Exercise 9

Give an SDT suitable to be implemented during top-down parsing for the language of all regular expressions on the alphabet A = {a, b, c} that, given a regular expression, construct the associated syntax tree assuming the following axioms:

$$(x^*)^* = x^* \text{ for } x \text{ in } \{a, b, c\}$$

$$((e)^*)^* = (e)^* \text{ for e regular expression}$$

In other words, for (x*)* the syntax tree that should be generated is the same tree of x* and for ((e)*)* the syntax tree that should be generated is the same tree of (e)*. For the regular expression corresponding to “epsilon” you can use the symbol “_”. For the or operation use the symbol “|” and for the concatenation operation use the symbol “.”.

Hints

Choose an LL(1) grammar that also reflects a given precedence and associativity for the operators. Then, you should define the operations to be used to construct the syntax tree together with the attributes needed to solve the problem of the axioms. Since the grammar must be LL(1), it is likely that inherited attributes are needed. In particular, a flag that records whether or not the last constructed tree has the operator * at the root may be useful.

Solution

Let’s use the following grammar, that should be checked to be LL(1). It is inspired to the version of the grammar for aritmetic expressions that is LL(1) and expresses precedence and associativity. Kleene star has precedence over “.”, which has precedence over “|”.

E::= F E
E::= “|” F E
E::= “_”
F::= H F
F::= . H F
F::= “_”
H::= (E) M

M::= “*”
M::= “_”
H::= "_"
H::= “a”
H::= “b”
H::= “c”

For constructing the syntax tree we define the following operations:

- MkT _ _: *string* x *forest* -> *tree* // construct a tree with root labelled with *string* and forest of children *forest*.
- MKE : -> *forest* // construct an empty forest
- Add _ _ : *tree* x *forest* -> *forest* // add *tree* as first tree of the forest *forest*

For the SDT we use the following attributes:

- **in** for E, F, M; it is inherited and contains the representation of the first operand (already traversed but not visible directly to the symbols E, F, M).
- **instar** for E, F, M; it is inherited and contains a boolean on/off set on whenever the first operand is a Kleene star
- **tree** for all non-terminals; it is synthesised and contains the pointer to the syntax tree for the expression associated to the symbol
- **star** for all non-terminals; it is synthesised and contains a boolean on/off set on whenever the expression associated to the symbol is Kleene starred

E::= **F** {E.in:= F.tree; E.instar:= F.star;} **E**{E.tree:= E.tree; E.star:= E.star}
E_1::= "|" **F** {E_2.in:= MkT("|", Add(E_1.in, Add(F.tree, MKE()))); E_2.instar:= "off"} **E_2** {E_1.tree:= E_2.tree; E_1.star:= "off"}
E::= "_" {E.tree:= E.in; E.star:= E.instar; }
F::= **H** {F.in:= H.tree; F.instar:= H.star} **F**{F.tree:= F.tree; F.star:= F.star}
F_1::= **.H** {F_2.in:= MkT(".", Add(F_1.in, Add(H.tree, MKE()))); F_2.instar:= "off"} **F_2** {F_1.tree:= F_2.tree; F_1.star:= F_2.star}
F ::= "_" {F.tree = F.in; F.star = F.instar}
H::= (**E** {M.in:= E.tree; M.instar:= E.star}) **M** {H.tree:= M.tree, H.star:= M.star}
M::= * {M.tree:= if (M.instar="on") then M.in else MkT("*", Add(M.in,MKE())); M.star:="on"}
M::= "_" {M.tree:= M.in; M.star:= M.instar}
H::= a {H.tree:= MkT("a",MKE()); H.star:= "off"}
H::= b {H.tree:= MkT("b",MKE()); H.star:= "off"}
H::= c {H.tree:= MkT("c",MKE()); H.star:= "off"}