Excel@FIT

65

Error Recovery during Top-Down Parsing: Acceptable-sets derived from continuation

Continuation Grammar

Supposed that when error is detected, part of the input has already been processed. This part is a prefix *u* of a sentence *uv*. *Uv* is a sentence in language, so the only thing that has to be done, is to find string {*v*, that is continuation of *u*. The continuation computes as follows:
Continuation is a sentence, which has to be produced using the fewest productions steps. The trick is to find the quickest end for each nonterminal, in other words to find which right-hand side leads the fastest to string composed only of terminals. It is possible to compute it in advance, using a number called step count.
For each nonterminal (left-hand side) the right-hand side with lowest step count is picked.
These rules form a continuation grammar. Continuation grammar doesn't have to form a proper grammar, it is used only for error recovery.

Algorithm

1. replace nonterminal:
First nonterminal on the stack is found. Using only continuation grammar, the nonterminal is replaced by its right-hand side. Each terminal from the right-hand side of the rule is added to the acceptable set. Also, the First set of derived nonterminal is added to the the acceptable set.
2. skip unacceptable tokens:
Zero or more tokens form the input are skipped, until a symbol from acceptable set is found. Since ending symbol (in this paper it is terminal end) is acceptable, this step is not infinite.
3. resynchronize the parser:
Parser tries to continue. If it is possible to make a move, parsing can normally continue. If parser can't continue, modified parser has to continue as follows:

a) nonterminal on the top of the stack:
If nonterminal is on top of the stack and there is no grammar rule to use, algorithm repeats from 1.

b) terminal on the top of the stack:
If there is a terminal on top of the stack and it cannot be matched with the input symbol, expected symbol is inserted. Since every token has its value, implicit value is given to each inserted symbol (e.g. 0 to int, err_id to id). Step 3 is repeated until the parser is resynchronized.

Example

```
begin
   a==b;
end
```

Sequence of tokens is: `T_begin T_id T_equal T_equal T_id T_semicolon T_end`. Error is on the second line: extra equal. The recovery with my modification works as follows:
Parser starts parsing the input.

stack	input	rule
<PROGRAM>	T_begin	1
end<BODY>begin	T_begin	POP
end<BODY>	T_id	3
end<BODY>;<STATEMENT>	T_id	6
end<BODY>;<EXPRESSION>=id	T_id	POP
end<BODY>;<EXPRESSION>=	T_equal	POP
end<BODY>;<EXPRESSION>	T_equal	

There is no grammar rule that can be used. Parser cannot continue, error recovery has to start.

Replace nonterminal:
First terminal, <EXPRESSION>, is found. It is replaces by its right-hand side, using continuation grammar:
<EXPRESSION> -> <TERM><EXP>,
<TERM> -> int,
<EXP> ->ε
Acceptable set = { (, int, id }.

Skip unacceptable tokens:
In the input is T_equal that is not in acceptable set, token is skipped. In the input is T_id that is in acceptable set. Token is not skipped.

Terminal on the top of the stack:
Terminal T_int is on the top of the stack. It can be matched with the input.

Error recovery is finished, parsing can continue.

Alena Obluková, xobluk00@stud.fit.vutbr.cz