Finishing Syntax-directed definitions & translation

~~Intermediate actions in bison~~

~~Top-down parsing~~

~~Project description~~

SDD (most general form)

Given a grammar G
Associate with each production a set of rules of the form

$$b := f(a_1, \ldots, a_k)$$

where $a_1, \ldots, a_k$ and b are attributes of symbols in the production.
f is some function that computes b in terms of the values of $a_1, \ldots, a_k$.
We say attribute b depends on $a_1, \ldots, a_k$ (individually)

G is an annotated grammar (an SDD)

[k could be ∅:

rule looks like

$$b := c$$

where c is some constant expr
(b depends on no attributes)

Terminal symbols can also have attributes. These may be provided by the lexical analyzer (synthesized) or inherited, depending on other attrs in the production.

On some parsable input w
Consider the complete parse tree T yielding w.

Each node of T has some number of attributes associated with it.
Rules determine dependencies between attributes
(local: between parent/child or siblings in T)

Determine order of attr computation by dependency constraint:
For every attr b in T, all attrs that b depends on must be computed before b.

Directed graph (dependency graph) with vertices being all attrs of all nodes of T, and edges are direct dependencies

i.e., $u \longrightarrow v$

means "v depends directly on u"

dependency graph must be acyclic (i.e., a dag)
(only constraint)

If dag, then topologically sort it. Arranges vertices in order

$v_1, v_2, \ldots, v_n$

so that if $v_i \longrightarrow v_j$ is any edge, then $i < j$.

Compute attrs in order by increasing i:
v_1 first
v_2
v_3
⋮

Usually not done in practice because it requires the entire input to be parsed before any attrs are computed.

Feb 23-10:56 AM

Restricted SDDs:

1. S-attributed definitions
2. L-attributed definitions

S-attributed means synthesized attributes only.

Advantage: attrs can be computed very promptly, as soon as a node's subtree is parsed. Also, you can "forget" all attrs of nodes below the root of any subtree, once the root's attrs are computed.

This is what bison/yacc is designed for.

Disadvantage: can't have inherited attrs.

Recall:

D → T L | L.type = T.type
T → int | real
L → id | id.type = L.type
L → L, id | L1.type = L.type

2. L-attributed definitions. more general than an S-attributed def.

- all synthesized attributes are still allowed
- inherited attrs are allowed provided they could be computed promptly based on a left-to-right reading of the input.

In other words all arrows in the depndency graph go either up or "to the right"

In an L-attributed def, each attribute can depend on the following only:

- a synthesized attr of a child
- any attr of a left sibling
- any attr of the parent that does not depend on a right sibling or the node itself

Disallowing

Intermediate actions in bison

```
session : { printf("reduce"); }
        expr_list
        ;
expr_list :
        expr_list expr
      | /* empty definition */
```

Generally:

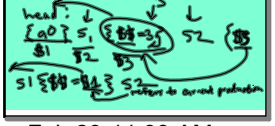
head : { a1 } S2 { a2 } S3 { reduce action }

bison converts each intermediate action into a reduce action as follows: converts to

head : a0 S1 a1 S2 a2 S3 { reduce action }

where a0, a1, a2 are new nonterminals created by bison. bison adds the following productions

a0 : { a0 }
a1 : { a1 }
a2 : { a2 }



Feb 23-11:33 AM