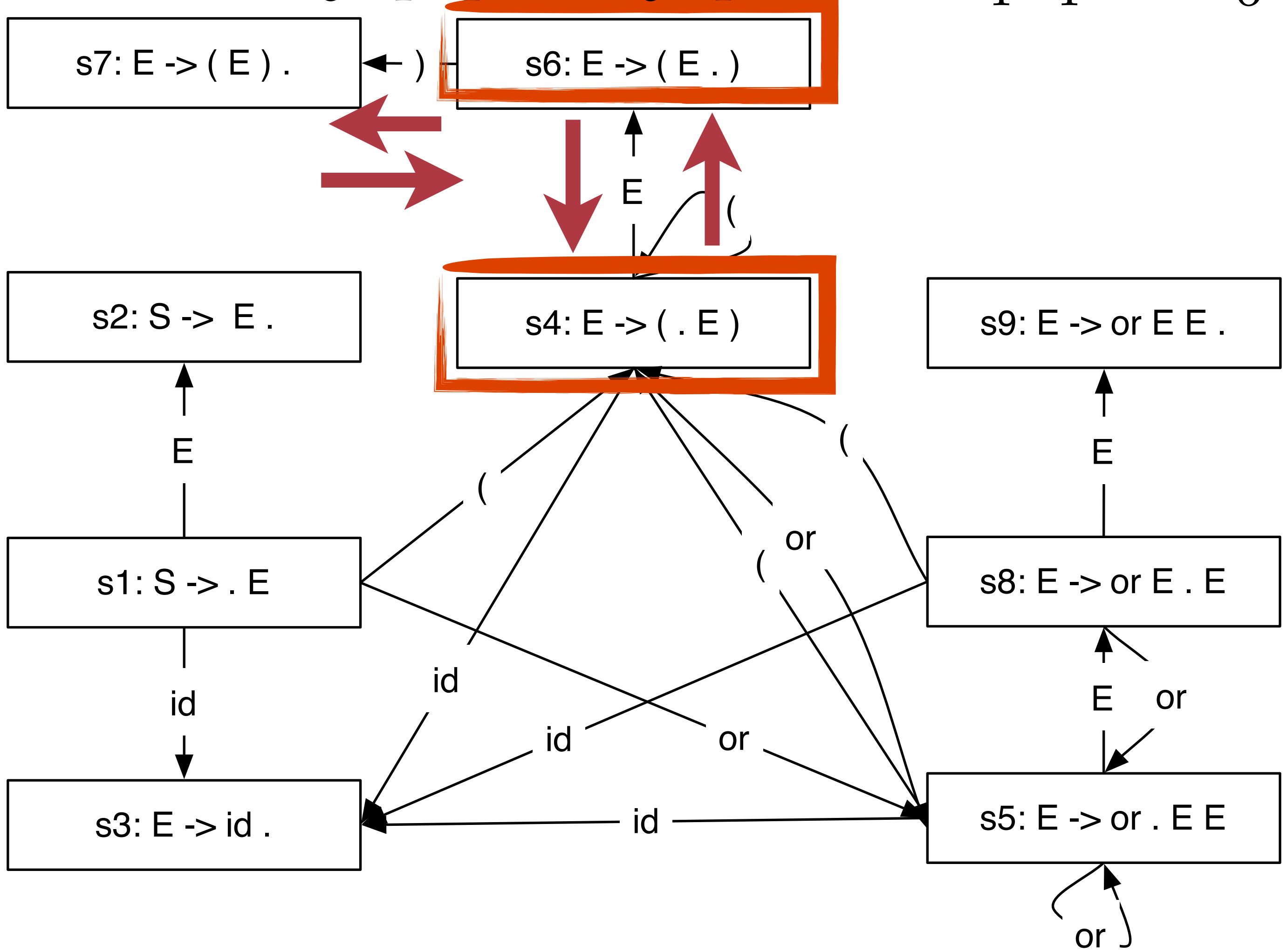
2nd Iteration

Certificate Generation with Example

Semantic Equation

$$T = \mathit{fix}\,\lambda T.\lambda s.(PA(s, \mathsf{a}) \cup$$
$$\lambda P.PA(P,)) \circ \lambda P.\mathit{reduce}(T(\mathit{top}(P))@\mathit{tail}(P)) \circ \lambda P.PA(P, ()s)$$

$s_6 s_4 s_4 \longrightarrow s_6 s_4$ $\quad$ $s_4 s_4 \longrightarrow s_6 s_4 s_4$ $\quad$ $s_4 \longrightarrow s_4 s_4$



Not Changed!

$s_8 \longmapsto s_9 s_8$

$s_4 \longmapsto s_6 s_4$

Not Changed!

Part of the LR(0) parsing controller for the safety grammar

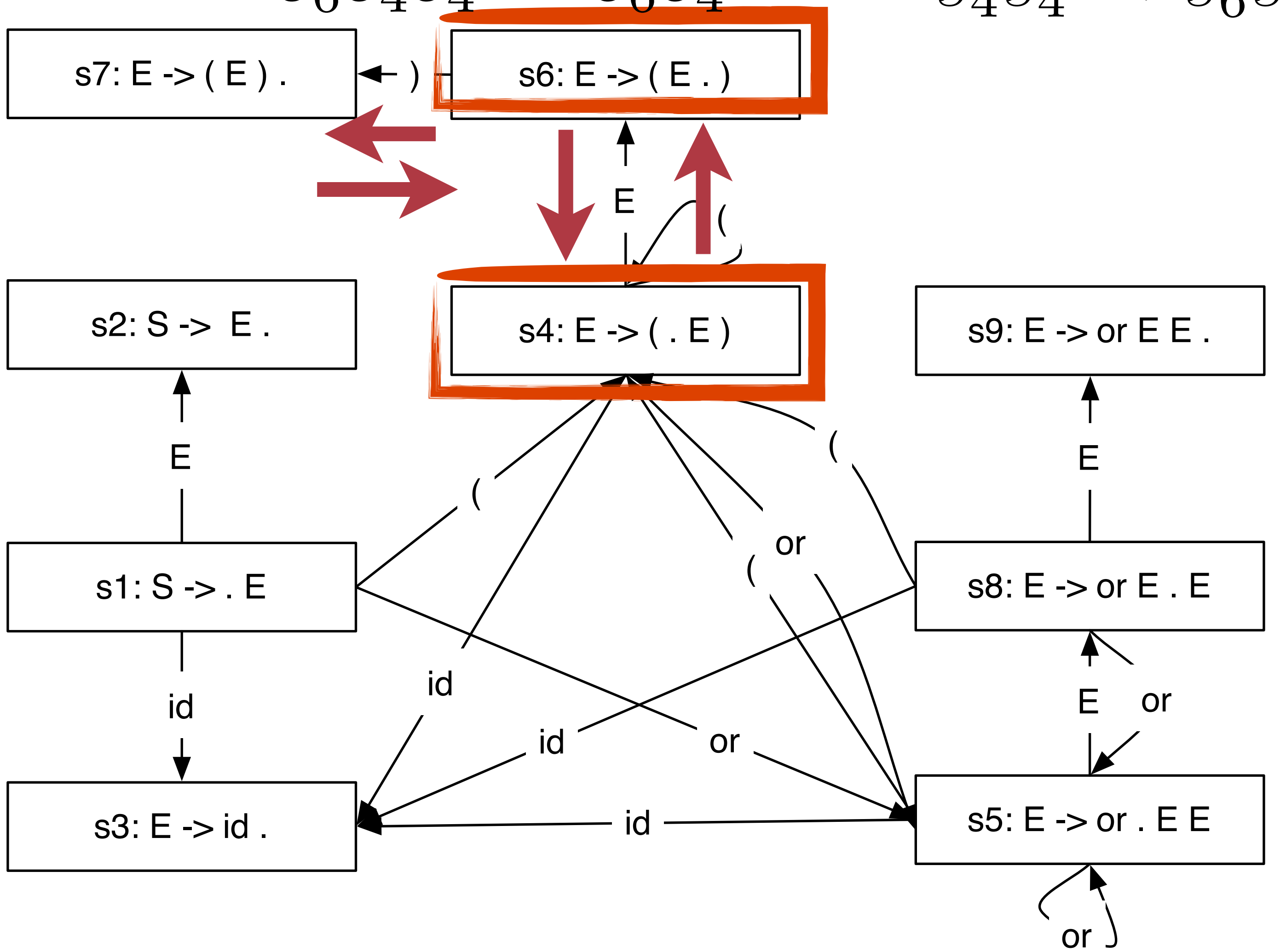
2nd Iteration

Certificate Generation with Example

Semantic Equation

$$T = \mathit{fix}\, \lambda T.\lambda s.(PA(s, \mathtt{a}) \cup \lambda P.PA(P,)) \circ \lambda P.\mathit{reduce}(T(\mathit{top}(P))@\mathit{tail}(P)) \circ \lambda P.PA(P, ()s)$$

$s_6 s_4 s_4 \longrightarrow s_6 s_4$ $\quad$ $s_4 s_4 \longrightarrow s_6 s_4 s_4$ $\quad$ $s_4 \longrightarrow s_4 s_4$



Fixed Point

$s_8 \longmapsto s_9 s_8$

$s_4 \longmapsto s_6 s_4$

Part of the LR(0) parsing controller for the safety grammar

2nd Iteration

Certificate Generation with Example

- Code producer sends the program and computed fixed-point solution.

Program

```
re x `a
   `( . ,x . )
   let y
       `or . a
       `,y . ,x
```

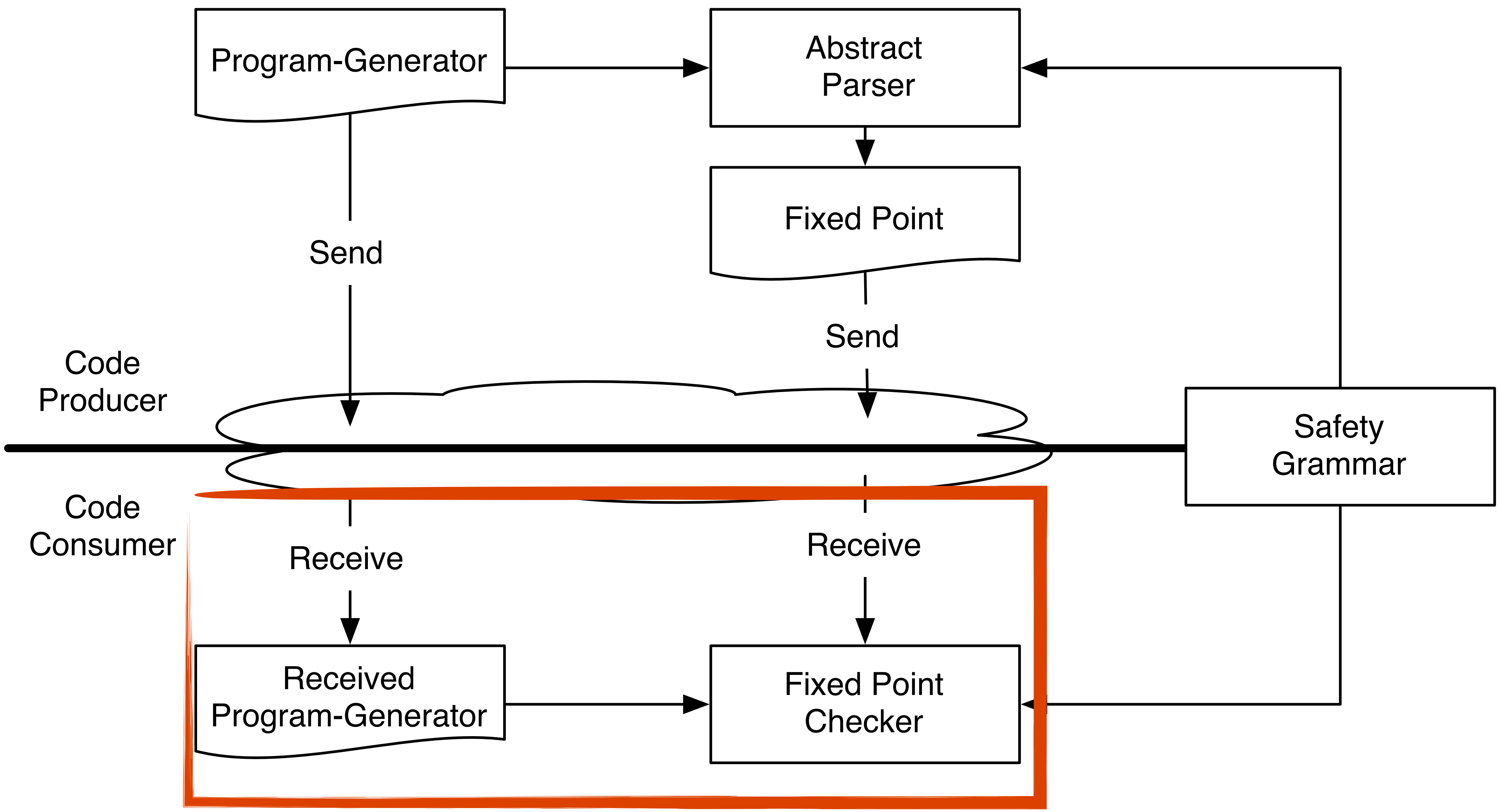
+

Fixed-point Solution

$$s_8 \mapsto s_9 s_8$$

$$s_4 \mapsto s_6 s_4$$

Certificate Check



Certificate Check with Example

Received Program

```
re x `a
   `( . ,x . )
   let y
      `or . a
      `,y . ,x
```

1. From the received program, derive semantic equations.

Semantic Equation

$$P = X \circ Y$$

$$Y = \lambda P.PA(PA(P, \texttt{or}), \texttt{a})$$

$$T = \mathit{fix}\lambda T.\lambda s.(PA(s, \texttt{a}) \cup$$
$$\lambda P.PA(P, \texttt{)}) \circ \lambda P.\mathit{reduce}(T(\mathit{top}(P))@\mathit{tail}(P)) \circ \lambda P.PA(P, \texttt{(})s)$$

$$X = \lambda P.\mathit{reduce}(T(\mathit{top}(P))@\mathit{tail}(P))$$

Certificate Check with Example

Received Fixed-point solution

$$s_8 \mapsto s_9 s_8$$

$$s_4 \mapsto s_6 s_4$$

2. Check that the received solution is indeed the fixed-point for the program. (one iteration is enough)

Semantic Equation

$$P = X \circ Y$$

$$Y = \lambda P.PA(PA(P, \texttt{or}), \texttt{a})$$

$$T = \mathit{fix}\, \lambda T.\lambda s.(PA(s, \texttt{a}) \cup \lambda P.PA(P, \texttt{)}) \circ \lambda P.\mathit{reduce}(T(\mathit{top}(P))@\mathit{tail}(P)) \circ \lambda P.PA(P, \texttt{(})s)$$

$$X = \lambda P.\mathit{reduce}(T(\mathit{top}(P))@\mathit{tail}(P))$$

Certificate Check with Example

Received Fixed-point solution

$$s_8 \mapsto s_9 s_8$$

$$s_4 \mapsto s_6 s_4$$

Semantic Equation

$$P = X \circ Y$$

$$Y = \lambda P.PA(PA(P, \texttt{or}), \texttt{a})$$

$$T = \mathit{fix}\lambda T.\lambda s.(PA(s, \texttt{a}) \cup$$

$$\lambda P.PA(P, \texttt{)}) \circ \lambda P.\mathit{reduce}(T(\mathit{top}(P))@\mathit{tail}(P)) \circ \lambda P.PA(P, \texttt{(})s)$$

$$X = \lambda P.\mathit{reduce}(T(\mathit{top}(P))@\mathit{tail}(P))$$

---

3. Using the fixed-point solution, construct abstract parsing semantics of the program.

$$P = \{s_1 \mapsto s_1 s_2\}$$

Certificate Check with Example

Received Fixed-point solution

$s_8 \mapsto s_9 s_8$

$s_4 \mapsto s_6 s_4$

Semantic Equation

$P = X \circ Y$

$Y = \lambda P.PA(PA(P, \texttt{or}), \texttt{a})$

$T = \mathit{fix}\lambda T.\lambda s.(PA(s, \texttt{a}) \cup$

$\lambda P.PA(P, \texttt{)}) \circ \lambda P.\mathit{reduce}(T(\mathit{top}(P))@\mathit{tail}(P)) \circ \lambda P.PA(P, \texttt{(})s)$

$X = \lambda P.\mathit{reduce}(T(\mathit{top}(P))@\mathit{tail}(P))$

---

3. Using the fixed-point solution, construct abstract parsing semantics of the program.

Accept Parse Stack

$P = \{s_1 \mapsto s_1 s_2\}$

Summary

- Our framework addresses two fundamental PCC issues.

1. The certificate, a fixed-point solution, is generated <u>automatically</u> by abstract parser.
2. Checking procedure on the code consumer side is done <u>efficiently</u> by validating the received fixed-point solution.

Issues

- Two issues need further investigation.

1. Size of the certificate:

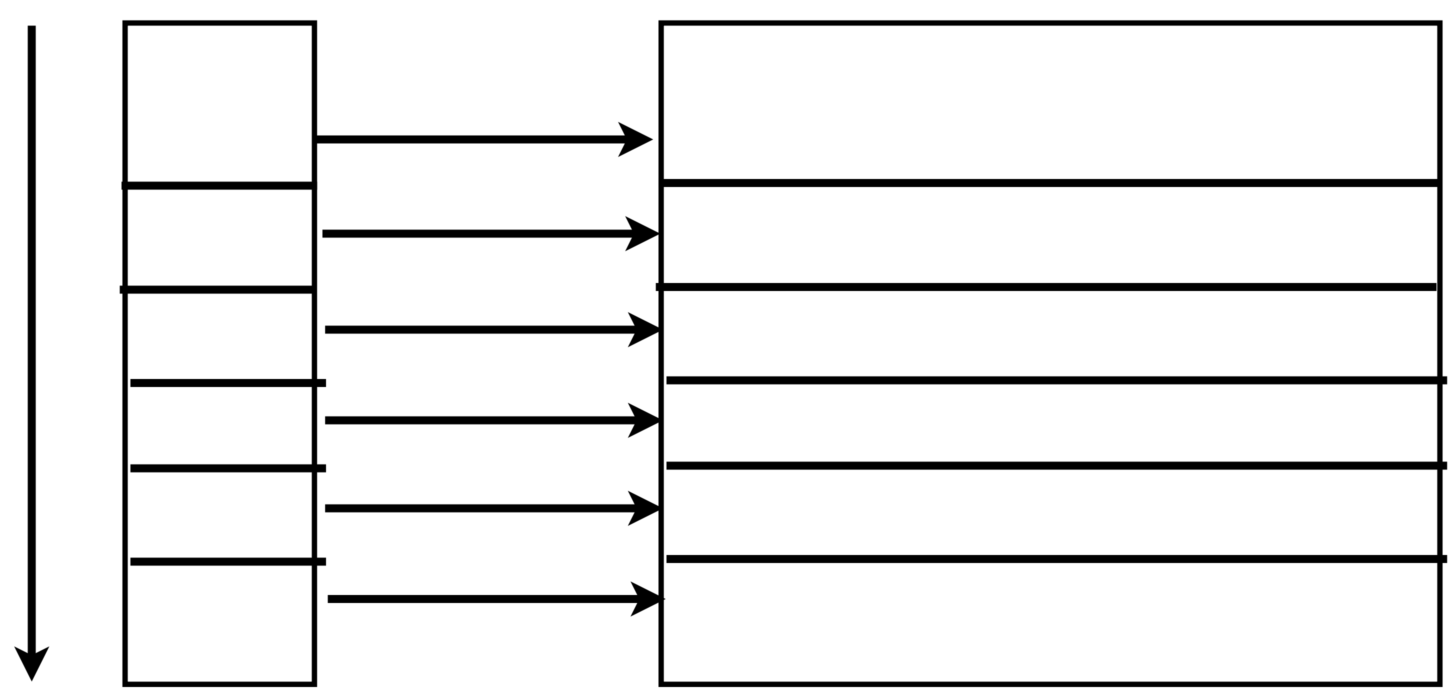
   Certificate in our framework:
   the fixed-point solution for every loop in the program.

   - O(# of loops) : linear to the program size

Issues

- Two issues need further investigation.

1. Size of the certificate:
   - O(# of parse states)
   # of parse states is fixed with the given grammar.



$$ParseState \rightarrow 2^{ParseStack}$$

Issues

- Two issues need further investigation.

  2. Complexity of the checker:
     - As complex as the certificate generator
       Need to derive the same semantic equations.
       Need to implement all the abstract operators.
     - Shared problem with other abstract-carrying code frameworks.

Future Work

- Work is in progress
  - Implement the abstract parser and do the experiment.

Thank You