Trusted Components

17 December 2007

Name, First name: ..............................................................................................

Stud.-Number: .....................................................................................................

I confirm with my signature, that I was able to take this exam under regular circumstances and that I have read and understood the directions below.

Signature: .............................................................................................................

Directions:

- Except for a dictionary and personal notes, you are not allowed to use any supplementary material.
- Please write your student number onto each sheet.
- Only one solution can be handed in per question. Invalid solutions need to be crossed out clearly.
- Please write legibly! We will only correct solutions that we can read.
- Manage your time carefully (take into account the number of points for each question).
- Please immediately tell the supervisors of the exam if you feel disturbed during the exam.
- The maximum duration of the examination is 1h45mn the minimum duration is 1h.

Good Luck!
Stud.-Number: ..............................................................................................................

<table>
<thead>
<tr>
<th>Question</th>
<th>Number of possible points</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>27</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>75</td>
<td></td>
</tr>
</tbody>
</table>

Grade: ..............................................................................................................
1. Axiomatic semantics (27 points)

1.1 (5 points) Write the partial correctness inference rule of axiomatic semantics for loops `from a until c loop b end'.

Hints:
- This is an inference rule whose purpose is to deduce \{P\} L \{Q\} where L is `from I until e loop B end'.
- The rule involves an assertion that may be called INV.

1.2 (1 point) Consider the following form of recursive function, with one (natural) integer argument and an integer result:

\[ f(n: \text{NATURAL}): \text{NATURAL} \]
\[ \begin{array}{l}
\text{do} \\
\text{if } n = 0 \text{ then} \\
\quad \text{Result} := c \\
\text{else} \\
\quad \text{Result} := g(n, f(n-1)) \\
\text{end} \\
\end{array} \]

where \( c \) is a constant and \( g \) is a two-argument function (whose value is defined entirely in terms of its arguments, i.e. without use of any other entity of the program).

An example is

\[ \text{factorial (n: \text{NATURAL}): \text{NATURAL}} \]
\[ \begin{array}{l}
\text{do} \\
\text{if } n = 0 \text{ then} \\
\quad \text{Result} := 1 \\
\text{else} \\
\quad \text{Result} := n \ast \text{factorial (n-1)} \\
\text{end} \\
\end{array} \]

What are \( c \) and \( g \) in this example?
1.3 (1 point) We consider the following loop equivalent for recursive definitions of the form [1] (you don't need to prove that this equivalence is correct).

\[ f(n: \text{INTEGER}) : \text{NATURAL} \]
\[ \text{local} \]
\[ i : \text{INTEGER} \]
\[ \text{do} \]
\[ \text{from} \]
\[ i := 0 \]
\[ \text{Result} := c \]
\[ \text{until} \]
\[ i = n \]
\[ \text{loop} \]
\[ i := i + 1 \]
\[ \text{Result} := g(i, \text{Result}) \]
\[ \text{end} \]
\[ \text{end} \]

Apply this transformation (literally, that is to say, purely by program transformation) to produce a recursion-free version of `factorial'. There is no need to prove anything about this transformation, just apply it as given by [2].

1.4 (10 points) From the rule for loops (question 1) and the recursion-loop equivalence ([2]), give an inference rule for proving the partial correctness of recursive functions of the form [1].

Hints:
- You need to apply the assignment axiom.
- The rule uses a notion of invariant.

1.5 (5 points) Using the rule from question 4, prove the correctness of the `factorial' function in its original form.

1.6 (5 points) What notion should be added to the above framework to yield a rule covering total correctness? (Only the name of the notion is required, no further justification or explanation. Hint: take advantage of the notion used to prove total correctness for loops.)
2 Component design and testing (15 points)

Analyze the class NETWORK_STREAM, and answer the following questions:

2.1 (9 points) For the routines make, descriptor and next_character, decide whether some designing principles for components are violated. If so, give the name of these principles and explain where and why they are violated.

2.2 (6 points) Suppose we are only working with HTTP protocol, for the feature is_url_valid, design test cases. Include what is the input and what is the expected output. You do not need to implement the feature is_url_valid.

class NETWORK_STREAM

create make

feature {NONE} -- Initialization

make (a_url: STRING; a_buffer_size: INTEGER)
    -- Initialize current network stream with URL `a_url' and with buffer size `a_buffer_size'.
    require
        a_url_attached: a_url /= Void
        a_buffer_size_positive: a_buffer_size > 0
        a_url_valid: is_url_valid (a_url)
    do
        set_url (a_url)
        set_buffer_size (a_buffer_size)
        -- Some other initialization, including initializing `buffer'.
    ensure
        url_set: url.is_equal (a_url)
        buffer_size_set: buffer_size = a_buffer_size
    end

feature -- Access

url: STRING
    -- URL associated with Current stream

buffer_size: INTEGER
    -- Size of buffer used to read data
descriptor: URL_DESCRIPTOR
    -- Descriptor for current stream, containing URL scheme information such as http, ftp.
    do
        if not is_descriptor_calculated then
            internal_descriptor := descriptor_from_url (url)
            is_descriptor_calculated := True
        end
        Result := internal_descriptor
    ensure
        result_attached: Result /= Void
    end

position: INTEGER
    -- Position in the stream

feature -- Status report

is_url_valid (a_url: STRING): BOOLEAN
    -- Is `a_url' valid?
    require
        a_url_attached: a_url /= Void
    do
        ...
    ensure
        -- Result is True if and only if `a_rul' is of correct format.
    end

feature -- Stream IO

next_character: CHARACTER
    -- Next character from the stream
local
    l_char_size: INTEGER
    do
        -- Retrieve number of bytes used to represent a character.
        l_char_size := {PLATFORM}.character_bytes

        -- Read `buffer' for new data if necessary.
        if buffer.is_empty or else buffer.count < l_char_size then
            read_buffer
        end
        -- Load a character from `buffer'.
        Result := buffer.item_as_character

        -- Increase `position' from current stream.
        position := position + l_char_size
    ensure
        position_increased: position = old position + {PLATFORM}.character_bytes
    end
**feature** -- Setting

`set_buffer_size (a_size: INTEGER)`
-- Set `buffer_size' with `a_size'.

**require**

`a_size_positive: a_size > 0`

**do**

```
...
```

**ensure**

`buffer_size_set: buffer_size = a_size`

**end**

`set_url (a_url: STRING)`
-- Set `url' with `a_url'.

**require**

`a_url_attached: a_url /= Void`
`a_url_valid: is_url_valid (a_url)`

**do**

```
...
```

**ensure**

`url_set: url /= Void and then url.is_equal (a_url)`

**end**

**feature** [NONE] -- Implementation

`buffer: BUFFER`
-- Buffer to store read data, used as cache.

`internal_descriptor: like descriptor`
-- Internal stream descriptor

`is_descriptor_calculated: BOOLEAN`
-- Has `descriptor' been calculated?

`descriptor_from_url (a_url: STRING): like descriptor`
-- Stream descriptor calculated from `a_url'

**require**

`a_url_attached: a_url /= Void`
`a_url_valid: is_url_valid (a_url)`

**do**

```
...
```

**ensure**

`result_attached: Result /= Void`

**end**
3 Program analysis (18 points)

3.1 (3 points) What kind of analysis do you need to know if a variable may be used in the following of the program before it is overwritten?

3.2 (10 points) Make control-flow graph of the following program and apply the analysis to it:

\[
\begin{align*}
\text{a := 1} \\
\text{b := a + 2} \\
\text{c := 10} \\
\text{if } b > 3 \text{ then} \\
\quad \text{c := 3 - c} \\
\text{else} \\
\quad \text{b := a} \\
\text{end} \\
\text{b := a + b} \\
\text{Result := b}
\end{align*}
\]

3.3 (5 points) How would you do to also store where the variables were defined? Explain informally.
4 Abstract Interpretation (15 points)

Consider the following language:

\[ i \in [\text{MIN\_INT}, ..., \text{MAX\_INT}] \]
\[ e::= i | e_1 \ast e_2 | e_1 + e_2 | e_1 - e_2 \]

The order of magnitude for \( \text{MAX\_INT} \) and \( \text{MIN\_INT} \) is around \( 10^{40} \) and \( -10^{40} \).

4.1 (10 points) Create an abstraction to evaluate the value of expressions and ensure that the expressions value do not go over \( \text{MAX\_INT} \) or below \( \text{MIN\_INT} \).

4.2 (5 points) Is the abstraction still valid if expressions are now defined as follows (\( \div \) is the integer division):

\[ e::= i | e_1 \ast e_2 | e_1 + e_2 | e_1 - e_2 | e_1 \div e_2 \]

If not, explain why and refine it.