**Basic Idea of First Symbol Computation**

**When Exactly Does Recursive Descent Work?**

When can we be sure that recursive descent parser will parse grammar correctly?

* it will accept without error exactly when string can be derived

Consider grammar without repetition construct \* (eliminate it using right recursion).

Given rules

X ::= p

X ::= q

that is,

X ::= p | q

where p,q are sequences of terminals and non-terminals, we need to decide which one to use when parsing X, based on the first character of possible string given by p and q.

* first(p) - first characters of strings that p can generate
* first(q) - first characters of strings that q can generate
* requirement: first(p) and first(q) are **disjoint**

How to choose alternative: check whether current token belongs to first(p) or first(q)

**Computing 'first' in Simple Case**

Assume for now

* no non-terminal derives empty string, that is:

For every terminal X, if X =⇒\* w and w is a string of terminals, then w is non-empty

We then have

* first(X ...) = first(X)
* first(”a” ...) = {a}

We compute first(p) set of terminals for

* every right-hand side alternative p, and
* every non-terminal X

Example grammar:

S ::= X | Y

X ::= "b" | S Y

Y ::= "a" X "b" | Y "b"

Equations:

* first(S) = first(X|Y) = first(X) $\cup$first(Y)
* first(X) = first(”b” | S Y) = first(”b”) $\cup$first(S Y) = {b} $\cup$first(S)
* first(Y) =

**How to solve equations for first?**

Expansion: first(S) = first(X) $\cup$first(Y) = {b} $\cup$first(S) $\cup${a} $\cup$first(Y)

* could keep expanding forever
* does further expansion make difference?
* is there a solution?
* is there unique solution?

Bottom up computation, while there is change:

* initially all sets are empty
* if right hand side is bigger, add different to left-hand side

Solving equations

* first(S) = first(X) $\cup$first(Y)
* first(X) = {b} $\cup$first(S)
* first(Y) = {a} $\cup$first(Y)

bottom up

|  |  |  |
| --- | --- | --- |
| **first(S)** | **first(X)** | **first(Y)** |
| {} | {} | {} |
| {} | {b} | {a} |
| {a,b} | {b} | {a} |
| {a,b} | {a,b} | {a} |
| {a,b} | {a,b} | {a} |

Does this process terminate?

* all sets are increasing
* a finite number of symbols in grammar

There is a unique **least** solution

* this is what we want to compute
* the above bottom up algorithm computes it

General Remark:

* this is an example of a ‘fixed point’ computation algorithm
* also be useful for semantic analysis, later