
To INF 5110, spring 2005
Comments and errata for Kenneth C. Louden:
COMPILER CONSTRUCTION, Principles and Practice

Page 53:

Last line: "figure 2.4" should be "figure 2.2".

Page 54:

Paragraph 2, line 2: Section 2.2.4 does not exist.

Page 88:

4th line from the bottom: Section 2.6.4 does not exist.

Page 116:

Footnote: Section 3.2.7 does not exist.

Page 155:

The following changes should be done to the program:
Line 3: Omit the second part of the test (otherwise the algorithm
may stop too early)
The last if-test: Omit the first part of the test (it is redundant)

Page 159:

In Figure 4.3 a second step doing substitution is missing.

Page 168/169:

We used a more basic definition of First(X), when X is a nonterminal:

First(X) is the set of terminals that can start a string derived from
X. In addition, epsilon is in First(X) if the empty string can be derived
from X. First(alfa) is defined accordingly.

Thus the theorem at page 169 is correct by definition.
As an algorithm for finding First(X), we simply used the definition
given at the middle of page 168, with the addition that step 2 is
repeated until nothing more happens (and we did not look at the
details of figure 4.6).

Page 173/174:

We used a more basic definition of Follow(X) (X a nonterminal):

Follow(X) is the set of terminals that can follow X in a string
derived from (the string) "S$", where S is the start symbol and the
set of terminals is extended with the "end-of-input marker" '$'.

As an algorthim for finding Follow(X), we used the definition at the
middle of page 173, where steps 2 and 3 are repeted until nothing more

happens (and we did not look at the details of Figure 4.8).

Page 177

Example 4.14: The set "Follow(stmt)" should also contain '$'

Page 202:

Line 4 from bottom. Add the following: An item where alfa is the empty string ("A -> . ") is both an initial item and a complete item.

Page 207:

There are minor corrections to the LR(0) parsing algorithm description. See foil 5.16C.

Page 210:

There are minor corrections to the SLR(1) parsing algorithm description. See foil 5.19.

Page 277/278:

Addition to the definition of synthesized attributes:
A synthesized attribute A.a is also allowed to depend on inherited attributes of A. (Note, however, that such dependencies will not occur in S-attributed grammars, as they only have synthesized attributes).

Page 278

New definition of "inherited attribute": An attribute is "inherited" if it is defined by a semantic rule as part of a symbol on the right hand side of a production.

Page 285

Example 6.15, line 6 of the code: In "EvalWithBase(..)", the second parameter is missing. However, this parameter is not used when T is a "basechar" node, and its value is therefore irrelevant.

Page 290:

Additional requirement in the definition of L-attributed grammars: In the associated equiations for a_j, the attributes a_1, ..., a_k of X_0 must all be INHERITED attributes. That is, a_j can NOT depend on synthesized attributes of X_0. Thereby the definition itself covers the essence of the condition given in a sentence below: "Given an L-attributed grammar [in which the .... (delete)...], a recursive descent ..." and it can be deleted (as indicated).

Page 295:

In section 6.3 we stressed that one could also use (part of) the syntax-tree itself as the symbol table. Then the 'lookup' operation will be a search process "upwards" in the tree, from where you currently are in the tree/program. This search should be done according to the visibility rules of the language. The 'insert' and

'delete' operations will then normally not be used, and they will be replaced by adding declaration nodes to the tree during buildup, and by changing the current position in the tree. Figure 6.17 can be seen as a version of this, where the declaration list of each block in the syntax tree is taken out as a (hash) table of its own.

Page 329

Figure 6.22: The "exp"-rules are missing from the grammar (but see next page)

Page 329/330:

As indicated at the middle of page 329, Table 6.10 is not an attribue grammar in the nomal sense, in that e.g. ":=" is used, indicating that it is more like an algorithm. In this connection it should also be added that the computation of the attributes should be done from left to right in the tree, so that e.g. the insert-operation (in the first rule) is called for exactly the right declarations whenever the lookup-operation is called. All together it would probably have been better to express this table as a recursive procedure going through the tree from left to right.

Page 359:

Comments to the the given "calling sequence":

1. At the start of the code for the procedure itself (that is, in the code executed after step 5), space should be allocted for the local varibles of the procedure (and if necessary they should be initialized).

2. Steps 1, 2, and 3 of the exit steps should be done at the end of the code for the procedure, while step 4 should be done by the caller (probably immediately after the call instruction).

3. If an "access link" is included (see page 367), this should be computed and pushed on the stack between step 1 and 2 of the calling sequence. It should be computed as "fp.al.al....al", where the number of "al-steps" (al = access link) is the number of block levels between the caller and the block enclosing the called procedure.

Page 362

To make the figure in Example 7.6 fit with the Ada program at the top of the page, we must assume that the array A is transmitted by taking a full copy, and we must change the local variable "temp" to the declaration of an integer variable "i".

Page 380/381:

In section 7.4.4 we stressed that in the algorithms discussed there is an underlying assumption that when arriving at an object through a pointer (or during a sequential sweep through memory) it is always possible to find the SIZE of the object and at WHICH RELATIVE ADDRESSES IT CONTAINS POINTERS (as opposed to intergers, reals etc.). One way to arrange for this is to have a pointer at a fixed relative

address in all objects, pointing to a statically allocted package containing information common to e.g. all objecs of a given class. This package could contain the size and pointer-positions in the objects of this class. (Then, also the virtual function table discussed at page 376 could be included in this package of information).

Page 427

Paragraph 1, line 5 and line 11: "lod field_offset(x,j)" should be "ldc field_offset(x,j)" since field_offset(..) is returning a constant.

Page 428

Paragraph 1, line 2: "lod field_offset(*p, lchild)" should be "ldc field_offset(*p, lchild)" since field_offset(..) is returning a constant.

Page 433

10th line from the bottom: "lod FALSE" should be "ldc FALSE" (FALSE is a constant).

- - - o o o - - -