Today’s topics

Language Translation
Revising the Syntax Rules and Notation

Upcoming
Language Translation
Generating Code

Reading
Great Ideas, Chapters 9

Importance of Language

- Vehicle for programming
- Use for human and machine communications
- Syntax Rules
  - First pass already covered
  - Need to refine to notation
    - Must be suitable for machine to do
  - In addition, need to deal with the meaning
- Also, should see Levels or Layers in dealing with computer
  1. Hardware
  2. Machine language
  3. Assembler
  4. Java (or other high-level language)
  5. Application (e.g. Word, Excel, Filemaker, …)

Language Translation

- Goal is to automatically
  - Translate from Java:
    z = x + y;
  - to Assembler:
    copy ax, x
    add ax, y
    copy z, ax
- What is the meaning we are looking for?
  Machine gives assembler statements meaning because the machine knows what to do with them (after trivial translation to binary). E.g., the machine knows what add means.

Revise Syntactic Rules

- Need to revise Syntactic Production Rules
  - New rule:
    R1: <n>j -> a sequence of letters and/or digits that begin with a letter
  - Replaces (have seen this before):
    R1: <name> -> a sequence of letters and/or digits that begin with a letter
  - The new R1 says “change <n>j into a sequence of letters and/or digits that begin with a letter”
- Use rules to modify strings
  - For syntactic productions, must end up with valid Java Programs
Using Syntax Rules

- **Examples using R1:**
  - `<n>3` → `x`
  - or
  - `<n>6` → `data`
  where “n” stand for “name”

- **Further use of R1:**
  - `( <n>3 + <n>6 )`
    - Use `<n>3` and `<n>6` above to get
    - `(x + data)`

- **More Rules:**
  - R2 : `<e>i` → `<n>j`
  - Where “e” stands for “expression”
  - Example:
    - `<e>1` → `<n>3`

**Using Syntax Rules**

- **and**
  - R3: `<s>1` → `<n>2` = `<e>3`
  - Where “s” stands for “statement”
  - It says “<s>1” can be replaced by “<n>2 = <e>3;”
  - **Can now do:** `ans = data;`

  **derivation rule**
  - `<e>1` → `<n>2`
  - `<n>2` = `<e>3`
  - `<n>2` → `ans`
  - `ans` = `<e>3`
  - `<e>3` → `( <e>4 + <e>5 )`
  - `ANS = (X + (Y * Z));`
  - (notice shorthand/simplifications used)

**More Rules**

- **Need two more rules to make it worthwhile**
  - R4: `<e>1` → `( <e>7 + <e>8 )`
  - R5: `<e>1` → `( <e>7 * <e>8 )`
  - These are additional rules for expressions

- **Can now handle** `ANS = (X + (Y * Z));`

**Longer Example**

```
derivation rule
s1
n2 = e3;  R3: s1 → n2 = e3;
ANS = e3;  R1: n2 → ANS
ANS = (e4 + e5);  R4: e3 → (e4 + e5)
ANS = (n6 + e5);  R2: e4 → n6
ANS = (X + e5);  R1: n6 → X
ANS = (X + (e7 * e8));  R5: e5 → (e7 * e8)
ANS = (X + (n9 * e8));  R2: e7 → n9
ANS = (X + (Y * e8));  R1: n9 → Y
ANS = (X + (Y * n10));  R2: e8 → n10
ANS = (X + (Y * Z));  R1: n10 → Z
```
Notes

- **Abbreviations**
  - Just omitted the angle brackets. Could do this because the notation remained *unambiguous*.

- **Role of the subscripts**
  - The subscripts are required to make sure each term is *unique*.
  - Simplest technique is to start at one and increment every time a different subscript is needed.

- *Simple substitution is all that is required!!!*
  - If you are doing something *more than that*, it is probably *wrong*!

- **The notation and form are important**
  - You will be expected to match them on tests.