

Leiden University
Leiden Institute for Advanced Computer Science

Concepts of Programming Languages
F. Arbab
Final Exam

13 January 2017

Student name:

Student number:

This exam consists of 7 pages and 14 questions, for a total of 110 points. Question 14 is a bonus question, for 10 points. If you score x points in this exam, your final exam grade will be calculated as x out of 100.

Instructions:

- Write your name and student number on this hand-out.
- Verify that your copy of this hand-out is complete and legible.
- This is a closed-book, closed-notes, individual exam.
- You are not allowed to use your laptop, phone, or any other computing or telecommunication device.
- You can work on this exam only within the allocated time-slot.
- Do not unstaple or tear off pages of this hand-out.
- Do not write your answers on this hand-out.
- You must return all pages of this hand-out to the proctor at the end of the exam, regardless of whether or not you have written anything on it.

1. Consider the following grammar, with $\langle S \rangle$ as its distinguished symbol:

$\langle S \rangle \rightarrow \langle S \rangle * \langle S \rangle \mid x \mid y \mid z$

- (a) (3 points) Prove that this grammar is ambiguous.
(b) (4 points) Give an unambiguous grammar whose language is the same as that of the above grammar.
2. (3 points) Precisely describe in simple English the language of the following grammar, with $\langle S \rangle$ as its distinguished symbol:

$\langle S \rangle \rightarrow \langle S1 \rangle \langle S2 \rangle$
 $\langle S1 \rangle \rightarrow ab \mid a \langle S1 \rangle b$
 $\langle S2 \rangle \rightarrow bc \mid b \langle S2 \rangle c$

3. Compute the weakest pre-condition for each of the following program fragments and its given post-condition:

- (a) (1 point) $a = 4 * b - 6 \{a = 10\}$
(b) (2 points) $a = 3 * (b + a); b = a - 2 \{b = 10\}$
(c) (3 points)

```
b = 8;
if (a > b)
then b = a - 2
else b = a + 2
fi;
{b > 5}
```

- (d) (4 points)

```
y = - 4 * c;
if ( y >= 0 )
then z = sqrt(y); a = - z / 2; b = z / 2
fi
{a = 3, b = -3}
```

4. (a) (2 points) Consider the following declarations in Ada:

```
type DerivedSmallInt is new Integer range 1..100;
type SubrangeSmallInt is Integer range 1..100;
```

Explain what the derived type `DerivedSmallInt` and the subtype `SubrangeSmallInt` have in common and what, if anything, makes them different.

- (a) **(3 points)** This programming language exclusively uses static scoping.
 - (b) **(3 points)** This programming language exclusively uses dynamic scoping.
6. **(2 points)** Character strings can be implemented either as an array of characters together with an integer that specifies its length (as in Pascal), or as a sequence of characters with a terminator character at the end (as in C). Describe the main advantages and disadvantages of each scheme.
 7. **(4 points)** Explain two common methods used to reclaim memory allocated to dynamically allocated variables that are no longer needed.
 8. **(10 points)** Write a Scheme program that finds the maximum element in a non-empty list of numbers.
 9. **(10 points)** Write a Prolog program `leq(L1, L2)` that returns true only if `L1` and `L2` are fully instantiated lists with identical (nested list) structures, and every element in `L1` is less than or equal to its corresponding element in `L2`.
 10. Consider the following Scheme program:

```
(define (e a b)
  (cond
    ((and (not (list? a)) (not (list? b))) (> a b))
    ((and (list? a) (list? b))
     (cond
       ((null? a) (null? b))
       ((null? b) #F)
       ((e (car a) (car b)) (e (cdr a) (cdr b)))
       (else #F)
     )
    )
    (else #F)
  )
)
```

- (a) **(2 points)** What type of parameters are acceptable in a call to `e`?
- (b) **(8 points)** Explain succinctly what this program does with a call to `e` with every appropriate combination of types of parameters.

11. (10 points) Consider the following Ada program:

```

task TaskA is
  entry Req1(Item : in Integer);
end TaskA;

task body TaskA is
begin
  loop
    accept Req1(Item : in Integer) do
      -- process Item
    end Req1;
  end loop;
end TaskA;

task TaskB;

task body TaskB is
begin
  loop
    -- produce a NewItem
    TaskA.Req1(NewItem);
    -- do other things
  end loop;
end TaskB;

```

Implement the same concurrent behavior of the two tasks TaskA and TaskB in a C-like language using semaphores. Assume Semaphore `x` declares variable `x` as an initially unlocked binary semaphore, and `lock(x)` and `unlock(x)` lock and unlock `x`, respectively.

12. Consider the following Prolog program:

```

f([H|T], M) :- g(T, H, M).
g([], M, M).
g([H|T], X, M) :- h(Y, H, X), g(T, Y, M).
h(M, A, B) :- A > B, !, M = A.
h(M, _, M).

```

- (1 point) What type of parameters are acceptable in a call to `h`?
- (2 points) Explain what exactly `h(M, A, B)` does with suitable parameter types.
- (1 point) Explain what purpose the `!` operator serves in the first rule for `h(M, A, B)`.
- (1 point) Explain what happens if the `!` operator is removed from the body of this rule.
- (1 point) What type of parameters are acceptable in a call to `f`?
- (3 points) Explain exactly what `f(L, M)` does with appropriate parameter types.
- (1 point) What is the result returned for `f([], M)`?

13. Consider an application that consists of the following four processes, where ... represents some sequential computation, and `bexp1`, `bexp2`, and `bexp3` represent Boolean expressions, whose execution or evaluation raises no error. The `lock()` and `unlock()` calls, respectively, lock and unlock their binary semaphore arguments. The variables `S1`, `S2`, and `S3` are globally declared, shared binary semaphores, and `x1`, `x2`, and `x3` are globally declared, shared variables of type `float`.

Process A:

```
...
lock(S1)
lock(S2)
x1 = x1 + 2
x2 = x2 + 2
unlock(S1)
unlock(S2)
...
```

Process B:

```
...
lock(S2)
lock(S3)
x2 = x2 * 3
x3 = x3 * 3
unlock(S3)
unlock(S2)
...
```

Process C:

```
...
lock(S3)
if (bexp1) then {
    lock(S1)
    x3 = x3 - 4
    x1 = x1 - 4
    unlock(S1)
}
unlock(S3)
...
```

Process D:

```
...
lock(S2)
if (bexp2) then {
    t1 = x2 + 1
    lock(S3)
    x2 = x3 + 4
    if (bexp3) then {
        t2 = t1 + x2
        lock(S1)
        x1 = t2 / 2
        unlock(S1)
        t1 = x1 + 1
    }
    x3 = t1 * 10
    unlock(S3)
    ...
}
unlock(S2)
...
```

- (a) (5 points) Explain every possible way that this application can deadlock.
- (b) (5 points) Modify the code of these processes by changing the order of its semaphore operations to ensure it runs free of deadlock, preserving maximum concurrency.

13. Consider an application that consists of the following four processes, where ... represents some sequential computation, and `bexp1`, `bexp2`, and `bexp3` represent Boolean expressions, whose execution or evaluation raises no error. The `lock()` and `unlock()` calls, respectively, lock and unlock their binary semaphore arguments. The variables `S1`, `S2`, and `S3` are globally declared, shared binary semaphores, and `x1`, `x2`, and `x3` are globally declared, shared variables of type `float`.

Process A:

```
...
lock(S1)
lock(S2)
x1 = x1 + 2
x2 = x2 + 2
unlock(S1)
unlock(S2)
...
```

Process B:

```
...
lock(S2)
lock(S3)
x2 = x2 * 3
x3 = x3 * 3
unlock(S3)
unlock(S2)
...
```

Process C:

```
...
lock(S3)
if (bexp1) then {
    lock(S1)
    x3 = x3 - 4
    x1 = x1 - 4
    unlock(S1)
}
unlock(S3)
...
```

Process D:

```
...
lock(S2)
if (bexp2) then {
    t1 = x2 + 1
    lock(S3)
    x2 = x3 + 4
    if (bexp3) then {
        t2 = t1 + x2
        lock(S1)
        x1 = t2 / 2
        unlock(S1)
        t1 = x1 + 1
    }
    x3 = t1 * 10
    unlock(S3)
    ...
}
unlock(S2)
...
```

- (a) (5 points) Explain every possible way that this application can deadlock.
- (b) (5 points) Modify the code of these processes by changing the order of its semaphore operations to ensure it runs free of deadlock, preserving maximum concurrency.





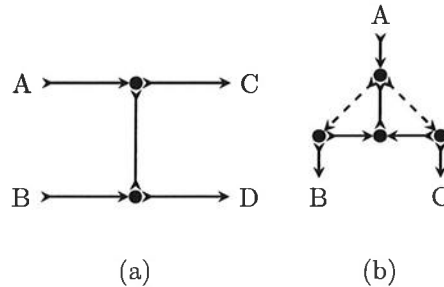
Name	Graphical syntax	Informal semantics
Sync		Atomically takes a data item d from its source end and writes d to its sink end.
LossySync		Atomically takes a data item d from its source end and either (1) writes d to its sink end if its sink end can dispense d , or (2) loses d .
SyncDrain		Atomically takes data items from both its source ends and loses them.
FIFO1		Has an initially empty buffer to hold a single data item. With an empty buffer, takes a data item d from its source end and places it in its buffer, making it full. With a full buffer, writes the contents of its buffer, d , to its sink end.

Table 1: A set of common Reo channels

14. Bonus question

Reo was introduced in class as a domain-specific language (DSL) to program concurrency protocols. The channels in Table 1 were introduced in class as a small set of useful primitives for construction of Reo circuits.

- (a) (4 points) Explain what each of the following circuits does:



- (b) (3 points) Using the primitives in Table 1, construct a circuit that behaves like a FIFO1, except that if the buffer of the FIFO1 is full, the write operation to this circuit succeeds and the value of the write operation is lost.
- (c) (3 points) Using the primitives in Table 1, construct a circuit that produces three copies of every data item that it reads from its input port A through its output port B . For instance, if successive writes to A feed this circuit with the sequence of natural numbers, 1, 2, 3, 4, ..., successive reads from B must obtain 1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 4, ...