ΕΠΛ323 - Θεωρία και Πρακτική Μεταγλωττιστών

Lecture 9b

Syntax-directed Translation

Elias Athanasopoulos
eliasathan@cs.ucy.ac.cy

Translation Scheme (for Predictive Translator)

PRODUCTION	SEMANTIC RULES
$E \rightarrow E_1 + T$	$E.nptr$:=*mknode*('+', $E_1.nptr$, $T.nptr$)
$E \rightarrow E_1 - T$	$E.nptr$:=*mknode*('-', $E_1.nptr$, $T.nptr$)
$E \rightarrow T$	$E.nptr$:=$T.nptr$
$T \rightarrow (E)$	$T.nptr$:=$E.nptr$
$T \rightarrow$ **id**	$T.nptr$:=*mkleaf*(**id**, **id**.*entry*)
$T \rightarrow$ **num**	$T.nptr$:=mkleaf(**num**, **num**.val)

Recall: Left-recursion Elimination (with semantic rules)

```
A→Aa|b
A→A₁Y { A.a := g(A₁.a, Y.y) }
A→X  { A.a := f(X.x) }
```

```
A→bR, R→aR|ε
A→X { R.i := f(X.x)}
   R { A.a := R.s }
R→Y { R₁.i := g(R.i, Y.y)}
    R₁{ R.s := R₁.s }
R→ε{ R.s := R.i }
```

Translation Scheme (Left-recursion Eliminated)

```
E → T     { R.i := T.nptr }
    R     { E.nptr := R.s }
R → +
    T     { R1.i := mknode('+', R.i, T.nptr) }
    R1    { R.s := R1.s }
R → -
    T     { R1.i := mknode('-', R.i, T.nptr) }
    R1    { R.s := R1.s }
R → ε     { R.s := R.i }
T → (
    E
    )     { T.nptr := E.nptr }
T → id    { T.nptr := mkleaf(id, id.entry) }
T → num   { T.nptr := mkleaf(num, num.val) }
```

Translation Scheme (merge +, - to **addop**)

```
E → T     { R.i := T.nptr }
    R     { E.nptr := R.s }
R → addop
    T     { R1.i := mknode(addop.lex, R.i, T.nptr) }
    R1    { R.s := R1.s }
R → ε     { R.s := R.i }
T → (
    E
    )     { T.nptr := E.nptr }
T → id    { T.nptr := mkleaf(id, id.entry) }
T → num   { T.nptr := mkleaf(num, num.val) }
```

Predictive Translator Methodology

1. For each nonterminal A, construct a function that has a formal parameter for each inherited attribute of A and that returns the values of the synthesized attributes of A. For simplicity, we assume that each nonterminal has just one synthesized attribute. The function for A has a local variable for each attribute of each grammar symbol that appears in production for A.
2. The code for nonterminal A decides what production to use based on the current input symbol.

Predictive Translator Methodology

3. The code associated with each production does the following. We consider the tokens, nonterminals, and actions on the right side of the production from left to right.
   - For token X with synthesized attribute x, save the value of x in the (local) variable declared X.x. Then generate a call to match token X and advance the input.
   - For nonterminal B, generate the assignment c := B(b_1, b_2, …, b_k) with a function call on the right side, where b_1, b_2, …, b_k are the variables of the inherited attributes of B and c is the variable for the synthesized attribute of B.
   - For an action, copy the code into the parser, replacing each reference to an attribute by the variable for that attribute.

Example

- R (slide 5) has
  - An inherited attribute, R.i (R_1.i depends on R.i, which is its parent)
  - A synthesized attribute, R.s (depends on R_1.s, child, and R.i, shelf)
- Prototype (generic)

  R.s R(R.i)
- R.s and R.i are pointers to AST's nodes (as returned by `mknode()`)
- Prototype (actual)

```
struct ast_node * R(struct ast_node *i);
```

Predictive Translator

```
struct ast_node * R(struct ast_node *i){
      struct ast_node *nptr, *i1, *s1, *s;

      if (lookahead == addop) {
            /* R → addop T R */
            addop.lex = lexval;
            match(addop);
            nptr = T();
            i1 = mknode(addop.lex, i, nptr);
            s1 = R(i1);
            s = s1;
      } else
            s = i; /* R→ε  */
      return s;
}
```