Language Translation

From Java to Assembly

Compilers

- A program that:
  - Checks the syntax of a language. (Makes sure each line of code adheres to the rules of the language.)
  - Translates the high-level language (Java) into machine language.

Production Rules

To study language, we have to learn the mathematics of symbols.
Production Rules categorize words (or symbols) and show how they are put together into sentences.

Sentence ➔ Noun Verb Noun

Translation Problem

Java or some other high-level language

Z = X + Y;

Assembly/Machine Language

COPY AX,X
ADD AX,Y
COPY CN1,AX
COPY AX, CN1
COPY Z,AX
**Production Rules**

Rule example:
Identifier: any sequence of alphanumeric symbols that begins with a letter

**Instantiation** example:
Identifier: x

**Example Rule Set**

Rule 1: An identifier is a word
R1: <i>j ➔ any word, or sequence of alphanumeric symbols, beginning with a letter
Examples: num, x, y, index, counter
<i>3 ➔ num

Rule 2: An expression can be an identifier
Example:
y is an identifier and an expression
R2. <e>1 ➔ <i>j

Rule 3: A statement is an identifier that equals an expression
Example:
y = x
R3. <s>1 ➔ <i>j = <e>k
**Example Rule Set**

Rule 4: An expression can be an expression plus an expression

Example:
\[ y = (x + z) \]

R4. \[ \langle e \rangle_i \Rightarrow (\langle e \rangle_j + \langle e \rangle_k) \]

**Example Rule Set**

Rule 5: An expression can be an expression times an expression

Example:
\[ y = (x * z) \]

R4. \[ \langle e \rangle_i \Rightarrow (\langle e \rangle_j * \langle e \rangle_k) \]

**Derivations**

- We can now **prove** that a statement is legal according to the rules.
- We can prove two ways – start with symbols and end with a statement, or vice versa.
- It’s usually easier to start with the statement and work backwards

**Example: \( Y = (XX + YY) \)**

\[<s>1\]
\[<i>2 = <e>3 \quad R3. <s> \Rightarrow <i> = <e>\]
\[Y = <e>3 \quad R1. <i> \Rightarrow Y\]
\[Y = (<e>4 + <e>5) \quad R4. <e> \Rightarrow <e> + <e>\]
\[Y = (<i>6 + <e>5) \quad R2. <e> \Rightarrow <i>\]
\[Y = (XX + <e>5) \quad R1. <i> \Rightarrow XX\]
\[Y = (XX + <i>7) \quad R2. <e> \Rightarrow <i>\]
\[Y = (XX + YY) \quad R1. <i> \Rightarrow YY\]
So far…

- We can check code to see if it has legal syntax. Syntax is simply the order of symbols.
- Checking syntax is the first job of the compiler.
- We have tools to study language.
- We have a way to define a language, called a grammar. A grammar is a set of production rules that tells what strings are allowable in a language.

Adding Meaning

- Semantics refer to the meanings of the symbols in a language.
- Now we learn how to do the compiler’s second job: turn a high level language into a low level language.

Semantics

- Meaning tags – tell how to attach meaning to the syntax.
- In this case, meanings are assembly code.
- When all of the substituting is done, we have the translated program.

Back to the Rules

R1: <i> \rightarrow w 
M(<i>) = w
R2: <e> \rightarrow <i>
M(<e>) = M(<i>)
code = nothing
R3: <s> \rightarrow <i> = <e>
code(<s>) = code(<e>)
COPY AX, M(<e>)
COPY M(<i>), AX
**Examples**

- The only way to understand this!
- Table H, page 273

**...And Natural Language**

- Syntactically correct: Man Bites Dog.
- Semantically correct: Dog bite mens.
- Both semantically and syntactically correct: A dog bit a man.
- Can you translate computer language ➔ English automatically???
- Can you translate English ➔ French automatically?