utbm
université de technologie
Belfort-Montbéliard

DA53: Compilation and Language Theory - Final Exam A2022

Duration: 2h00. No document allowed.
English recommended, French accepted.

Part 1: Introduction (5 points)

Question 1.1:

Consider the PHP language. Is this programming language be a type-safe language? Explain why.

Question 1.2:

What is the name of the description (or specification) that mixes grammar productions and semantics rules? Explain briefly the concept of this description and its content.

Question 1.3:

What are the names the different parts of the program memory in a standard execution environment? Explain briefly the role of each part of the program memory.

Part 2: Tiny-LISP (10 points)

In this part, you must consider a simplified version of the functional language, inspired from LISP and named Tiny-LISP, that is explained below. In Tiny-LISP, the functions are defined as:
`(defun <name> (<param`$_1$`> <param`$_2$`> ...<param`$_n$`>) <expression>)` where:

- The parenthesis are mandatory.
- `defun` is the keyword for defining a function.
- `<name>` is an identifier that is the name of the function.
- `<param`$_i$`>` is the name of the formal parameter at the position i for the function.
- `<expression>` is the body of the function as an expression that could be built with:
  - **function calls** of the form: `(<name> <arg`$_1$`> <arg`$_2$`> ...)`. Note that parentheses are mandatory and there is no coma separator between the arguments of the function.
  - **conditional function** that evaluates an expression to boolean value; and depending on this value, evaluates the true-expression or the false-expression, e.g., `(if <expression> <true-expression> <false-expression>)`. The conditional function returns the value of the true-expression or false-expression that is evaluated;

- **arithmetic expressions** with positive floating point numbers, the operator `-` and the operator `*`. Note that the arithmetic operators are invoked as functions, e.g., `(* 1 2)` means the multiplication of 1 and 2 (parentheses are mandatory);
- **equality test expression** that tests if two values are equals, i.e. `(= <value1> <value2>)` replies if `value1` is equal or not to `value2` (parentheses are mandatory).

Any function that is defined could be invoked by using the function call syntax, defined above.

Example: Factorial

```
(defun factorial (n)
   (if (= n 0) 1
      (* n (factorial (- n 1)))))
(factorial 4)
```

Question 2.1:

Write the BNF grammar for recognizing the nonterminal <expression> for the Tiny-LISP language.

Question 2.2:

Extend the previous BNF grammar for recognizing the function definition (`defun`) for the Tiny-LISP language.

Question 2.3:

Extend the previous BNF grammar for recognizing the root function call that may be written outside the scope of the nonterminal <expression>.

Part 3: Code Generation (5 points)

In this part, you must focus on the generation of three-address code and more specifically on the "quadruple" form. The language to translate is the one described in the Part 2.

Note:

For reasons of simplicity, in the generated three-address code, you may not define the addresses themselves, but you could use the symbolic names of the variables.

Question 3.1:

In this question, you must assume that you have **NOT** obtained a syntax tree from the parsing stage. The generation of the quadruples must be directly written in the semantic rules of the SDD associated to the BNF productions that are defined in Question 2.1.

1. Define the list of the additional attributes, and their meanings, to be attached to the grammar productions in Question 2.1 and that are required to generate the "quadruples".
2. Write the code (pseudo-code is accepted) of the SDD semantic rules to generate the "quadruples" for all the nonterminals of the BNF that defined in Question 2.1.