

Project 1

Installation:

read info for installation
TYPE needed
st.install()
ST20, ST22
st.load()
ST20, lib (not a)
Char $\rightarrow$ TRUESIGNEDCHAR
type by
buildbasic(77000...)
see top of types.c
to see mapping of basic
Pascal types to C types.

make.c $\rightarrow$ routines for
spitting out assembler
backgend routines called from
make.c or from grammar

Skip to Chap 5 (Syntax-
directed translation)
(Chap 4 later).

Syntax-directed definition
(SDD):

Grammar together with
attributes for grammar
symbols and rules
for computing value of
attributes.
("Annotated grammar")

Mar 21-11:01 AM

Grammar

Rules (semantics)

$E \rightarrow E_1 + T$ $E.val := E_1.val + T.val$
 $E \rightarrow T$ $E.val := T.val$
 $T \rightarrow T_1 * F$ $T.val := T_1.val * F.val$
 $T \rightarrow F$ $T.val := F.val$
 $F \rightarrow (E)$ $F.val := E.val$
 $F \rightarrow \text{const}$ $F.val := \text{const}, \text{literal}$

$(2 + 3) * 4$ $F = 20$

attribute digraph (distinct
from the parse tree)

Type declaration (C-style)

$D \rightarrow T \mid L$
 $T \rightarrow \text{int}$
 $T \rightarrow \text{real}$
 $L \rightarrow \text{id}$
 $L \rightarrow L_1 \mid L_2$

SDD set id.type to be the
type of the id.

int a, b, c

Mar 21-11:20 AM

int a, b, c

$D \rightarrow T \mid L$ $L.type := T.type$
 $T \rightarrow \text{int}$ $T.type := \text{"int"}$
 $T \rightarrow \text{real}$ $T.type := \text{"real"}$
 $L \rightarrow \text{id}$ $L.type := \text{id.type}$
 $L \rightarrow L_1 \mid L_2$ $L.type := L_1.type$
 $L.type := L_2.type$

Two restrictions for an SDD:

- Each rule is associated with a single production and only mentions attr of symbols in that production.

in parse tree

children
edges confined to those
"immediate families"
parent + children

- No cycles in the
dependency graph

Dependency graph:
vertices are attributes of
nodes in the parse tree
directed edge $u \rightarrow v$
if some rule computes
v attr directly in terms of
u attr.

Mar 21-11:39 AM

General SDD

Build dependency digraph
Topologically sort vertices
 $V_1, V_2, V_3, \dots, V_n$
Compute V_i value in order
of increasing i using rules.

Two types of attr:

Synthesized: occurs as the
attribute of the head of
the production in a rule
(only) defining to

$E.val := E_1.val + E_2.val$

↑
synthesized

A synthesized attribute only
depends on attr of children
in parse tree

Inherited: attr that when
defined, is associated with
a symbol in the body of
the production
can depend on attr of
parent and/or siblings
(or other attr of same symbol)

$T.type$ — synthesized
 $L.type, id.type$ — inherited

Cycles in dep graph:
 $A \rightarrow \dots B$ { $A.i := B.i$ }
 $B \rightarrow \dots A$ { $B.i := A.i + 1$ }

Mar 21-11:56 AM

An SDD is an S-attribute
definition if all attr are
synthesized.

Guarantees all edges in dep
graph go up.
 $\therefore$ acyclic.
 $\therefore$ easy to compute on
the fly with a
bottom-up parser
(bison, e.g.)

bison really only is set up
for computing synthesized
attr

More generally, an L-attribute
definition is one where
each attr is either

- synthesized
- depends on an
inherited attr of
parent and/or attr
of siblings to the
left.
- depends on another attr
of same symbol in
an acyclic way

Type def SDD is L-attribute

You can hack bison to
handle an L-attr SDD

Mar 21-12:08 PM