Compiler Construction/Fall 2014/Homework 4

Eva Rose
evarose@cs.nyu.edu

Kristoffer Rose
krisrose@cs.nyu.edu

Assigned Thursday 9/25/2014, due Thursday 10/2/2014 at 8am

Reading Assignments

- Lecture on 9/25/2014 (this homework): Dragon book 2.3, 5.1-5.3 (30p); HACS handout 4 (the H4 link on the course schedule).
- Lecture on 10/2/2014: Dragon book 1.6, 2.7 (15p).

Homework Assignments

The following assignments should be submitted for a maximum of 25 points.

1 Syntax-Directed Definitions (SDD)

Question 1.1 (6 points)**.** Consider the SDD for the desk calculator from the Dragon book:

PRODUCTION	SEMANTIC RULE
$L \to E_1$ **\$**	$L.val = E_1.val$
$E \to E_1 + T_2$	$E.val = E_1.val + T_2.val$
$E \to T_1$	$E.val = T_1.val$
$T \to T_1 * F_2$	$T.val = T_1.val \times F_2.val$
$T \to F_1$	$T.val = F_1.val$
$F \to (E_1)$	$F.val = E_1.val$
$F \to$ **digit**	$F.val =$ **digit**$.lexval$

1. Extend the SDD to handle division ("/").
2. Draw the annotated parse tree for $3 + 8/2$ \$.

Question 1.2 (9 points)**.** For each of the following SDDs, explain whether they are S-attributed and/or L-attributed.

1. For synthesized attribute s, and inherited attribute i:

PRODUCTION	SEMANTIC RULES
$A \to BC$	$A.s = B.i$; $B.i = C.s$
$C \to BAC$	$C.s = A.s$
...	...

2. For synthesized attribute z and given an externally defined scheme IfZero:

PRODUCTION	SEMANTIC RULES
$E \rightarrow$ **(** * E_1 E_2 **)**	$E.z = E_1.z \vee E_2.z$
| **(** + E_1 E_2 **)**	$E.z = E_1.z \wedge E_2.z$
| **(** – E_1 **)**	$E.z = E_1.z$
| **num**	$E.z = \text{IfZero}(\textbf{num}.sym, \textbf{true}, \textbf{false})$

3. For inherited attribute d and a synthesized attribute n:

PRODUCTION	SEMANTIC RULES
$E \rightarrow F_1$	$F_1.d = 0; E.n = F_1.n$
$F \rightarrow$ **(** F_1 **)**	$F_1.d = F.d + 1; F.n = F_1.n$
| $\textbf{id}_1$	$\textbf{id}_1.n = F.d$

2 Syntax Directed Translation in Hacs

In this exercise, we will study and extend the following HACS script for simple boolean expressions (also available through the Bool.hx link).

```
module "edu.nyu.cs.cc.Bool" {

  // Syntax.
  sort B
  | ⟦ T ⟧@4 | ⟦ F ⟧@4                    // constants
  | sugar ⟦ ( ⟨B#⟩ ) ⟧@4 →B#             // parenthesis
  | ⟦ ⟨B@2⟩ | ⟨B@1⟩ ⟧@1                 // disjunction
  | ⟦ ⟨B@3⟩ & ⟨B@2⟩ ⟧@2                  // conjunction
  | ⟦ ! ⟨B@3⟩ ⟧@3                        // negation
  ;

  // Semantic operations.
  sort B;
  | scheme And(B,B) ;
  And(⟦T⟧, #2) →#2 ;
  And(⟦F⟧, #2) →⟦F⟧ ;

  | scheme Or(B,B) ;
  //MISSING: rules for Or

  | scheme Not(B) ;
  Not(⟦F⟧) → ⟦T⟧ ;
  Not(⟦T⟧) → ⟦F⟧ ;

  // Propagation.
  attribute ↑b(B);
  sort B;
  ↑b;          // associate b attribute to (current) E sort
```

```
  // Evaluation scheme
  sort B;
  | scheme Evaluate(B) ;
  Evaluate(B# ↑b(#b)) →#b ;

  ⟦ ⟨B#1 ↑b(#b1)⟩ & ⟨B#2 ↑b(#b2)⟩ ⟧↑b( And(#b1, #b2) ) ;
  //MISSING: synthesis for |−expression.
  ⟦ ! ⟨B# ↑b(#b)⟩ ⟧↑b( Not(#b) ) ;
  ⟦ T ⟧ ↑b(⟦T⟧);
  ⟦ F ⟧ ↑b(⟦F⟧);
}
```

Question 2.1 (5 points). Use HACS to compile the above boolean grammar and run HACS to show the output of parsing the term `((T)&!(F|T))`. Explain why the output is different from the input.

Question 2.2 (5 points). Replace the two lines marked "MISSING" in the code with proper HACS rules to complete the system to also handle disjunction. Show that you can then run your system as follows:

```
$ make Bool.run
$ ./Bool.run --sort=B --action=Evaluate --term='((T)&!(F|T))'
F
```