SDT for L-attributed definitions

Assuming a pre-order traversal of the parse tree we can transform a L-attributed SDD in a SDT as follows:

1. action computing inherited attributes must be computed before the occurrence of the non terminal. In case of more inherited attributes for the same non terminal order them as they are needed
2. actions for computing synthesized attributes go at the end of the production

Example

Consider the production:

$$S \rightarrow \textbf{while}\ (C)\ S_1$$

assuming the "traditional" semantics for this statement let's generate the intermediate code assuming a three-address code where three control flow statements are generally used:

- `ifFalse` x `goto L`
- `ifTrue` x `goto L`
- `goto L`

Intermediate Code Structure

Example

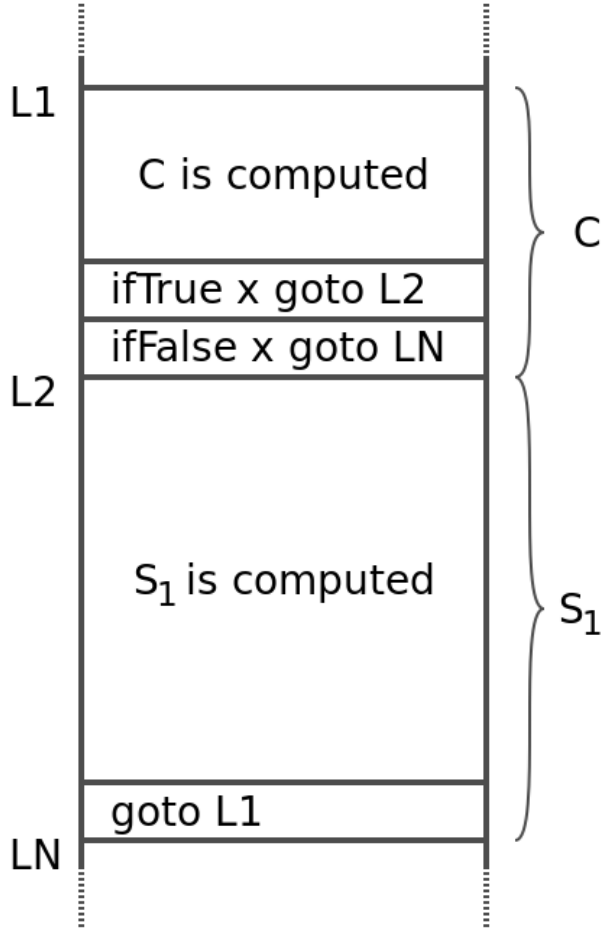
Consider the production:

$$S \rightarrow \textbf{while}\ (C)\ S_1$$

assuming the "traditional" semantics for this statement let's generate the intermediate code assuming a three-address code where three control flow statements are generally used:

- `ifFalse` *x* `goto L`
- `ifTrue` *x* `goto L`
- `goto L`

Intermediate Code Structure



while statement - rationale

The following attributes can be used to derive the translation:

- *S.next*: labels the beginning of the code to be executed after *S* is finished
- *S.code*: sequence of intermediate code steps that implements the statement *S* and ends with *S.next*
- *C.true*: label for the code to be executed if *C* is evaluated to true
- *C.false*: label for the code to be executed if *C* is evaluated to false
- *C.code*: sequence of intermediate code steps that implements the condition *C* and jumps to *C.true* or to *C.false* depending on the evaluation

while statement - SDD and SDT

SDD

$S \rightarrow \textbf{while}\ (C)\ S_1$	$L1 = new();$
	$L2 = new();$
	$S_1.next = L1;$
	$C.false = S.next;$
	$C.true = L2$
	$S.code = \textbf{label}\|\|L1\|\|C.code\|\|\textbf{label}\|\|L2\|\|S_1.code$

while statement - SDD and SDT

Note for the translation:

- L_1 and L_2 can be treated as synthesized attributes for dummy nonterminals and can be assigned to the first action in the production

SDT

$S \rightarrow$ **while** (	$\{L1 = new(); L2 = new(); C.false = S.next;$
	$C.true = L2; \}$
C)	$\{S_1.next = L1; \}$
S_1	$\{S.code =$ **label**$\|\|L1\|\|C.code\|\|$**label**$\|\|L2\|\|S_1.code\}$

while statement - SDD and SDT

Note for the translation:

- L_1 and L_2 can be treated as synthesized attributes for dummy nonterminals and can be assigned to the first action in the production

SDT

$S \rightarrow$ **while** (	$\{L1 = new(); L2 = new(); C.false = S.next;$
	$C.true = L2;\}$
C)	$\{S_1.next = L1;\}$
S_1	$\{S.code =$ **label**$\|\|L1\|\|C.code\|\|$**label**$\|\|L2\|\|S_1.code\}$

Implementing L-attributed SDD

Translation can be performed according to two different strategies:

- traversing a parse tree
- during parsing

Traversing a parse tree

- Build the parse tree and annotate; if the SDD is not circular there is at least an order of execution that works
- Build the parse tree, add actions, and execute the actions in preorder; e.g. L-attributed SDDs translated into SDTs

During parsing

- Use a recursive descent parser
- Generate code on the fly
- Implement an SDT in conjunction with an LL-parser
- Implement an SDT in conjunction with an LR-parser

Implementing L-attributed SDD

Translation can be performed according to two different strategies:

- traversing a parse tree
- during parsing

Traversing a parse tree

- Build the parse tree and annotate; if the SDD is not circular there is at least an order of execution that works
- Build the parse tree, add actions, and execute the actions in preorder; e.g. L-attributed SDDs translated into SDTs

During parsing

- Use a recursive descent parser
- Generate code on the fly
- Implement an SDT in conjunction with an LL-parser
- Implement an SDT in conjunction with an LR-parser

Implementing L-attributed SDD

Translation can be performed according to two different strategies:

- traversing a parse tree
- during parsing

Traversing a parse tree

- Build the parse tree and annotate; if the SDD is not circular there is at least an order of execution that works
- Build the parse tree, add actions, and execute the actions in preorder; e.g. L-attributed SDDs translated into SDTs

During parsing

- Use a recursive descent parser
- Generate code on the fly
- Implement an SDT in conjunction with an LL-parser
- Implement an SDT in conjunction with an LR-parser