Programming 2 (SS 2023)
Saarland University
Faculty MI
Compiler Design Lab

Sample Solution 7
C0 Semantics and C0bp

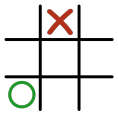
Prof. Dr. Sebastian Hack
Pascal Lauer, M. Sc.
Marcel Ullrich, B. Sc.

The calendar indicates which script chapters you should study in conjuction with each lecture. The exercises are designed to enhance your understanding of the lecture material and prepare you for the mini-tests and the final exam. Additional exercises can be found at the end of each chapter in the script.
The difficulty of an exercise on the sheet is determined by the number of annotated 'X' and 'O' marks in the tic-tac-toe field, with four levels (1-4) increasing by one mark per level.

1 C0 Semantics

Exercise 7.1: For Loop

Extend the syntax and semantics of the C0-language by the following constructs:



1. The do-while loop (see the Control Flow chapter in the book).
2. A limited version of the for-loop. It should have the concrete syntax $\langle \texttt{for}\ (x\ =\ e_1;\ e_2;\ x{+}{+})\ s \rangle$.

Solution

1. Extension of the syntax:

		Abstract Syntax	Concrete Syntax	Description
$Stms \ni s$	$:=$	$\texttt{DoWhile}[s, e]$	$\texttt{do}\ s\ \texttt{while}(e)$	Do-While Loop

   Extension of the semantics:

$$\langle \textbf{do}\ s\ \textbf{while}(e) | \sigma \rangle \rightarrow \langle s \_\ \textbf{while}(e)\ s | \sigma \rangle \qquad \text{[DoWhile]}$$

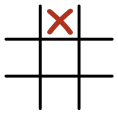
2. Extension of the syntax:

		Abstract Syntax	Concrete Syntax	Description
$Stms \ni s$	$:=$	$\texttt{For}[x, e_1, e_2, s]$	$\texttt{for}\ (x\ =\ e_1;\ e_2;\ x{+}{+})\ s$	Do-While Loop

   Extension of the semantics:

$$\langle \textbf{for}\ (x\ =\ e_1;\ e_2;\ x{+}{+})\ s | \sigma \rangle \rightarrow \langle \{ x\ =\ e_1; \_\ \textbf{while}(e_2) \{\ s \_\ x = x + 1; \}\} | \sigma \rangle \qquad \text{[For]}$$

Exercise 7.2:



Consider the following state in shorthand notation as defined in the lecture script:

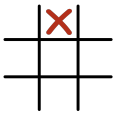
$$\sigma = \{\mathrm{x} \mapsto 1, \mathrm{y} \mapsto 2\}$$

Write down ρ and μ. Use $\triangle, \pentagon \ldots$ do depict addresses.

Solution

$$\sigma = (\rho, \mu) \text{ where}$$
$$\rho = \{\mathtt{x} \mapsto \triangle, \mathtt{y} \mapsto \pentagon\},$$
$$\mu = \{\triangle \mapsto 1, \pentagon \mapsto 2\}$$

Exercise 7.3: C0 execution protocol



Execute the following program according to C0 semantics on the state $\sigma = \{\mathtt{n} \mapsto 3, \mathtt{r} \mapsto ?\}$

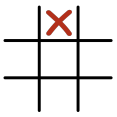
```
r = 1;
while (r <= n) {
    r = r * n;
    n = n - 1;
}
```

Solution

To improve readability, we use the abbreviated notation σ and the shorthand S for loop body.

$$\begin{aligned}
& \langle \mathtt{r} = 1; \mathtt{while}\,(\mathtt{r} <= \mathtt{n})\, S \mid \{\mathtt{n} \mapsto 3, \mathtt{r} \mapsto ?\}\rangle \\
\to\ & \langle \mathtt{while}\,(\mathtt{r} <= \mathtt{n})\, S \mid \{\mathtt{n} \mapsto 3, \mathtt{r} \mapsto 1\}\rangle && \text{[Assign]} \\
\to\ & \langle \{\mathtt{r} = \mathtt{r} * \mathtt{n}; \mathtt{n} = \mathtt{n} - 1;\} \mathtt{while}\,(\mathtt{r} <= \mathtt{n})\, S \mid \{\mathtt{n} \mapsto 3, \mathtt{r} \mapsto 1\}\rangle && \text{[WhileTrue]} \\
\to\ & \langle \mathtt{r} = \mathtt{r} * \mathtt{n}; \mathtt{n} = \mathtt{n} - 1; \mathtt{while}\,(\mathtt{r} <= \mathtt{n})\, S \mid \{\mathtt{n} \mapsto 3, \mathtt{r} \mapsto 1\}\rangle && \text{[Block]} \\
\to\ & \langle \mathtt{n} = \mathtt{n} - 1; \mathtt{while}\,(\mathtt{r} <= \mathtt{n})\, S \mid \{\mathtt{n} \mapsto 3, \mathtt{r} \mapsto 3\}\rangle && \text{[Assign]} \\
\to\ & \langle \mathtt{while}\,(\mathtt{r} <= \mathtt{n})\, S \mid \{\mathtt{n} \mapsto 2, \mathtt{r} \mapsto 3\}\rangle && \text{[Assign]} \\
\to\ & \langle \varepsilon \mid \{\mathtt{n} \mapsto 2, \mathtt{r} \mapsto 3\}\rangle && \text{[WhileFalse]}
\end{aligned}$$

Exercise 7.4:



Which kind of termination can you observe? In case of proper termination also give the state after execution. You may use short-hand notation.

1. $\sigma = \{\}$

```
1       x = 1;
```

2. $\sigma = \{\mathtt{x} \mapsto 1, \mathtt{y} \mapsto 2\}$

```
1       x = 5;
2       {
3         y = 6;
4       }
```

3. $\sigma = \{\mathtt{b} \mapsto 5, \mathtt{z} \mapsto 1, \mathtt{c} \mapsto 3\}$

```
1       z = b + 3;
2       if (b == 5) {
3         abort();
4       }
```

4. $\sigma = \{\mathtt{c} \mapsto 5, \mathtt{x} \mapsto 1, \mathtt{z} \mapsto 3\}$

```
1       x = 8;
2       z = 2;
3       while (c < 7) {
4         z = z + 1;
5         c = c + 1;
6         x = x + z;
7       }
8       if (x == 16) {
9         abort();
10      }
```

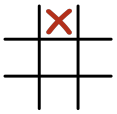
Solution

1. Since the container of the variable x does not exist, the program gets stuck (`Assign` cannot be applied).
2. The program terminates properly. State: $\sigma = \{\mathtt{x} \mapsto 5, \mathtt{y} \mapsto 6\}$
3. The program aborts, since b equals 5 in the given state. The `abort();` statement is thus executed.
4. The program terminates properly. State: $\sigma = \{\mathtt{c} \mapsto 7, \mathtt{x} \mapsto 15, \mathtt{z} \mapsto 4\}$

2 C0bp

Exercise 7.5:

Decide whether the following statements are true or false and justify your answer.



1. The statement `y = &&x;` is syntactically correct in C0p.
2. `x = **y;` is syntactically correct in C0p.
3. The program `x = x / y;` will get stuck for the initial state $\sigma = \{\mathtt{y} \mapsto \triangle\}; \{\triangle \mapsto 42\}$.
4. $R[\![\mathtt{**x}]\!]\sigma$ is defined for $\sigma = \{\mathtt{x} \mapsto \triangle\}; \{\triangle \mapsto \pentagon, \pentagon \mapsto 4\}$
5. The following is a correct application of the [Scope] rule:

$$\langle \{\mathtt{int\ x;\ x = 42;}\} \mid \{\}; \{\}\rangle \to \langle \mathtt{x = 42;} \mid \{\}, \{\mathtt{x} \mapsto \triangle\}; \{\triangle \mapsto ?\}\rangle \quad \text{[Scope]}$$

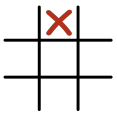
6. $R[\![\mathtt{x}]\!]\{\mathtt{x} \mapsto \triangle\}, \{\mathtt{y} \mapsto \pentagon\}, \{\mathtt{x} \mapsto \diamond\}; \{\triangle \mapsto 1, \pentagon \mapsto 2, \diamond \mapsto 3\} = 1$

Solution

1. No, since Addr[x] $\notin LExpr$. Semantically, the program would also not be correct, since $\&x$ would not be an lvalue.
2. Yes.
3. Yes, since x is not initialized.
4. No, since $L[\![\mathtt{*x}]\!]\sigma$ does not yield an address.

5. No, ■ is missing.
6. No, x = 3, since the last variable assignment has to be used.

Exercise 7.6: C0 AST

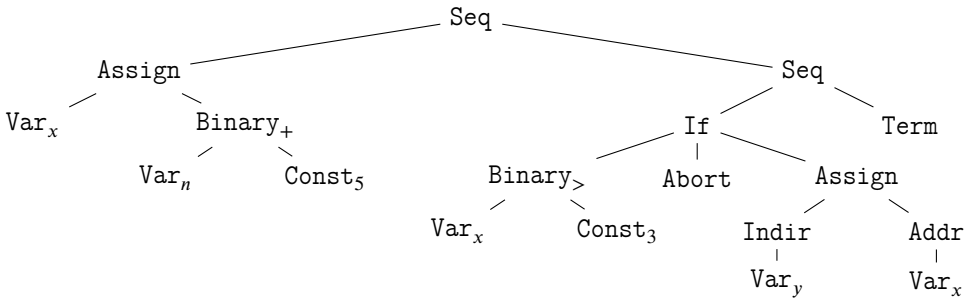


Draw the AST for the following C0 program.

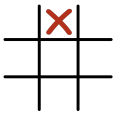
```
x = n + 5;
if (x > 3)
    abort();
else
    *y = &x;
```

Solution

At the highest level, the program has a sequence of an assignment and the if-statement. We need a new Seq node for every statement in the sequence. Hence, the root of the AST is Seq, the left child the assignment and the right one another Seq with the if-statement and empty program as the children. For unary and binary operators, the operator itself is the node and the arguments are the children. However, notice that the assignment and the $*$ and & are written a bit differently than e.g. the plus operator. The if-statement has three children (the condition and the then- and else-case).



Exercise 7.7:



Execute the following C0p program on the state $\sigma = (\rho; \mu) = \{x \mapsto \triangle, y \mapsto \pentagon\}; \{\triangle \mapsto ?, \pentagon \mapsto ?\}$

```
1 {
2     x = 3;
3     y = &x;
4     *y = *y + 1;
5 }
```

Explicitly state the evaluation of $R[\![\texttt{\&x}]\!]$ and $R[\![\texttt{*y + 1}]\!]$.

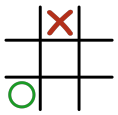
Solution

$$
\begin{array}{lll}
 & \langle \texttt{\{x = 3; y = \&x; *y = *y + 1;\}} \mid \rho; \{\triangle \mapsto ?, \pentagon \mapsto ?\}\rangle & \\
\rightarrow & \langle \texttt{x = 3; y = \&x; *y = *y + 1;} \mid \rho; \{\triangle \mapsto ?, \pentagon \mapsto ?\}\rangle & \text{[Block]} \\
\rightarrow & \langle \texttt{y = \&x; *y = *y + 1;} \mid \rho; \{\triangle \mapsto 3, \pentagon \mapsto ?\}\rangle & \text{[Assign]} \\
\rightarrow & \langle \texttt{*y = *y + 1;} \mid \rho; \{\triangle \mapsto 3, \pentagon \mapsto \triangle\}\rangle & \text{[Assign]} \\
\rightarrow & \langle \varepsilon \mid \rho; \{\triangle \mapsto 4, \pentagon \mapsto \triangle\}\rangle & \text{[Assign]}
\end{array}
$$

$$R[\![\texttt{\&x}]\!]\sigma = L[\![\texttt{x}]\!]\sigma = \rho(\texttt{x}) = \triangle$$

$$\begin{aligned} R[\![\texttt{*y + 1}]\!]\sigma &= R[\![\texttt{*y}]\!]\sigma + R[\![1]\!]\sigma \\ &= \mu(L[\![\texttt{*y}]\!]\sigma) + 1 \\ &= \mu(R[\![\texttt{y}]\!]\sigma) + 1 \\ &= \mu(\mu(L[\![\texttt{y}]\!]\sigma)) + 1 \\ &= \mu(\mu(\rho(\texttt{y}))) + 1 \\ &= \mu(\mu(\pentagon)) + 1 \\ &= \mu(\triangle) + 1 \\ &= 3 + 1 \\ &= 4 \end{aligned}$$

Exercise 7.8:



Execute the following C0p program on the state $\sigma = (\rho; \mu)$ where

$$\rho := \{\texttt{x} \mapsto \triangle, \texttt{y} \mapsto \pentagon, \texttt{z} \mapsto \diamond, \texttt{w} \mapsto \triangledown\}$$
$$\mu := \{\triangle \mapsto ?, \pentagon \mapsto ?, \diamond \mapsto ?, \triangledown \mapsto ?\}$$

```
1   x = 42;
2   y = &x;
3   z = &y;
4   w = **z;
```

Observe especially how the containers are “linked”.

Solution

$\langle$ `x = 42; y = & x; z = & y; w = **z;` $\mid \rho; \{\triangle \mapsto ?, \pentagon \mapsto ?, \diamond \mapsto ?, \triangledown \mapsto ?\}\rangle$

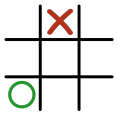
$\rightarrow$ $\langle$ `y = &x; z = &y; w = **z;` $\mid \rho; \{\triangle \mapsto 42, \pentagon \mapsto ?, \diamond \mapsto ?, \triangledown \mapsto ?\}\rangle$ [Assign]

$\rightarrow$ $\langle$ `z = &y; w = **z;` $\mid \rho; \{\triangle \mapsto 42, \pentagon \mapsto \triangle, \diamond \mapsto ?, \triangledown \mapsto ?\}\rangle$ [Assign]

$\rightarrow$ $\langle$ `w = **z;` $\mid \rho; \{\triangle \mapsto 42, \pentagon \mapsto \triangle, \diamond \mapsto \pentagon, \triangledown \mapsto ?\}\rangle$ [Assign]

$\rightarrow$ $\langle \epsilon \mid \rho; \{\triangle \mapsto 42, \pentagon \mapsto \triangle, \diamond \mapsto \pentagon, \triangledown \mapsto 42\}\rangle$ [Assign]

Exercise 7.9:



Execute the following C0b program on the state $\sigma = (\{\}; \{\})$

```
1 {
2   int x;
3   int y;
4   y = 5;
5   x = y + 3;
6   {
7     int x;
8     x = 2;
9   }
10  y = 2 + x;
11 }
```

Solution

$\langle$`{int x; int y; y = 5; x = y + 3; {int x; x = 2;} y = 2 + x;}` $\mid \{\}; \{\}\rangle$

$\rightarrow$ $\langle$`y = 5; x = y + 3; {int x; x = 2;} y = 2 + x;` $\blacksquare \mid \{\}, \{\mathtt{x} \mapsto \triangle, \mathtt{y} \mapsto \pentagon\}; \{\triangle \mapsto ?, \pentagon \mapsto ?\}\rangle$ [Scope]

$\rightarrow$ $\langle$`x = y + 3; {int x; x = 2;} y = 2 + x;` $\blacksquare \mid \{\}, \{\mathtt{x} \mapsto \triangle, \mathtt{y} \mapsto \pentagon\}; \{\triangle \mapsto ?, \pentagon \mapsto 5\}\rangle$ [Assign]

$\rightarrow$ $\langle$`{int x; x = 2;} y = 2 + x;` $\blacksquare \mid \{\}, \{\mathtt{x} \mapsto \triangle, \mathtt{y} \mapsto \pentagon\}; \{\triangle \mapsto 8, \pentagon \mapsto 5\}\rangle$ [Assign]

$\rightarrow$ $\langle$`x = 2;` $\blacksquare$ `y = 2 + x;` $\blacksquare \mid \{\}, \{\mathtt{x} \mapsto \triangle, \mathtt{y} \mapsto \pentagon\}, \{\mathtt{x} \mapsto \diamondsuit\}; \{\triangle \mapsto 8, \pentagon \mapsto 5, \diamondsuit \mapsto ?\}\rangle$ [Scope]

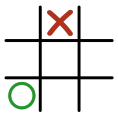
$\rightarrow$ $\langle\blacksquare$ `y = 2 + x;` $\blacksquare \mid \{\}, \{\mathtt{x} \mapsto \triangle, \mathtt{y} \mapsto \pentagon\}, \{\mathtt{x} \mapsto \diamondsuit\}; \{\triangle \mapsto 8, \pentagon \mapsto 5, \diamondsuit \mapsto 2\}\rangle$ [Assign]

$\rightarrow$ $\langle$`y = 2 + x;` $\blacksquare \mid \{\}, \{\mathtt{x} \mapsto \triangle, \mathtt{y} \mapsto \pentagon\}; \{\triangle \mapsto 8, \pentagon \mapsto 5\}\rangle$ [Leave]

$\rightarrow$ $\langle\blacksquare \mid \{\}, \{\mathtt{x} \mapsto \triangle, \mathtt{y} \mapsto \pentagon\}; \{\triangle \mapsto 8, \pentagon \mapsto 10\}\rangle$ [Assign]

$\rightarrow$ $\langle\varepsilon \mid \{\}; \{\}\rangle$ [Leave]

Exercise 7.10:



A pointer is *dangling* if it refers to an address for which there is no container. Construct a C0pb program which contains a dangling pointer during execution. Verify this by executing the program according to the C0pb semantics.

Solution

```
1 {
2     int x;
3     int *p;
4     {
5         int y;
6         p = &y;
7     }
8     x = *p;
9 }
```

According to the constructed program above, the pointer `p` points to a container which does not exist anymore after execution of the inner block. The semantics gets stuck at the assignment statement `x = *p;`. Formally:

$\langle$`{ int x; int *p; { int y; p = &y; } x = *p; }` $\mid \{\}; \{\}\rangle$

$\rightarrow$ $\langle$`{ int y; p = &y; } x = *p;` $\blacksquare \mid \{\}, \{\mathtt{x} \mapsto \triangle, \mathtt{p} \mapsto \pentagon\}; \{\triangle \mapsto ?, \pentagon \mapsto ?\}\rangle$ [Scope]

$\rightarrow$ $\langle$`p = &y;` $\blacksquare$ `x = *p;` $\blacksquare \mid \{\}, \{\mathtt{x} \mapsto \triangle, \mathtt{p} \mapsto \pentagon\}, \{\mathtt{y} \mapsto \diamondsuit\}; \{\triangle \mapsto ?, \pentagon \mapsto ?, \diamondsuit \mapsto ?\}\rangle$ [Scope]

$\rightarrow$ $\langle\blacksquare$ `x = *p;` $\blacksquare \mid \{\}, \{\mathtt{x} \mapsto \triangle, \mathtt{p} \mapsto \pentagon\}, \{\mathtt{y} \mapsto \diamondsuit\}; \{\triangle \mapsto ?, \pentagon \mapsto \diamondsuit, \diamondsuit \mapsto ?\}\rangle$ [Assign]

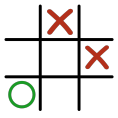
$\rightarrow$ $\langle$`x = *p;` $\blacksquare \mid \{\}, \{\mathtt{x} \mapsto \triangle, \mathtt{p} \mapsto \pentagon\}; \{\triangle \mapsto ?, \pentagon \mapsto \diamondsuit\}\rangle$ [Leave]

Let $\sigma := (\rho_0, \rho_1; \mu)$ be $(\{\}, \{\mathtt{x} \mapsto \triangle, \mathtt{p} \mapsto \pentagon\}; \{\triangle \mapsto ?, \pentagon \mapsto \diamondsuit\})$.

$$\begin{aligned} R[\![\mathtt{*p}]\!]\sigma &= \mu(L[\![\mathtt{*p}]\!]\sigma) \\ &= \mu(R[\![\mathtt{p}]\!]\sigma) \\ &= \mu(\mu(L[\![\mathtt{p}]\!]\sigma)) \\ &= \mu(\mu(\rho_1(\mathtt{p}))) \\ &= \mu(\mu(\pentagon)) \\ &= \mu(\diamondsuit) \quad \text{undefined!} \end{aligned}$$

The container $\diamondsuit$ does not exist anymore, so $R[\![\mathtt{*p}]\!]\sigma \notin \mathit{Val} \cup \{?\}$. Thus, the rule [Assign] is not applicable.

Exercise 7.11: C0 Functions



Extend the formal semantics of C0b with untyped functions and calls. To this end, proceed as follows:

1. Define a new category Func for function definitions and add function calls to the expression category in the abstract syntax. Also extend the existing statements with the return statement. Our new program no longer consists of a simple sequence of statements, but is a sequence of statements and function definitions. Note that it should not be possible to declare functions within functions.

2. Define a new component $\Phi : Var \rightharpoonup$ Func in a state. This component maps identifiers to the code of a function.

3. Extend the operational semantics with sensible rules for any statements you added in the first step. Make function calls R-evaluable. When evaluating a call, you can look up the code of the function in Φ and generate a corresponding configuration.

Solution

1. We introduce a new category (MPrg) so that we can have a separation between function declarations and statements. Since it would be nice to allow function calls as simple statements like in C, we also add expression statements.

Category		Abstract Syntax	Concrete Syntax	Description
$MPrg \ni mp$	::=	$\texttt{Meta}[d, mp]$	$d\ mp$	Meta program
	|	$\texttt{Term}$	ε	Empty program
$Dist \ni d$	::=	$f \mid p$		Distinction
$Func \ni f$	::=	$\texttt{FunDef}[l, x_1, \ldots, x_n, b]$	$l(x_1, \ldots, x_n)\ b$	Function definition
$Stmt \ni s$	::=	$\ldots$		
	|	b		Block
	|	$\texttt{Expression}[e]$	$e;$	Expression statement
	|	$\texttt{Return}[e]$	*return* e;	Return statement
	|	$\texttt{EmptyReturn}$	*return*;	Empty return statement
$Block \ni b$	::=	$\texttt{Block}_{x_1,\ldots,x_n}[p]$	$\{x_1; \ldots; x_n; p\}$	Block with variables

   Finally, we add function calls to the list of expressions.

Category		Abstract Syntax	Concrete Syntax	Description
$LExpr \ni l$	::=	$\ldots$		
$Expr \ni e$	::=	$\ldots$		
	|	$\texttt{FunCall}[l, e_1, \ldots, e_n]$	$l(e_1, \ldots, e_n)$	Function call

2. To support variable bindings to functions, we add a new function Φ to our state. Here, a function contains all arguments and the body of the function. This way we can look up functions for function calls.

$$\Sigma ::= (\text{Var} \rightharpoonup \text{Addr}) \times (\text{Addr} \rightharpoonup \text{Val} \cup \{?\}) \times \Phi$$
$$\Phi := (\text{Var} \rightharpoonup \text{Func})$$

3. We start off by adding rules for the statements we added. If a statement only consists of an expression, we evaluate it and move on. A return statement results in the proper termination of the current program. In case we encounter a function definition, we extend our mapping Φ with it (we omit a check of uniqueness here).

$$\langle e;\ mp \mid \sigma \rangle \to \langle mp \mid \sigma \rangle \quad \texttt{if } R[\![e]\!]\sigma \in \texttt{Val} \qquad \texttt{[Expression]}$$
$$\langle \texttt{return } e;\ mp \mid \rho; \mu; \Phi \rangle \to \langle \varepsilon \mid \rho; \mu[a \mapsto v]; \Phi \rangle \quad \texttt{if } R[\![e]\!]\sigma = v \in \texttt{Val} \texttt{ and } L[\![\omega]\!] = a \in \texttt{Addr} \qquad \texttt{[Return]}$$
$$\langle \texttt{return};\ mp \mid \sigma \rangle \to \langle \varepsilon \mid \sigma \rangle \qquad \texttt{[EmptyReturn]}$$
$$\langle l(x_1, \ldots, x_n)\ b\ mp \mid \rho; \mu; \Phi \rangle \to \langle mp \mid \rho; \mu; \Phi[l \mapsto (x_1, \ldots, x_n, b)] \rangle \qquad \texttt{[FunDef]}$$

   Evaluating a function takes more work, but is somewhat similar to a block. We need to look up the function definition, evaluate the arguments, set up a new scope with them, and then execute the function body according to our semantics. If it terminates, we evaluate the entire call to its return value. Here we take advantage of the fact that missing return statements in returning functions are undefined behaviour in C, and a type

system would catch attempts to use the return value of void-returning functions. ω serves as a temporary container for the return value, whose identifier is not usable in the program.

$$R[\![l(e_1, \ldots, e_n)]\!]\rho_1, \ldots, \rho_k; \mu; \Phi = R[\![\omega]\!]\sigma' \quad \begin{array}{l} \texttt{if } \Phi(l) = (x_1, \ldots, x_n, b) \\ \texttt{and } R[\![e_1]\!] = v_1 \in \texttt{Val}, \ldots, R[\![e_n]\!] = v_n \in \texttt{Val} \\ \texttt{and } \langle b \mid \rho_1, \rho_2; \mu'; \Phi \rangle \downarrow \langle \varepsilon \mid \sigma' \rangle \end{array}$$

with

$$\begin{array}{rl} \rho_2 & = \{\omega \mapsto a_0, x_1 \mapsto a_1, \ldots, x_n \mapsto a_n\} \\ \mu' & = \mu[a_0 \mapsto ?, a_1 \mapsto v_1, \ldots, a_n \mapsto v_n] \\ \emptyset & = \{a_0, a_1, \ldots, a_n\} \cap \texttt{dom}\ \mu \end{array}$$

Remark: The other rules must be adjusted slightly such that they are defined on mp instead of p and accept a triple argument as the state.