

School of Computer Science and Software Engineering

CSE5340 Programming Languages Concepts and Semantics

Assignment 2

Due 12 noon Monday 8th November, 2004

The purpose of this assignment is to practice the Modular Structural Operational Semantics as a formal way of specifying the semantics of programming languages modularly. We will use the ML language for most of the exercises.

Question 1

[6 + 6 = 12 marks]

This question deals mainly with expressions. Consider a construct for binary operation application with abstract syntax:

$$\text{Exp} ::= \text{app}(\text{Op}, \text{Exp}, \text{Exp})$$

- (a) Give MSOS rules (thus creating .msdf files) for the evaluation of $\text{app}(0, E1, E2)$ such that:
 - (a.1) the evaluations of $E1$ and $E2$ are sequential, left-to-right;
 - (a.2) the evaluations of $E1$ and $E2$ are interleaved (i.e., (parts of) $E2$ can be evaluated before or after (parts of) $E1$ are evaluated);
 - (a.3) the evaluations of $E1$ and $E2$ are interleaved AND if one of the evaluations computes a zero for the operation, the other evaluation should be abandoned (the total result is a zero).
- (b) Validate your rules by translating the above .msdf files into prolog (using the automatic translator in MSOS) and testing them by running appropriate expressions. Note that you will have to modify the DCG in Lang/ML/SYN.pl to generate the new constructs. Show the derivation (without the labels) for one test (choose it so that it is no longer than one page).

Question 2

[2 + 2 = 4 marks]

This question deals mainly with declarations. Suppose that $D1$, $D2$ and $D3$ are arbitrary value declaration in ML. Argue for or against each of the following semantic equivalences¹:

- (a) $D1 ; (D2 ; D3)$ equivalent to $(D1 ; D2) ; D3$
- (b) $\text{local } D1 \text{ in } (D2 ; D3)$ equivalent to $(\text{local } D1 \text{ in } D2) ; (\text{local } D1 \text{ in } D3)$

Question 3

[5 + 2 + 5 = 12 marks]

This question deals also mainly with declarations. A (simplified) structure declaration in ML can be written:

```
struct I = structure D end
```

where the declaration D may include nested structure declarations. The bindings computed by D can be referenced using expressions of the form $I.I1$ and similarly for bindings computed by nested structures. For example, assume we have a structure called `Tree`, which declares functions `empty`, `lookup` and `insert`. To call these functions one

¹The parenthesis merely indicate the intended grouping, they are not part of the ML concrete syntax

must use `Tree.empty`, `Tree.lookup` and `Tree.insert`, respectively. The declaration `open I` computes the same binding as `D`. For example, after `open add`, one can call `empty`, `lookup` and `insert` without adding `Tree.` in front.

For a more detailed introduction to structures consult Ullman's ML book. However, note that this question only requires you to implement a simplified version of the structure declaration (the one indicated above). No extra features (such as `signatures`) will be required.

- (a) Introduce abstract syntax for the declaration and use of structures, indicating its relationship to the concrete syntax indicated above.
- (b) Introduce values which represent structures.
- (c) Specify transition rules for each of the introduced constructs.

Question 4

[8 + 8 = 16 marks]

This question deals mainly with commands. Consider the following loops:

- The command `repeat C until E`
- The corresponding expression `repeat E1 until E2` where the value computed by the entire expression is the first value computed by `E1`.
- The corresponding expression `repeat E1 until E2` where the value computed by the entire expression is the last value computed by `E1`.

For each of the above loops:

- (a) map it to combinations of constructs that have already been introduced.
- (b) introduce abstract constructs for the loops and their associated transition rules; validate the answers.

Question 5

[4 + 2 = 6 marks]

This question deals mainly with abstractions.

- (a) Consider the following function declaration:

```
let
  val f = fn y => 2*y
in
  f 3
end
```

Run the above abstraction in MSOS (using the `run/2` predicate) displaying all computation steps (but hide the labels). Print the displayed steps and comment (handwritten) the MSOS rules used in each step.

- (b) Consider the following equivalent function declaration:

```
let
  fun f y = 2*y
in
  f 3
end
```

Run the above abstraction in MSOS (using the `run/2` predicate). Print the initial abstract construct and briefly indicate the differences in abstraction for the two cases.

Submission instructions

The above exercises contribute 50% to your total CSE5340 mark.

Submission will be electronic. You have to zip all submitted files in one directory and send them as a mail attachment to `mbanda@mail.csse.monash.edu.au`. As documentation you can either submit plain text files, PDF files or postscript files. Each submission will receive an electronic acknowledgement by e-mail. If you do not receive it, you must assume your submission did not get through.

You must also submit a printed version of all parts (including any program listings) of the assignment to the general office in printed form.

The assignment is due at 12 noon Monday 8th of November, 2004. Assignments handed in after the due date will attract a late penalty of 10% per day unless special consideration applies or there has been prior agreement in writing from the lecturer. No submission will be accepted later than one week after the due date.