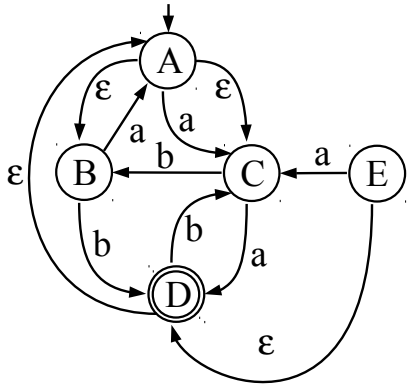
June 27, 2014

Compilers

Surname, Name	
Student identifier	

1. After generating the DFA equivalent to the following NFA, specify the BNF expressing the regular language relevant to the DFA.



2. Given the following grammar G in BNF notation, we ask to transform G into an equivalent non left-recursive grammar G* and then, based on the complete parsing table, determine whether G* is LL(1).

S → *S* **a** *T* | *T*
T → *S* **b** | **a** | **b**

3. Given the following BNF, we ask to generate the complete LR(1) parsing table and to establish whether the grammar is LR(1).

S → *S* **a** *T* | **ε**
T → **a** | **b** | **ε**

4. A language of boolean expressions is defined by the following ambiguous BNF:

program → *expr*
expr → *expr* **and** *expr* | *expr* **or** *expr* | **not** *expr* | **(** *expr* **)** | **true** | **false**

Assuming that **not** has highest precedence and right associativity, **and** has intermediate precedence and left associativity, while **or** has lowest precedence and left associativity, we ask to specify in *Yacc* the interpreter of the language, which is required to print the result (either "`true`" or "`false`") of the expression phrase.
Note: The grammar specified in the translation rules of *Yacc* shall be equal to the given BNF.

5. Specify the attribute grammar relevant to the following BNF,

program → *rec-def* *rec-assign*
rec-def → **def id : record (** *attr-list* **)**
attr-list → *attr* **,** *attr-list* | *attr*
attr → **id :** *type*
type → **int** | **string** | **bool**
rec-assign → **id := record (** *const-list* **)**
const-list → *const* **,** *const-list* | *const*
const → **intconst** | **strconst** | **boolconst**

```
def r: record (a: int, b: string, c: bool)
r := record (12, "omega", true)
```

based on the following semantic constraints:

- The name of the defined record shall equal the name of the assigned record;
- Attribute names shall be unique;
- The attribute values in the assignment shall be consistent with the attribute types in the definition.

6. Given the language for the manipulation of integers, defined by the following BNF,

program → *stat-list*
stat-list → *stat* *stat-list* | *stat*
stat → **id** := *expr*
expr → *expr* + *expr* | *expr* * *expr* | *expr* **and** *expr* | *expr* **or** *expr* | **not** *expr* | **(** *expr* **)** | **id** | **num**

assuming a concrete syntax tree where nodes are structured by the following fields:

- `Symbol symbol`: the grammar symbol,
- `char *lexval`: lexical value,
- `child`: pointer to first child,
- `brother`: pointer to right brother,

we ask to specify a procedure of P-code generation based on the following requirements:

- Logical operators are based on the same rules of the C programming language (0 stands for **false**, while a number different from 0 stands for **true**);
- Operands are evaluated from left to right;
- Logical **and** is evaluated in short circuit, (while logical **or** is fully evaluated);
- The language of the P-machine includes the following set of instructions:

LDA <id>	(loading of address of variable **<id>** on stack)
LOD <id>	(loading of value of variable **<id>** on stack)
LDC <const>	(loading of integer constant **<const>** on stack)
PLUS	(arithmetic addition)
TIMES	(arithmetic multiplication)
AND	(conjunction)
OR	(disjunction)
NOT	(negation)
GOFALSE <label>	(conditional jump)
GOTO <label>	(unconditional jump)
LABEL <label>	(implicit address)
STO	(store)
HALT	(program termination: to be generated as the final instruction)