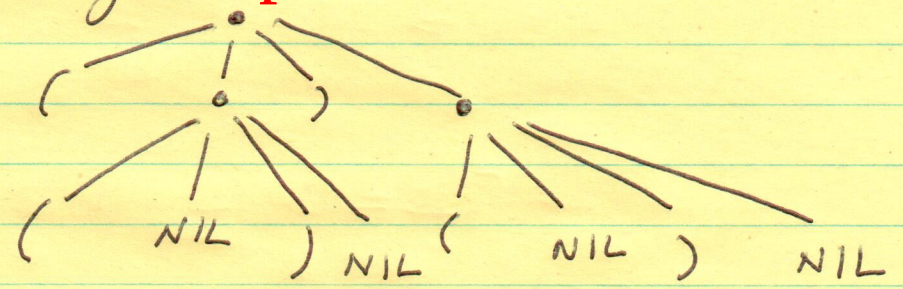
Parsing

Consider the following unambiguous gramm
for parentheses

$$S \rightarrow (S)\ S \mid \varepsilon$$

This is an inductive definition so a parser for this can be naturally written as a recursive descent parser.
We assume we have a datatype Tree with constructor MakeTree which expects 4 arguments. For an input string (())()
we build the tree




```
Tree parseString (st)
  { T = parseS(st)
    if not eos(st) then error("Extra tokens", st)
    else return(T); }
```

```
Tree parseS(st)
  { if first(st) = '(' then
      { advance(st);
        Sleft = parseS(st);
        if (~~token~~ first(st) = ')') then
          { advance(st);
            Sright = parseS(st); }
        else error("expecting closing paren"); &&
→ return (makeTree('(', Sleft, ')', Sright)) };
  else
      return (NIL); }
```


I have not written exhaustive error handling.

LL(1) grammars use a one-symbol lookahead to decide which rule to use.

LR(1) grammars work bottom up & also use a one symbol lookahead.

LALR is a hybrid & is the most popular choice.