① Fie specificația lexicală de mai jos:

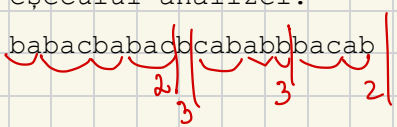
(1) babac

(2) (bac*|abc*)+

(3) (ac*b|c*ab)*b

(a) Care este secvența de reguli lexicale (1-3) utilizate în analiza următorului șir? Luați în calcul și posibilitatea eșecului analizei.

babacbabacbcababbbacab



(b) Dați un exemplu de șir care nu poate fi analizat de această specificație.

Hint: începeți cu șiruri de 1 caracter: ex: a

② Fie gramatica de mai jos: **E -> a+E | E+c | b**

(a) Descrieți șirurile recunoscute corect de strategia *recursive descent*.

(b) Se schimbă răspunsul la întrebarea de mai sus dacă reordonăm alternativele? => alternativă recursivă factorizare + elim. rec. la stânga

a) Prima dată ne uităm după cazul de bază și dacă nu cumva 3 rec. la stânga înainte.

=> E -> a+E | E+c | b -> cB

rec. la stânga => cB nu se atinge niciodată

=> niciun șir

b) Reordonare: E -> a+E | b | E+c

SUBIECT ALTERNATIV 2

Rescrieți gramatica de mai jos într-o variantă factorizată la stânga și fără recursivitate la stânga.

S -> Aa | b

A -> Bb | c

B -> S | c

=> Substituim B în A => $S \to Aa \mid b$, $A \to Sb \mid cb \mid c$ $\xrightarrow{A}$ $S \to Sba \mid cba \mid ca \mid b$

(α = ba; β_1 = cba; β_2 = ca; β_3 = b)

$S \to cbaS' \mid caS' \mid bS'$

$S' \to baS' \mid \varepsilon$

=> factorizare la stânga $S \to cTS' \mid bS'$, $T \to ba \mid a$, $S' \to baS' \mid \varepsilon$

Rescrieți gramatica de mai jos într-o variantă factorizată la stânga și fără recursivitate la stânga.

S -> SbS | aS | a

$S \to SbS \mid aS \mid a$ (α = bS; β_1 = aS; β_2 = a)

$S \to aSS' \mid aS'$, $S' \to bSS' \mid \varepsilon$ $\xrightarrow{fact}$ $S \to aX$, $X \to SS' \mid S'$, $S' \to bSS' \mid \varepsilon$

Fie gramatica:

S -> xSyA | ε
A -> Bz | zS
B -> Au | ε

3 Calculați mulțimile First și Follow, construiți tabelul de analiză LL(1) și conchideți dacă gramatica este sau nu LL(1).

S -> ~~xSyA~~ | ~~ε~~
A -> ~~Bz~~ | ~~zS~~
B -> ~~Au~~ | ~~ε~~

First(S) = {x, ε}
First(A) = {z}
First(B) = {ε}

	FIRST
S	x, ε
A	z
B	ε, z

First(A) = First(Bz)
= First(B) \ {ε} ∪ First(z) = {z}

First(B) = First(Au) = First(A) = {z}

S -> xSyA | ε
A -> Bz | zS
B -> Au | ε

$ ∈ Follow(S)
Follow(S) = {y}
Follow(A) = Follow(S)
Follow(B) = {z}
Follow(S) = Follow(A)
Follow(A) = {u}

	FOLLOW
S	$ y u
A	$ y u
B	z

	x	y	z	u	$
S	xSyA	ε		ε	ε
A			Bz zS		
B			Au ε		

=> non-LL(1)

4 a) Construiți automatul de parsare LR doar până întâlniți o stare cu un conflict LR(0) și precizați tipul acestuia și în ce constă.

(b) Precizați dacă și de ce dispare conflictul depistat mai sus, asumând euristica SLR(1).

S	**->**	**xSyA** |	**ε**
A	**->**	**Bz** |	**zS**
B	**->**	**Au** |	**ε**

a) $S' \to \cdot S$

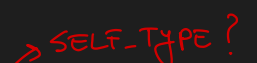
$S \to \cdot xSyA$) conflict SR

$S \to \cdot$

b) SLR(1) $x \notin Follow(S) \Rightarrow$ conflictul dispare

```
class MinQueue {
    isEmpty() : Bool { true };
    getMin() : Object { { abort(); 0; } };
    removeMin() : MinQueue { { abort(); self; } };
    insert(elem : Object, prio : Int) : NEMinQueue {
        new NEMinQueue.init(elem, prio, self)
    };
};
class NEMinQueue inherits MinQueue {
    elem : Object;
    prio : Int;
    next : MinQueue;
    init(e : Object, p : Int, n : MinQueue) : SELF_TYPE {{
        elem <- e;
        prio <- p;
        next <- n; self;
    }};

    isEmpty() : Bool { false };
    getMin() : Object { elem };
    removeMin() : MinQueue { next };
    insert(e : Object, p : Int) : NEMinQueue {
        if p <= prio
            then self@MinQueue.insert(e, p)
            else new NEMinQueue.init(elem, prio, next.insert(e,p))
        fi
    (* P1 *)
    };
};
class Main inherits IO {
    main() : Object {
        let queue : MinQueue <- new MinQueue.insert(5, 20)
                                    .insert(true, 10)
                                    .insert("abc", 30),
        temp : MinQueue <- queue.insert(queue, 0)
        in {
            (* P2 *)
            while not temp.isEmpty() loop {
                out_string(temp.getMin().type_name());
                temp <- temp.removeMin();
            } pool;
            (* P3 *)
        }
    };
};
```



5 Care este conținutul contextelor de tipare pentru obiecte, O, în punctul P1, respectiv pentru metode, M?

Sintaxă:

O ⇒ O(v) = T ⇒ obiectul v are tipul T

M ⇒ M(C, f) = (T1, T2 ... Tm, Tm+1) ⇒ în clasa C, metoda f are param. formali cu tipurile T1 ... Tm și tipul de retur Tm+1

(T1, T2 ... Tm — tip. param; Tm+1 — tip retur)

O(self) = SELF_TYPE_NEMinQueue
O(e) = Object
O(p) = Int
O(elem) = Object
O(prio) = Int
O(next) = MinQueue

M(MinQueue, isEmpty) = (Bool)
M(MinQueue, getMin) = (Object)
M(MinQueue, removeMin) = (MinQueue)
M(MinQueue, insert) = (Object, Int, NEMinQueue)

metode moștenite:
M(NEMinQueue, isEmpty) = (Bool)
M(NEMinQueue, getMin) = (Object)
M(NEMinQueue, removeMin) = (MinQueue)
M(NEMinQueue, insert) = (Object, Int, NEMinQueue)

M(NEMinQueue, init) = (Object, Int, NEMinQueue, SELF_TYPE)

SELF_TYPE — nu ST ceva

mențiune: dacă aveam o funcție în MinQueue care nu apărea în NEMinQueue o aveam!!

+ metode moștenite din Object

(6) (a) Aplicați regula potrivită pentru verificarea tipului expresiei self@MinQueue.insert(e, p), din metoda insert a clasei NEMinQueue. Utilizați manualul Cool.
(b) Este posibilă modificarea tipului întors de metoda insert de la NEMinQueue la SELF_TYPE?

a)

$$O, M, C \vdash e_0 : T_0 \quad \Rightarrow T_0 = \text{SELF\_TYPE}_{\text{NEMinQueue}}$$
$$O, M, C \vdash e_1 : T_1 \quad \Rightarrow T_1 = \text{Object} \quad O(e) = \text{Object}$$
$$\vdots$$
$$O, M, C \vdash e_n : T_n \quad \Rightarrow T_2 = \text{Int} \quad O(p) = \text{Int}$$
$$T_0 \le T \quad \text{SELF\_TYPE}_{\text{NEMinQueue}} \le \text{MinQueue}$$
$$M(T, f) = (T_1', \ldots, T_n', T_{n+1}') \Rightarrow M(\text{MinQueue}, \text{insert}) = (\text{Object}, \text{Int}, \text{NEMinQueue})$$
$$(*)\ T_i \le T_i' \quad 1 \le i \le n \ \checkmark$$
$$T_{n+1} = \begin{cases} T_0 & \text{if } T_{n+1}' = \text{SELF\_TYPE} \\ T_{n+1}' & \text{otherwise} \end{cases} \Rightarrow T_{n+1} = \text{NEMinQueue}$$
$$\frac{}{O, M, C \vdash e_0@T.f(e_1, \ldots, e_n) : T_{n+1}} \Rightarrow \text{NEMinQueue}$$

e_0: NEMinQueue, self; @T: MinQueue; f: insert; e_1: e; e_n: p

(*) Object ≤ Object ✓
Int ≤ Int ✓

b)

$$M(C, f) = (T_1, \ldots, T_n, T_0) \quad \rightarrow \text{SELF\_TYPE}$$
$$O_C[\text{SELF\_TYPE}_C/self][T_1/x_1] \ldots [T_n/x_n], M, C \vdash e : T_0' \quad T_0' : \text{NEMinQueue}$$
$$T_0' \le \begin{cases} \text{SELF\_TYPE}_C & \text{if } T_0 = \text{SELF\_TYPE} \\ T_0 & \text{otherwise} \end{cases} \quad T_0' \le \text{SELF\_TYPE}_{\text{NEMinQueue}} \ \times$$
$$\frac{}{O_C, M, C \vdash f(x_1 : T_1, \ldots, x_n : T_n) : T_0 \ \{ e \}}$$

nu pot avea așa ceva

NEMinQueue; insert; e : Obj; p : Int; SELF_TYPE → if ...

$T \le \text{SELF\_TYPE}_C$ (nu pot ști)

C → T → A

7. (a) Descrieți reprezentarea în memorie a obiectelor prototip pentru clasele MinQueue și NEMinQueue, incluzând tabelele de metode.
(b) Presupunând că toate informațiile necesare evaluării corpurilor metodelor se depun pe stivă, care este dimensiunea minimă a înregistrării de activare pentru metoda insert din clasa NEMinQueue?

a)

MinQueue → flag 0
dim 3
MinQueue-dispTab → Object ... x3
MinQueue.isEmpty
MinQueue.getMin
MinQueue.removeMin
MinQueue.insert

NEMinQueue → flag 1
dim 6
NEMinQueue-dispTab → Object ... x3
elem NEMinQueue.isEmpty
prio NEMinQueue.getMin
next NEMinQueue.removeMin
NEMinQueue.insert
NEMinQueue.init

b)

Conținut	Adresă
Parametru n	
⋮	⋮
Parametru 2	$fp + 16
Parametru 1	$fp + 12
$fp	
$s0	
$ra	$fp
	$sp

locații temporare

① registri: $fp, $s0, $ra ⇒ 3
② param: 1
③ locații temporare: if p <= prio ⇒ max(NT(p), 1 + NT(prio))
... ⇒ 0
... ⇒ 0
fi = max(0, 1+0) = 1

⇒ dimensiune = 3+2+1 = 6 cuvinte

(8) Pentru expresia out_string(temp.getMin().type_name()), din metoda main a clasei Main, este propusă următoarea secvență de cod MIPS, unde registrul $s0 conține adresa lui self, iar convențiile de organizare a înregistrării de activare sunt cele din enunțul temei 3. Completați cele 4 spații libere cu valorile corecte. **Justificați!**

```
lw      $a0 -8 ($fp)                    } temp
<verificare dispatch on void>
lw      $t1 8($a0)  -> dispTab MinQueue }
lw      $t1 16($t1)                     } getMin() => Object
jalr    $t1                             }
<verificare dispatch on void>
lw      $t1 8($a0)  -> dispTab Obj      } type_name()
lw      $t1 4($t1)                      }
jalr    $t1                             }
sw      $a0 0($sp)                      } push param
addiu   $sp $sp -4                      }
move    $a0 $s0     -> self (implicit)  }
<verificare dispatch on void>
lw      $t1 8($a0)  -> dispTab Main     } out_string(...)
lw      $t1 12($t1)                     }
jalr    $t1                             }
```

Unde stă temp? => $fp - 8

Conținut	Adresă
Parametru n	
⋮	⋮
Parametru 2	$fp + 16
Parametru 1	$fp + 12
$fp	
$s0	
$ra	$fp
Variabilă let 1	$fp - 4
Variabilă let 2	$fp - 8
⋮	⋮
Variabilă let m	
	$sp

MinQueue -> 0 tag 0
4 dim 3
8 MinQueue_dispTab -> Object. ... x3 0, 4, 8
12 MinQueue.ioEmpty
16 MinQueue.getMin
20 MinQueue.removeMin
24 MinQueue.insert

```
Object_dispTab:
    .word 0 Object.abort
    .word 4 Object.type_name
    .word 8 Object.copy
```

Main_dispTab ->

```
IO_dispTab:
    .word 0  Object.abort
    .word 4  Object.type_name
    .word 8  Object.copy
    .word 12 IO.out_string
    .word 16 IO.out_int
    .word 20 IO.in_string
    .word 24 IO.in_int
```

main

OBS: Metodele Object, IO, ... vor avea mereu același offset, indiferent de clasă!

10 Descrieți, pas cu pas, aplicarea strategiei Local Value Numbering asupra basic block-ului de mai jos. Completați tabelul valorilor și indicați cum se modifică fiecare instrucțiune, unde este cazul.

```
a := 1 + 4
b := 4
c := 1
d := b + c
e := d - 4
f := e * x
g := x - f
```

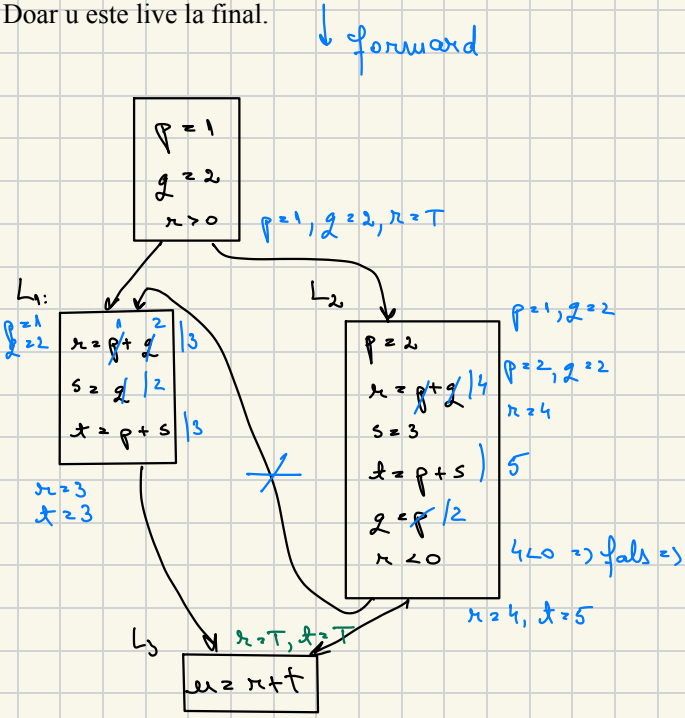
a := 5
b := 4
c := 1
d := 5
e := 1
f := x
g := 0

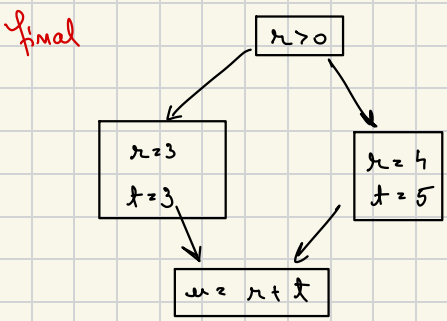
⇒ a = 5
b = 4
c = 1
d = 5
e = 1
f = x
g = 0

#	valoare	repr. canonică
0	1	1
1	4	4
2	#0 + #1	5
2	a	5
2	5	5
1	b	4
0	c	1
2	d	5
0	#2 − #1	1
0	e	1
3	x	x
3	#0 * #3	x
3	f	x (1*x)
4	#3 − #3	0
4	g	0

```
    p := 1
    q := 2
    if r > 0 goto L2
L1:
    r := p + q
    s := q
    t := p + s
    jump L3
L2:
    p := 2
    r := p + q
    s := 3
    t := p + s
    q := p
    if r < 0 goto L1
L3:
    u := r + t
```

Desenați graful fluxului de control și aplicați repetat optimizări locale și propagarea globală a constantelor, până când nu mai este posibil. Doar u este live la final.

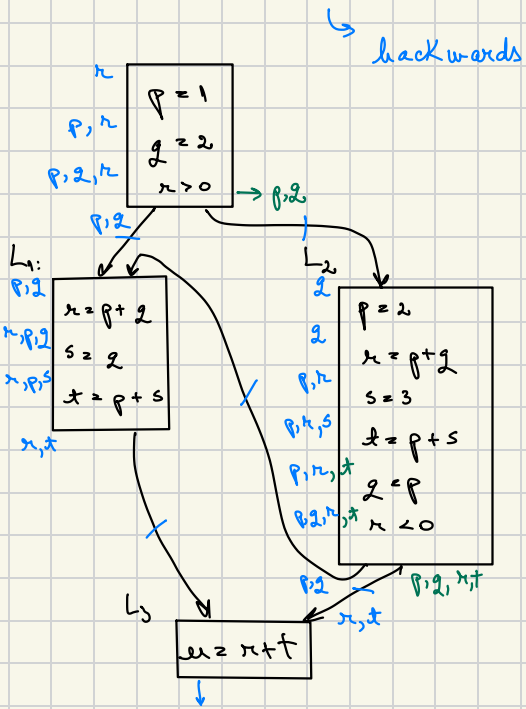


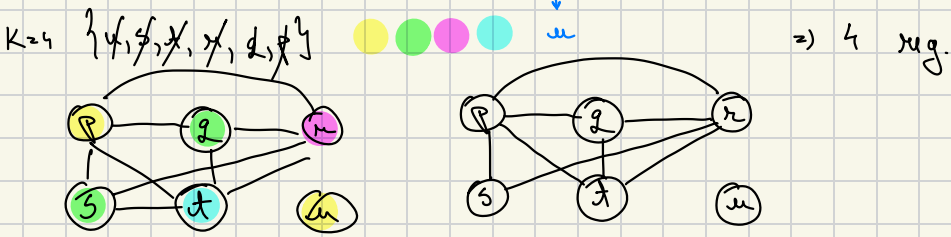


12

```
    p := 1
    q := 2
    if r > 0 goto L2
L1:
    r := p + q
    s := q
    t := p + s
    jump L3
L2:
    p := 2
    r := p + q
    s := 3
    t := p + s
    q := p
    if r < 0 goto L1
L3:
    u := r + t
```

Independent de alte optimizări, menționați variabilele live în fiecare punct al secvenței. Doar u este live la final. Propuneți numărul minim de regiștri necesari în vederea alocării fără spilling.



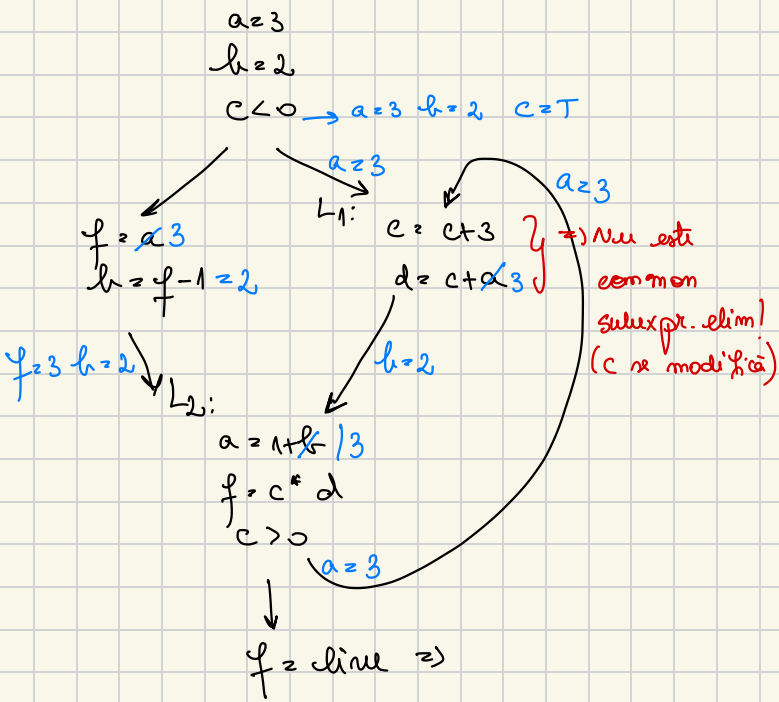


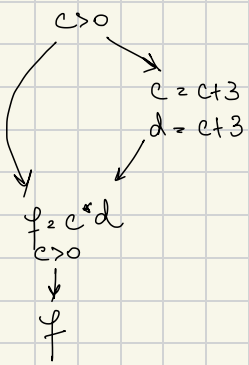
⇒ 4 reg.

(11)

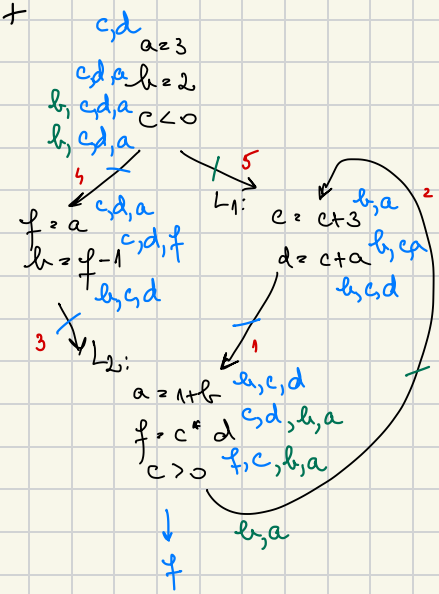
```
a := 3
b := 2
if c < 0 goto L1
f := a
b := f - 1
jump L2
L1:
c := c + 3
d := c + a
L2:
a := 1 + b
f := c * d
if c > 0 goto L1
```

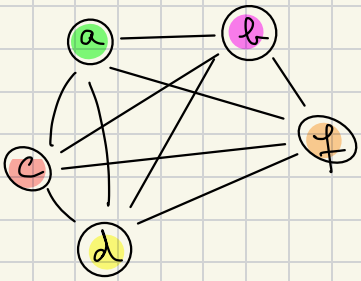
optimizări locale + global d. propagation





(12) liveness analysis + alocare reg.





=> graf complet => K=5

SUBIECT ALTERNATIV : SEMANTICA OPERAȚIONALĂ

```
class A {
    key : Int;
    next : A <- self;
    setKey(k : Int) : A {{ key <- k; self; }};
    setNext(n : A) : A {{ next <- n; self; }};
    getNext() : A { next };
    add(k : Int) : A {{
        let crt : A <- self in {
            while not (crt.getNext() = self) loop
                crt <- crt.getNext()
            pool;
            let fresh : A <- new A.setKey(k) in {
                crt.setNext(fresh);
                fresh.setNext(self);
            };
        };
        self;
    }};
};

class B inherits A {
    add(k : Int) : A {{
        self@A.add(k);
        self@A.add(k + 1);
    }};
};

class Main {
    main() : Object {
        let a : A <- new A.setKey(1) in {
            a.add(2).add(3);
        }
    };
};
```

aici a' încă nu este nimic

Aplicați regula de evaluare din semantica operațională pentru expresia "new A" din metoda main, în raport cu un context determinat de so, E, S, explicitate în prealabil de voi. Utilizați manualul Cool.

$so = Main()$

$S_1 = []$

$E = []$

$$T_0 = \begin{cases} X & \text{if } T = \text{SELF\_TYPE and } so = X(\ldots) \\ T & \text{otherwise} \end{cases} \Rightarrow T_0 = A$$

$class(T_0) = (a_1 : T_1 \leftarrow e_1, \ldots, a_n : T_n \leftarrow e_n)$ $\Rightarrow class(A) = (key : Int, next : A \leftarrow self)$

$l_i = newloc(S_1)$, for $i = 1 \ldots n$ and each l_i is distinct $\Rightarrow l_1 = newloc(S_1), l_2 = newloc(S_1)$

$v_1 = T_0(a_1 = l_1, \ldots, a_n = l_n)$ $v_1 = A[key = l_1, next = l_2]$

$S_2 = S_1[D_{T_1}/l_1, \ldots, D_{T_n}/l_n]$ $S_2 = S_1[Int(0)/l_1, void/l_2]$

$$\frac{v_1, S_2, [a_1 : l_1, \ldots, a_n : l_n] \vdash \{a_1 \leftarrow e_1; \ldots; a_n \leftarrow e_n;\} \mapsto v_2, S_3 \quad (*)}{so, S_1, E \vdash \text{new } T \mapsto v_1, S_3} \quad [\text{New}]$$

$T = A$

$(*)$ $v_1, S_2, [key : l_1, next : l_2] \vdash \{next \leftarrow self;\} \rightarrow v_2, S_3$

v_1, S_3

$$\frac{so, S_1, E \vdash e_1 \mapsto v_1, S_2 \qquad E(Id) = l_1 \qquad S_3 = S_2[v_1/l_1]}{so, S_1, E \vdash Id \leftarrow e_1 \mapsto v_1, S_3} \quad \text{[Assign]}$$

$$\frac{}{so, S, E \vdash \text{self} \mapsto so, S} \quad \text{[Self]}$$