PPL – Assignment 2

Omer Shafrir , Nadav Dvir

Question 1: Theoretical Questions

Q1.1

Q1.2

1. Special forms are required in programming languages since they require special treatment, as their names suggests. When evaluating special forms expressions , a different evaluation method is called.

An example : define expressions. An expression in the form of : define <var> <val> can not be evaluated like a "regular" expression from left to right , since at the time the expression is being evaluated there is no variable named "<var>" . regular evaluation attempt will result in an error.

1. The logical operator 'or' can be defined as a primitive operator, although such implementation will be less efficient. The logic meaning of 'or' allows the use of **shortcut semantics**.

Expression such as '<cond1> or <cond2>' will be evaluated to TRUE when at least one condition of the 2 will get the value TRUE, therefore a better design will consist of initially evaluating just the first condition. If cond1 value will be TRUE, there is no need to compute the second one, and one computation is spared. Only when the first condition will be evaluated to false , the interpreter will evaluate the second one.

Q1.3

The term **syntactic abbreviation** refers to a way one can define the operational semantic of a new expression type in a programming language without defining a new computation rule for this specific expression type. Instead, we can use a combination of other expressions that already exist in the language, and through their computation rules, get an equivalent expression which produce the same result, but without the new expression type.

First example: let expression

Every let expression can be translated to a define expression, followed by an app expression. This kind of transformation will be evaluated to the same result when transforming properly.

Second example: caar

The caar operator (on pairs) can be translated to a double car operator.

Q1.4

1. The value of the program is 3

In expressions , the initial values are computed before any of the variables become bound. In other words, gets the value of in an environment where value is 1, and not 5. Therefore, the final value being computed is 3, and this is the value of the program.

1. The value of the program is 15.

Unlike expressions, the bindings in \* are performed sequentially from left to right, in a manner that the second binding is done in an environment in which the first binding is visible, and so on. Therefore, when the y variable gets the value of , x value is 5 **and not** 1, as it would be in a let expression . Therefore , the final value is.

1. ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~





