innovate achieve lead

BITS Pilani
Pilani Campus

# Syntax Directed Definition for Synthesized and Inherited Attributes


Dr. Shashank Gupta
Assistant Professor
Department of Computer Science and Information Systems

# Dependency Graph

Dependency Graphs are the most general technique used to evaluate syntax directed definitions with attributes.

- A Dependency Graph shows the interdependencies among the attributes of the various nodes of a parse-tree.
  - There is a node for each attribute;
  - If attribute b depends on an attribute c there is a link from the node for c to the node for b ( b ← c).

**Dependency Rule**: If an attribute b depends from an attribute c, then we need to fire the semantic rule for c first and then the semantic rule for b.

# Inherited Attributes

An inherited attribute is one whose value is defined in terms of attributes at the parent and/or siblings.

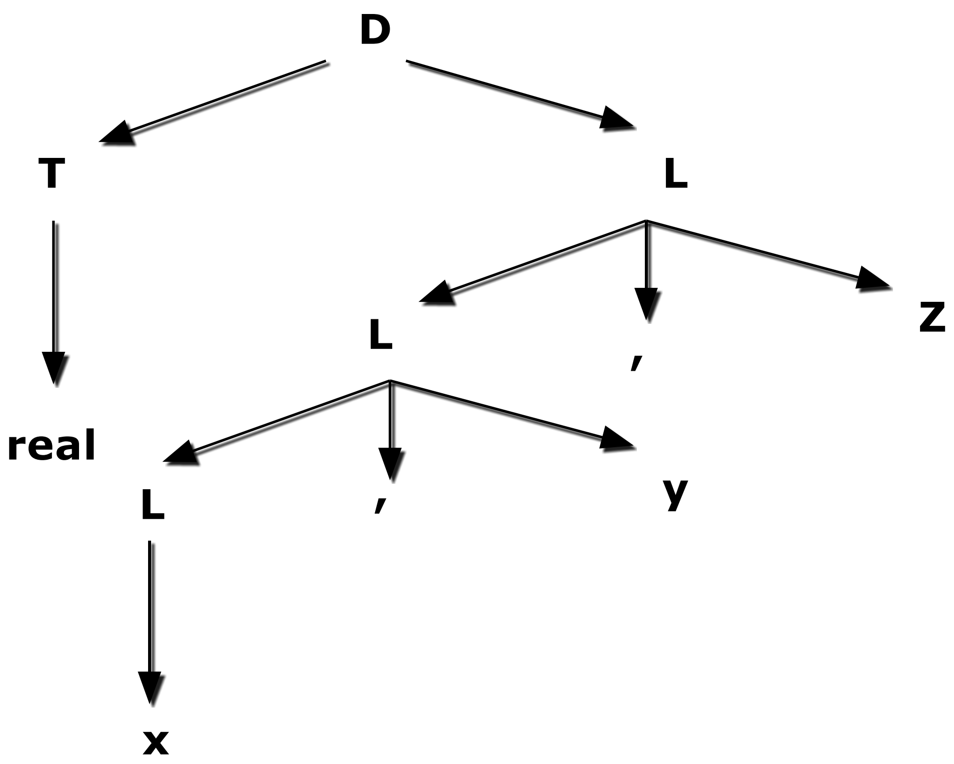
- Used for finding out the context in which it appears

Possible to use only S-attributes but more natural to use inherited attributes

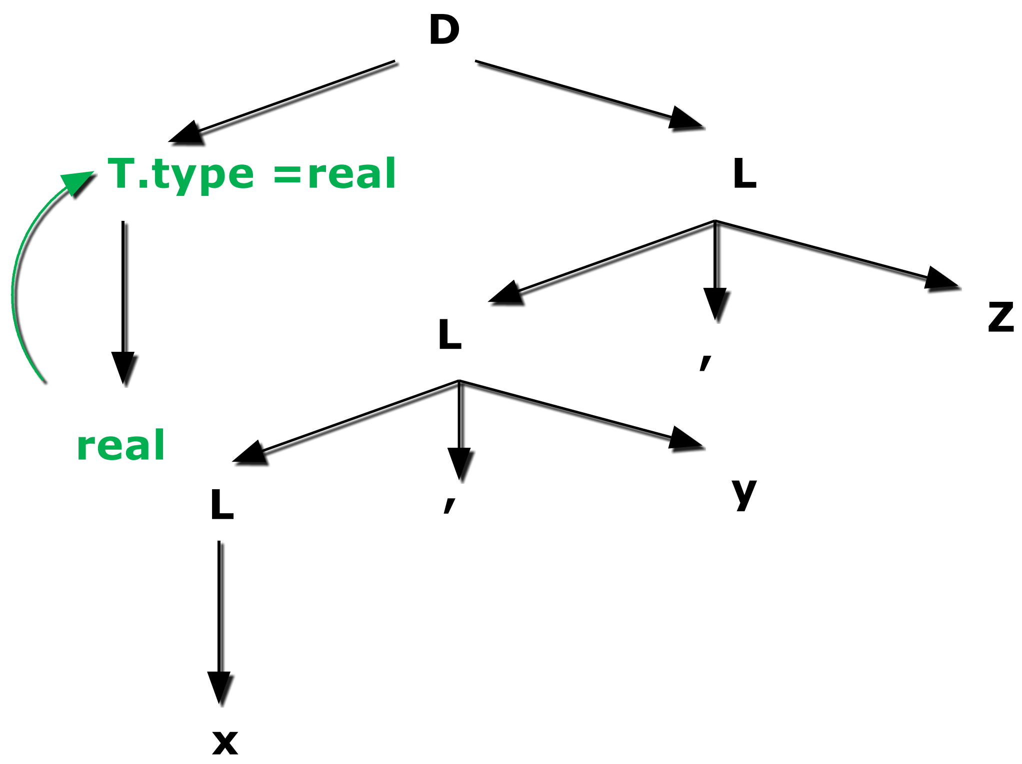
# CFG for Type Information

- $D \rightarrow T\ L$
- $T \rightarrow real$
- $T \rightarrow int$
- $L \rightarrow L_1\ ,\ id$
- $L \rightarrow id$

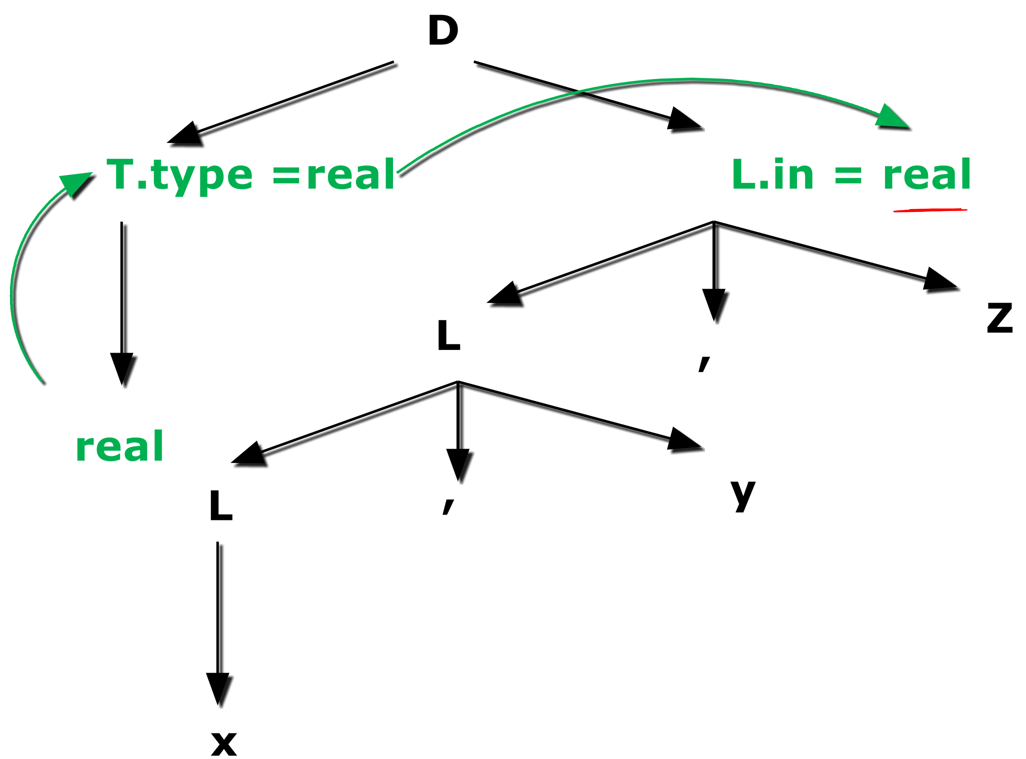
# Parse Tree for real x, y, z



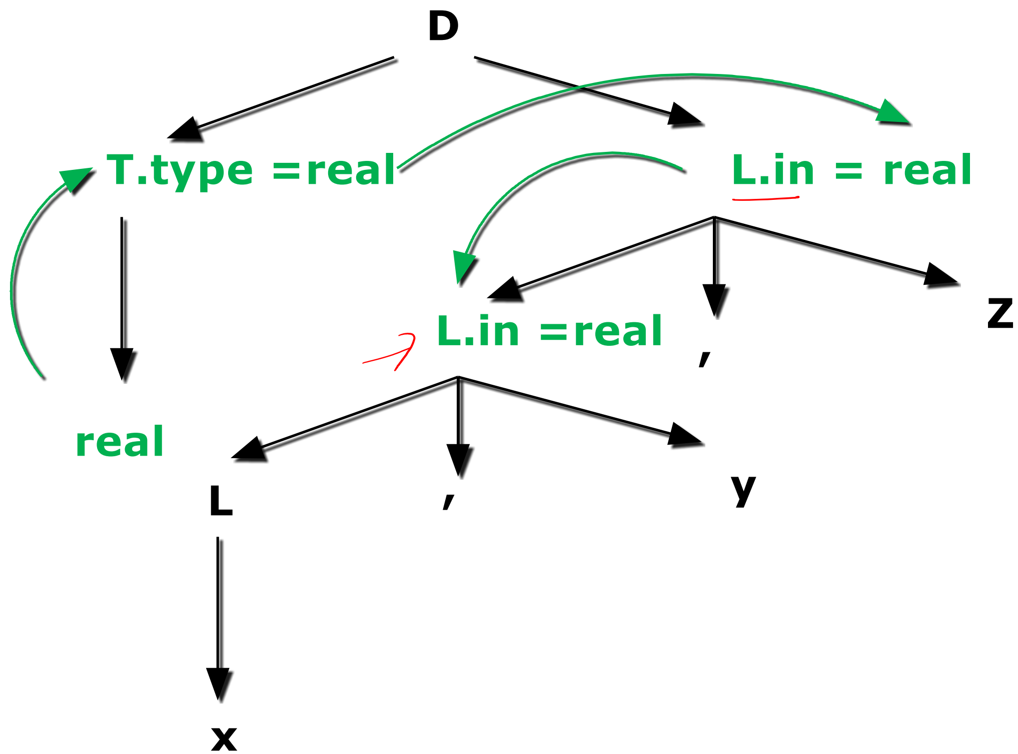
# Dependence Graph



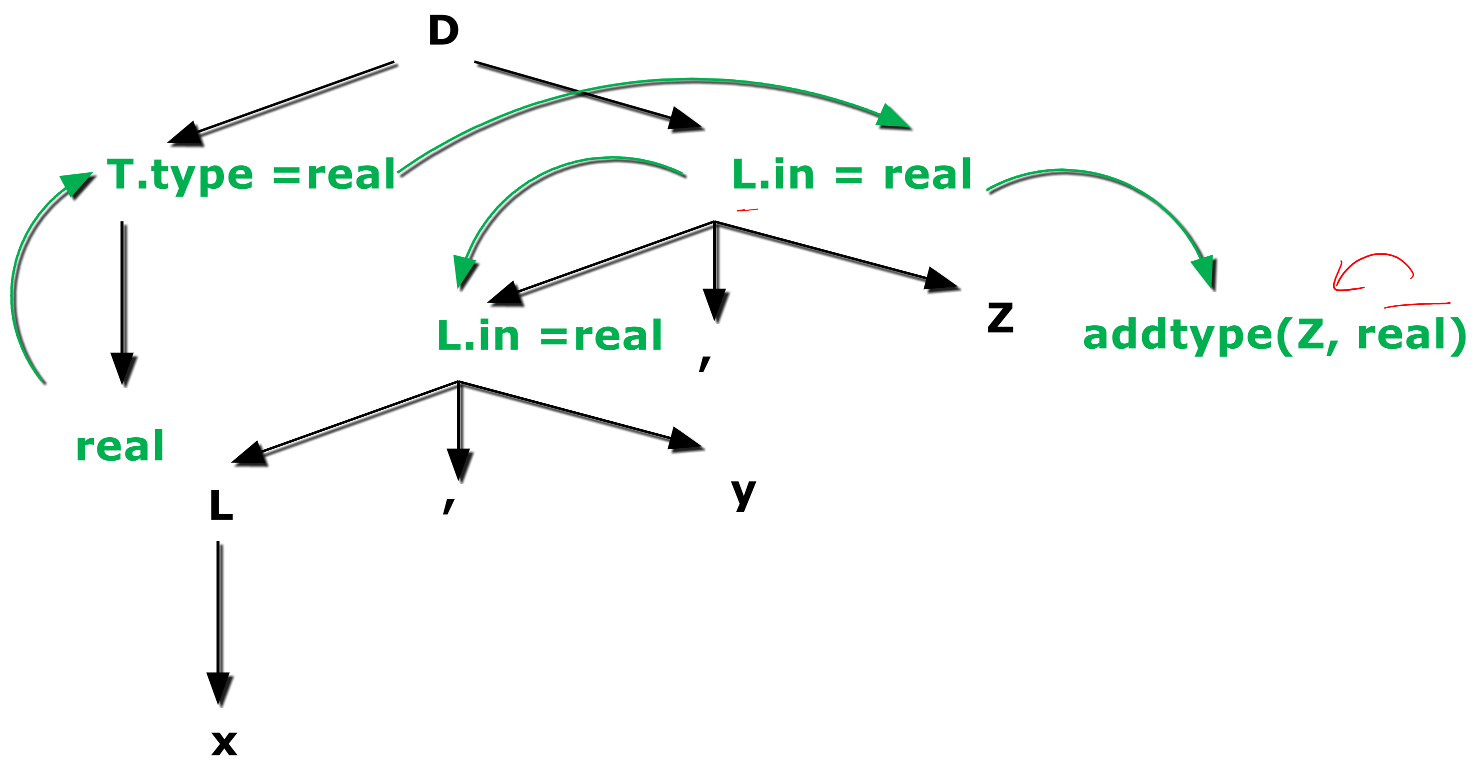
# Dependence Graph



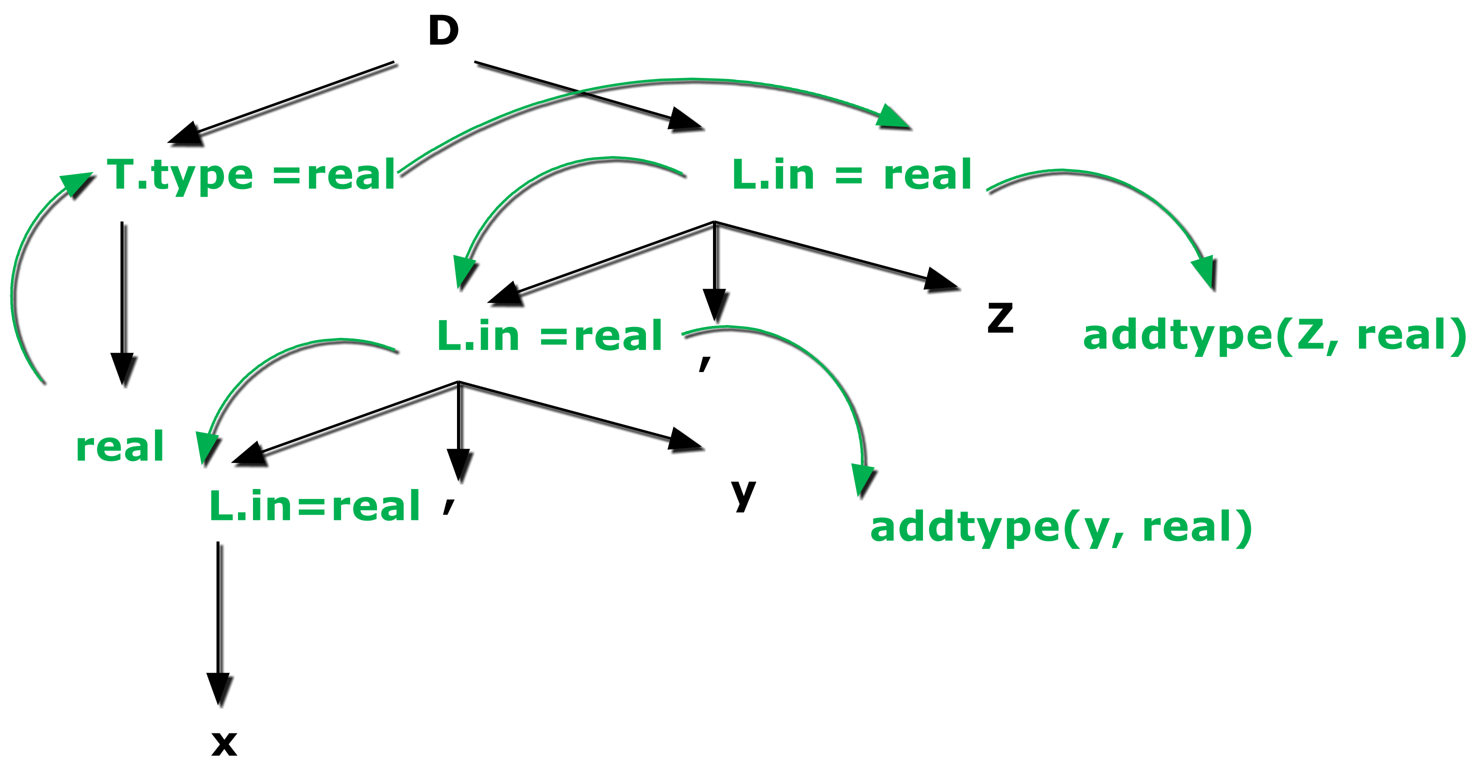
# Dependence Graph



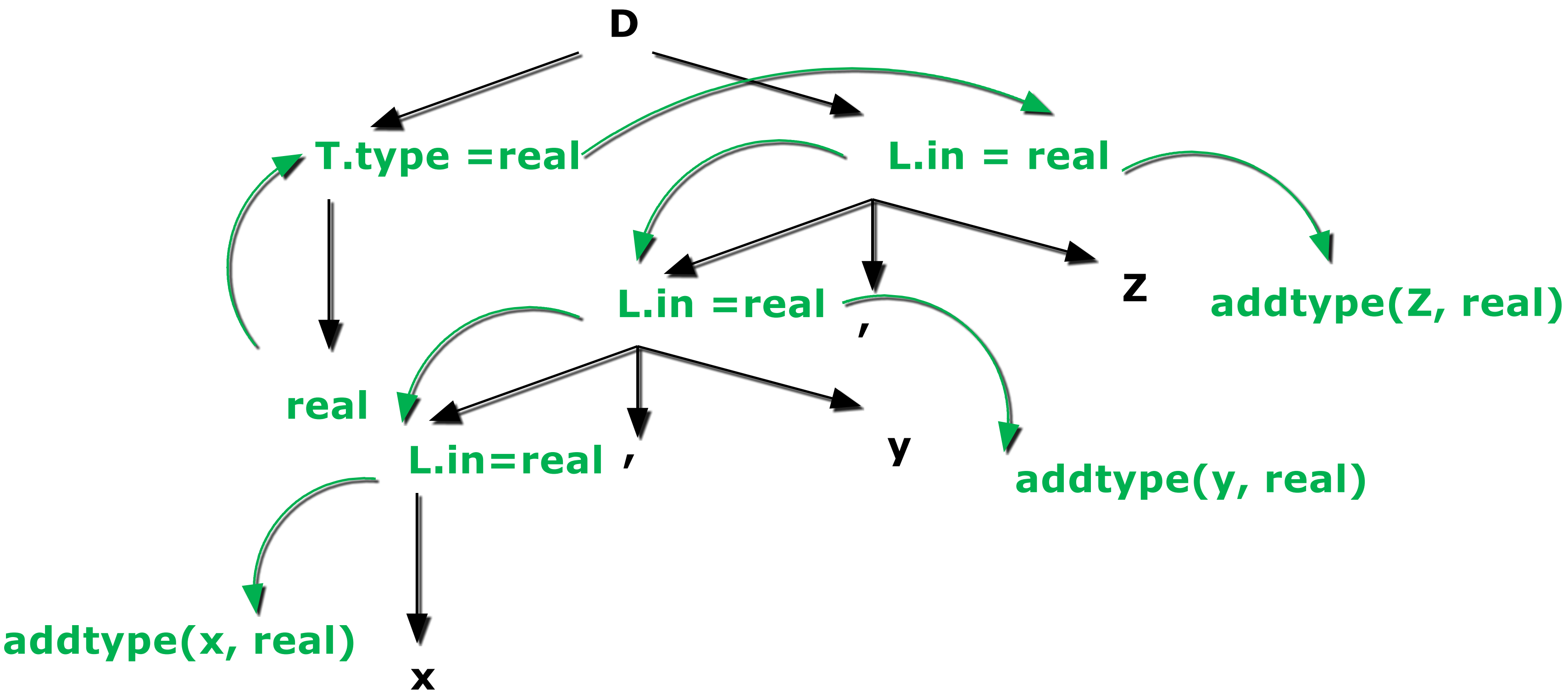
# Dependence Graph



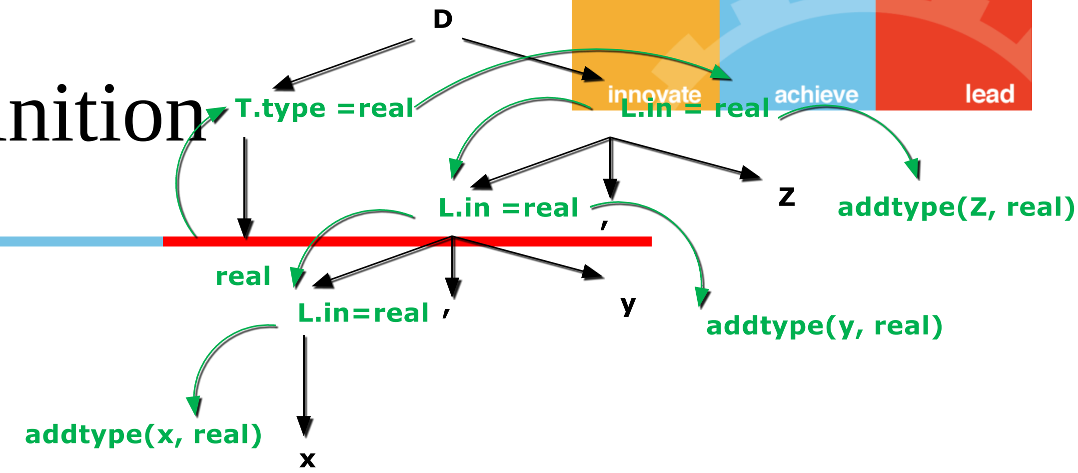
# Dependence Graph



# Dependence Graph



# Syntax Directed Definition




- D → T L  L.in = T.type
- T → real  T.type = real
- T → int  T.type = int
- L → $L_1$ , id  L1 .in = L.in

  addtype(id.entry, L.in)
- L → id  addtype (id.entry,L.in)

# Example

Consider the following grammar for signed binary numbers

$$number \rightarrow sign \;\; list$$

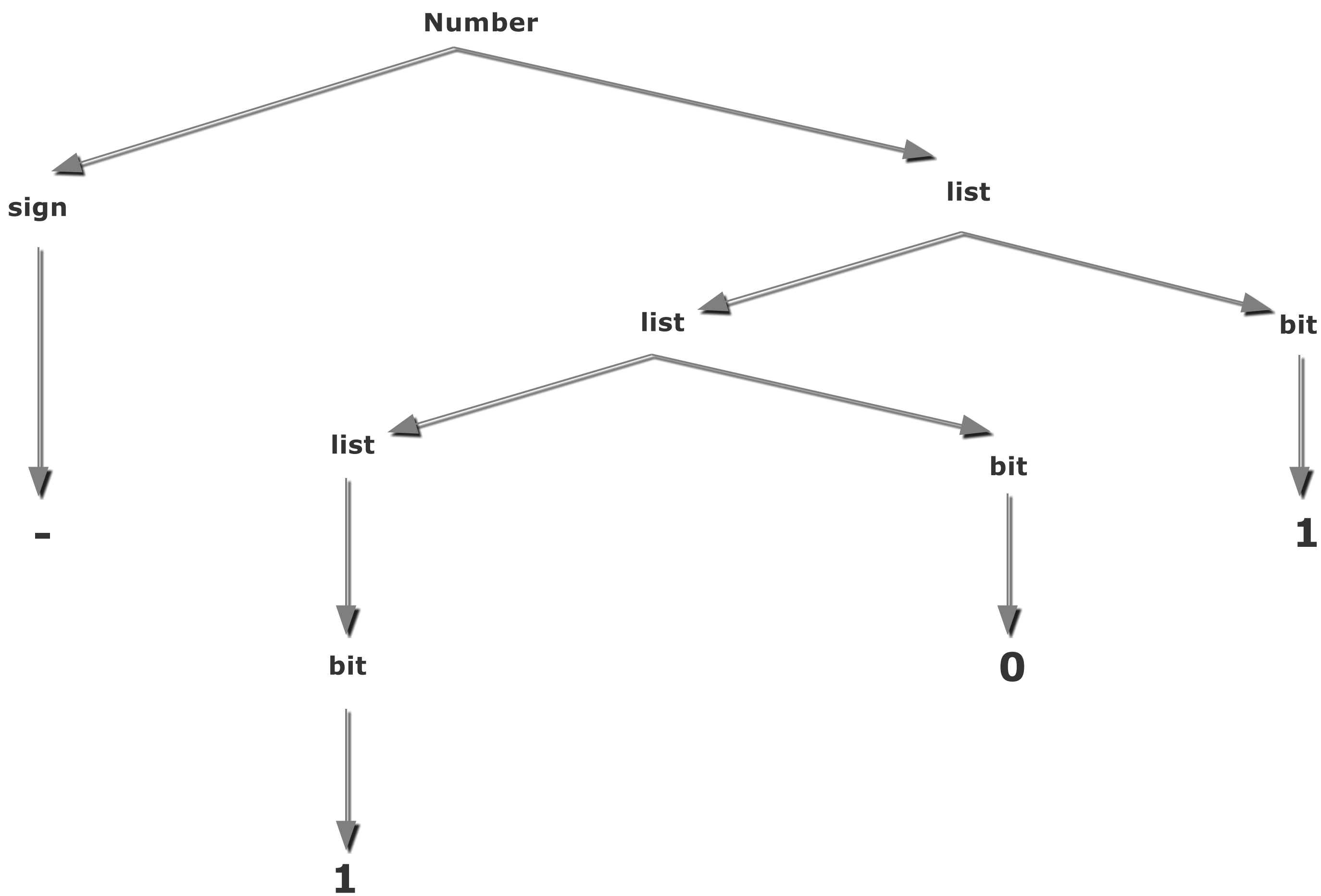
$$sign \rightarrow + | -$$

$$list \rightarrow list \;\; bit \; | bit$$

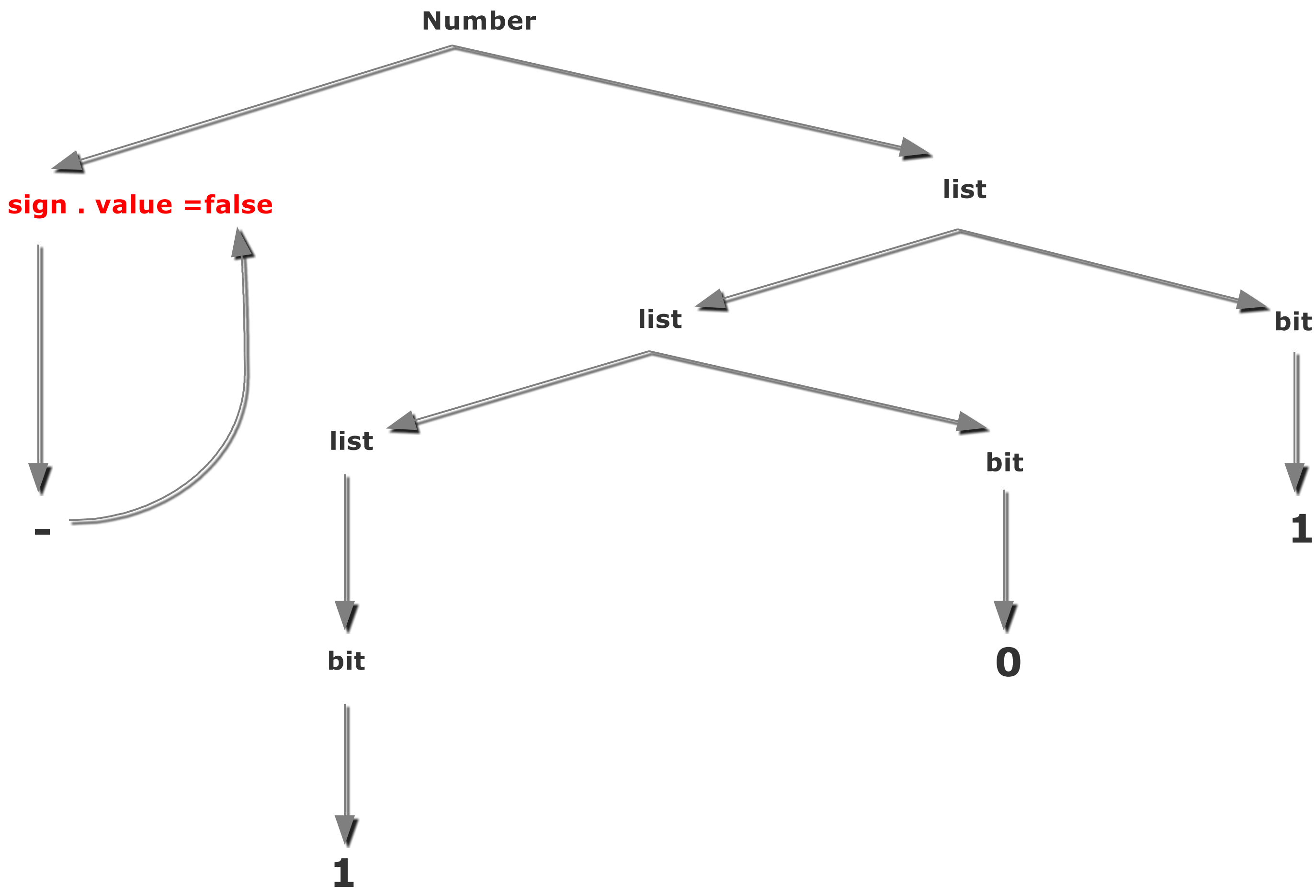
$$bit \rightarrow 0 | 1$$

Build an attribute grammar that annotates number with the value it represents. Write the semantic rules associated with each of the production rules and also, show the step-by-step construction of parse tree.

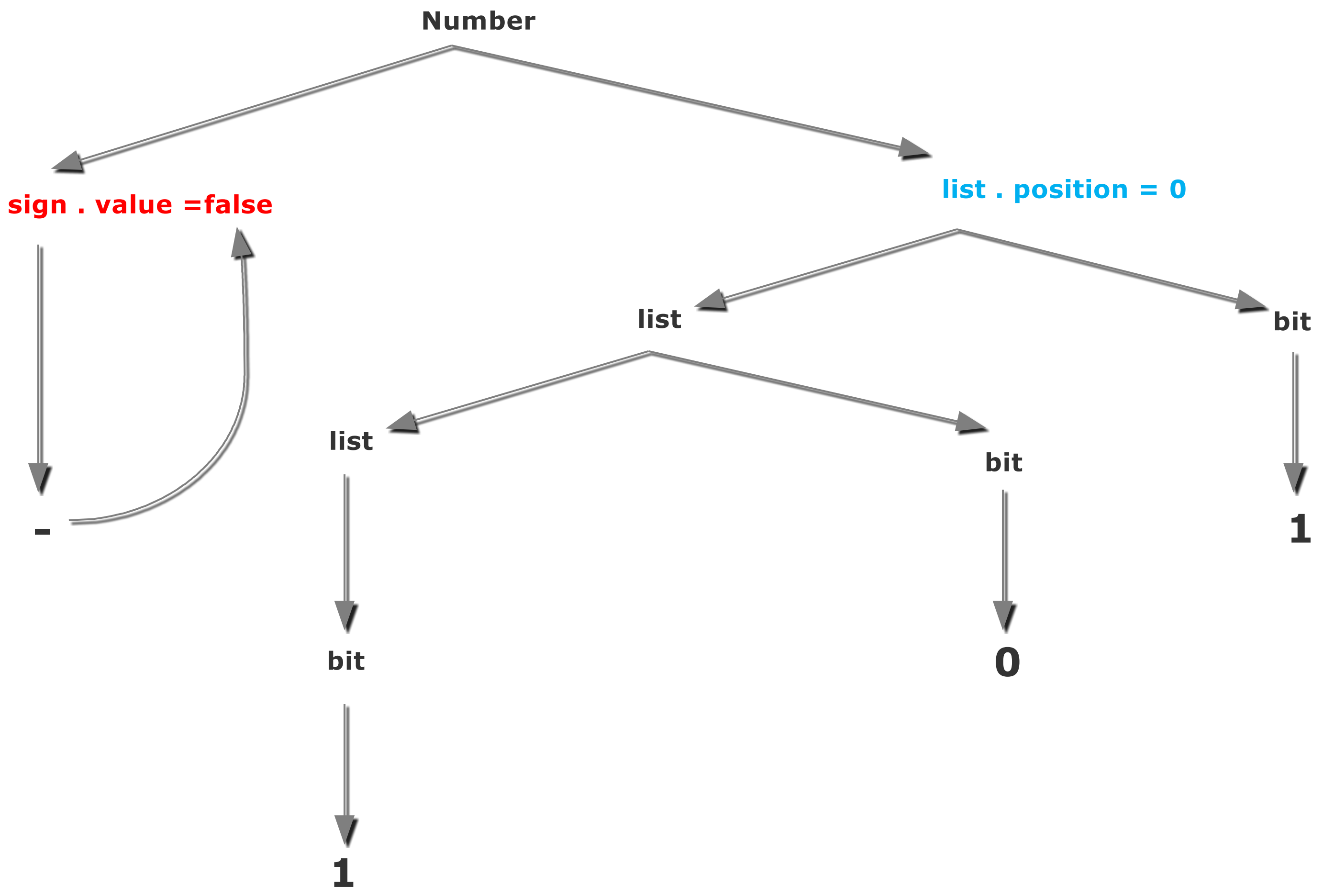
# Parse Tree



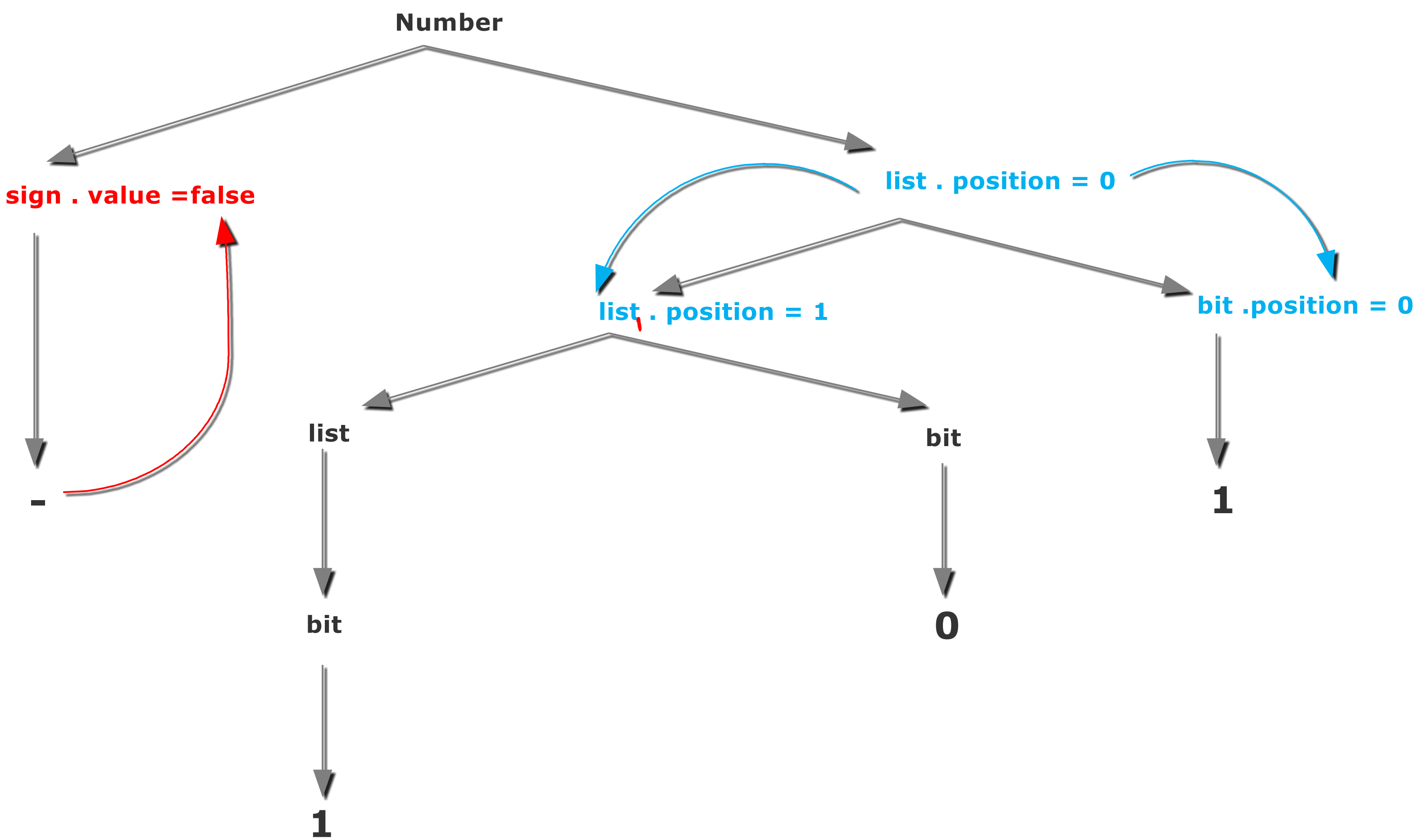
# Parse Tree and Dependence Graph



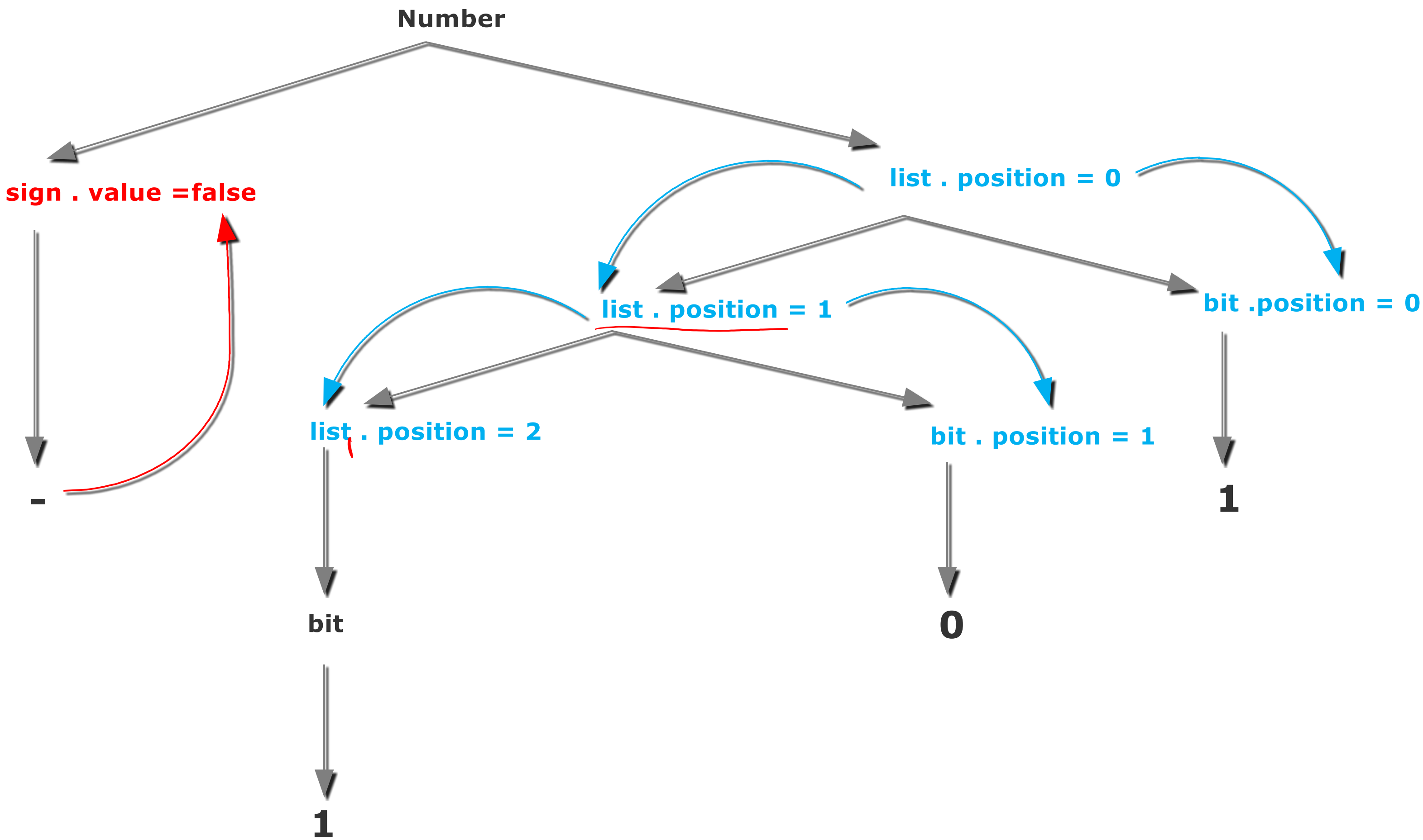
# Parse Tree and Dependence Graph



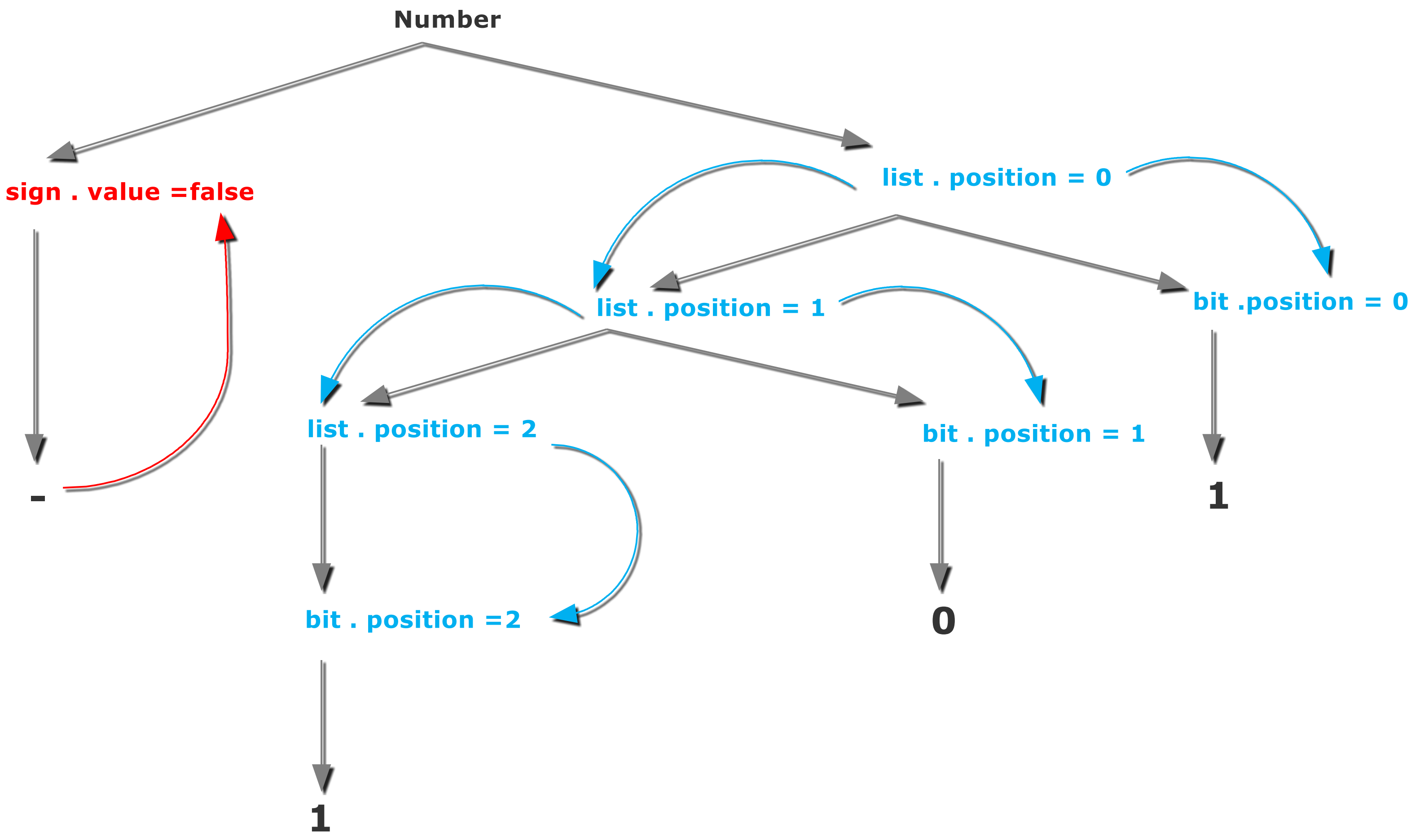
# Parse Tree and Dependence Graph



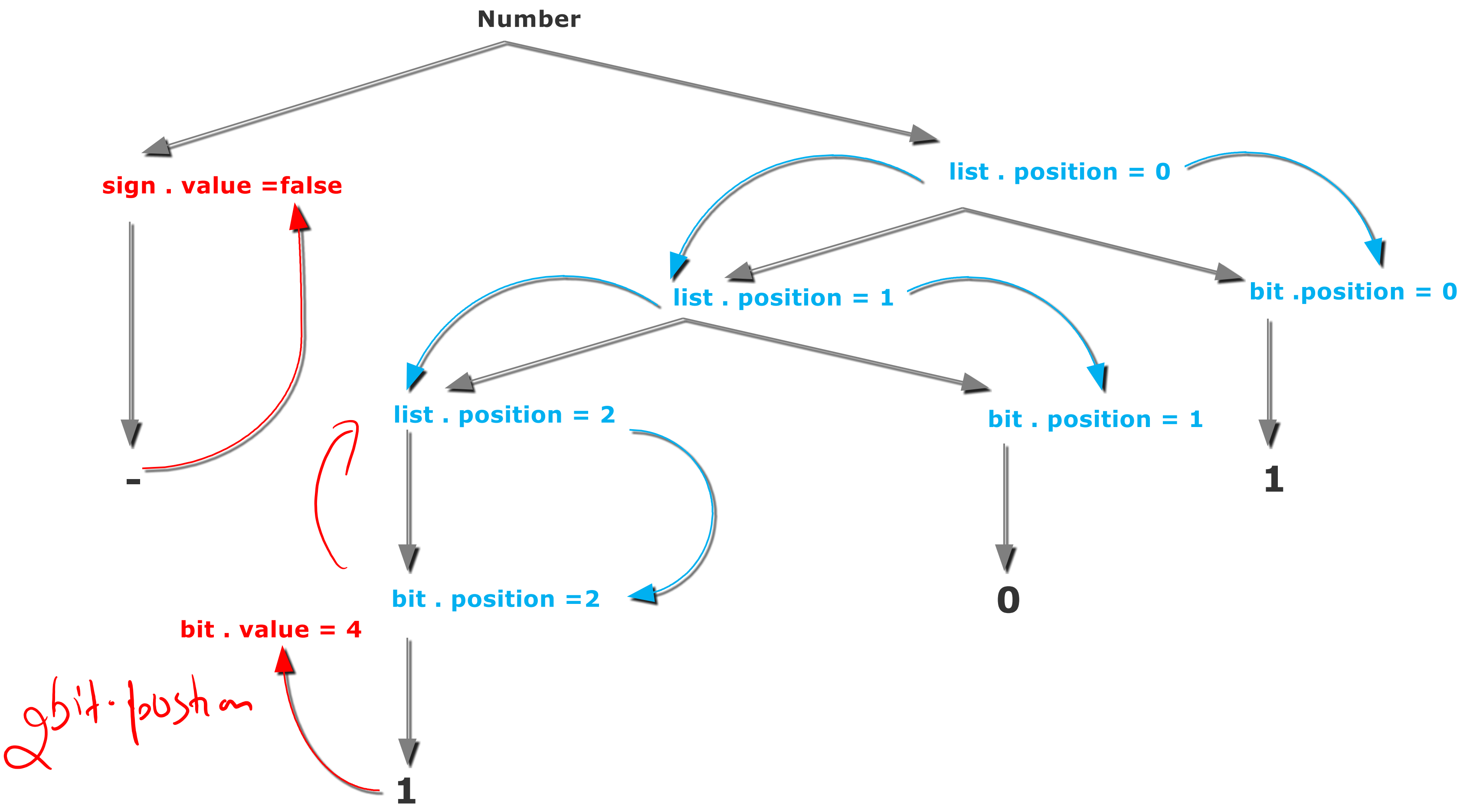
# Parse Tree and Dependence Graph



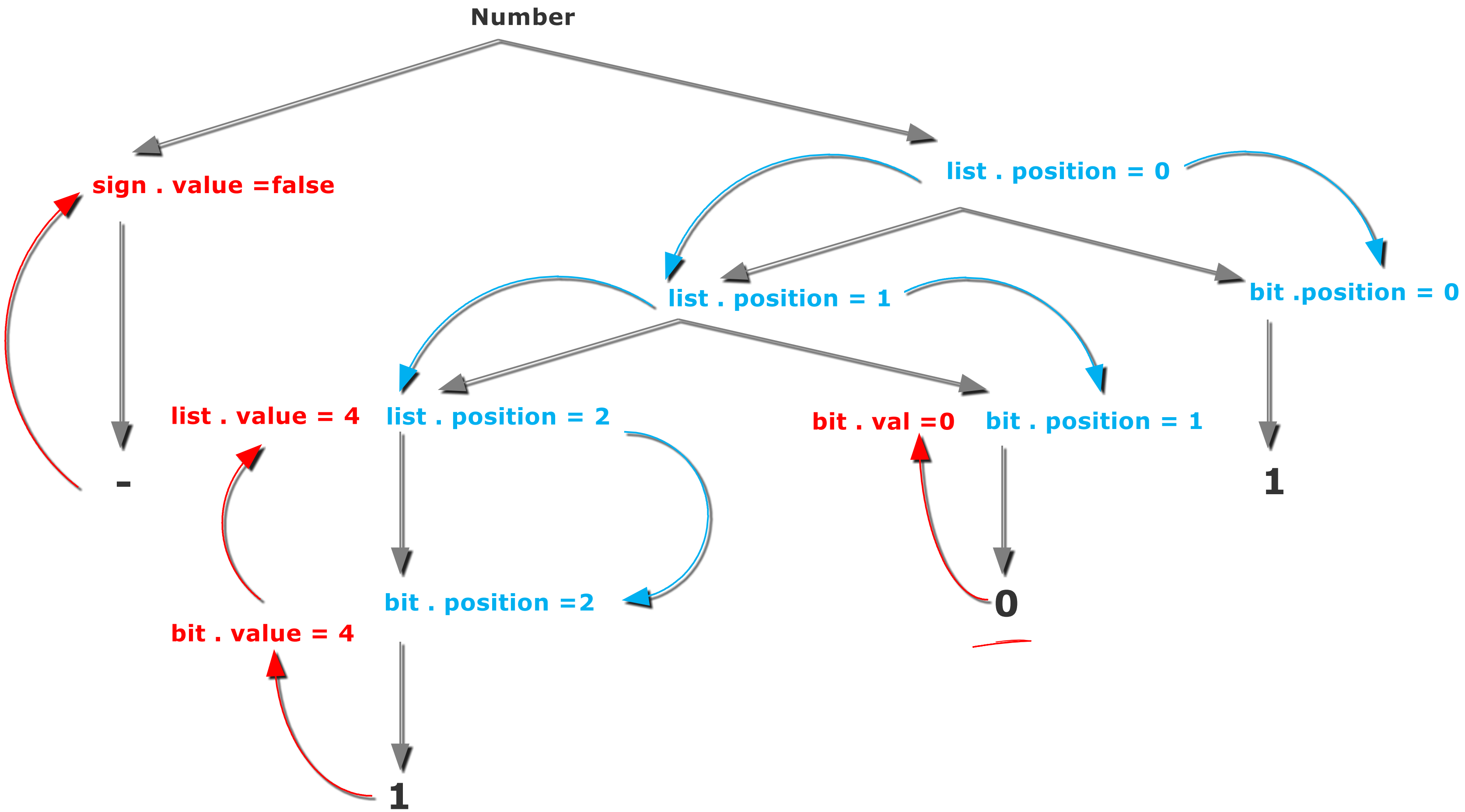
# Parse Tree and Dependence Graph



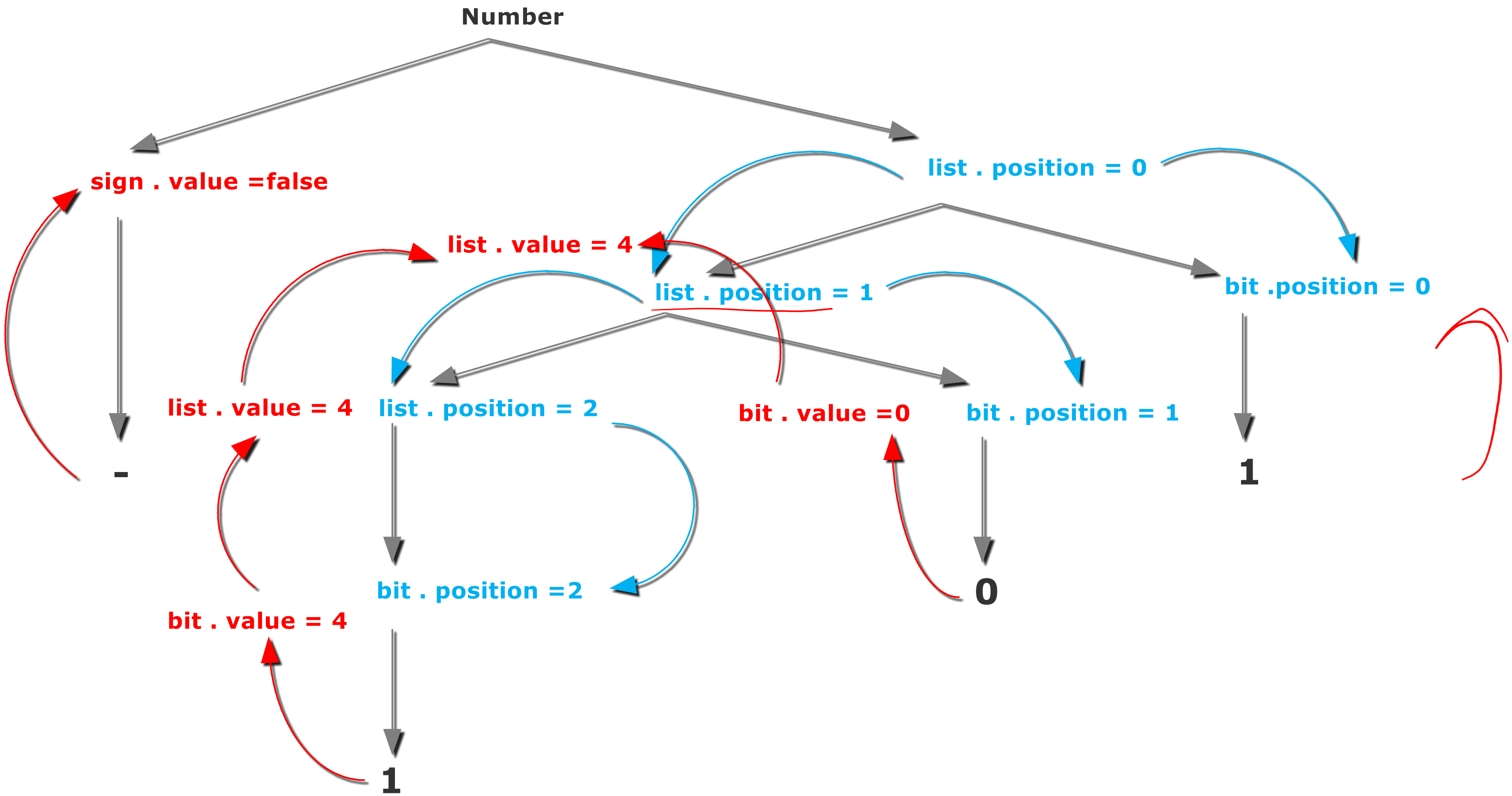
# Parse Tree and Dependence Graph



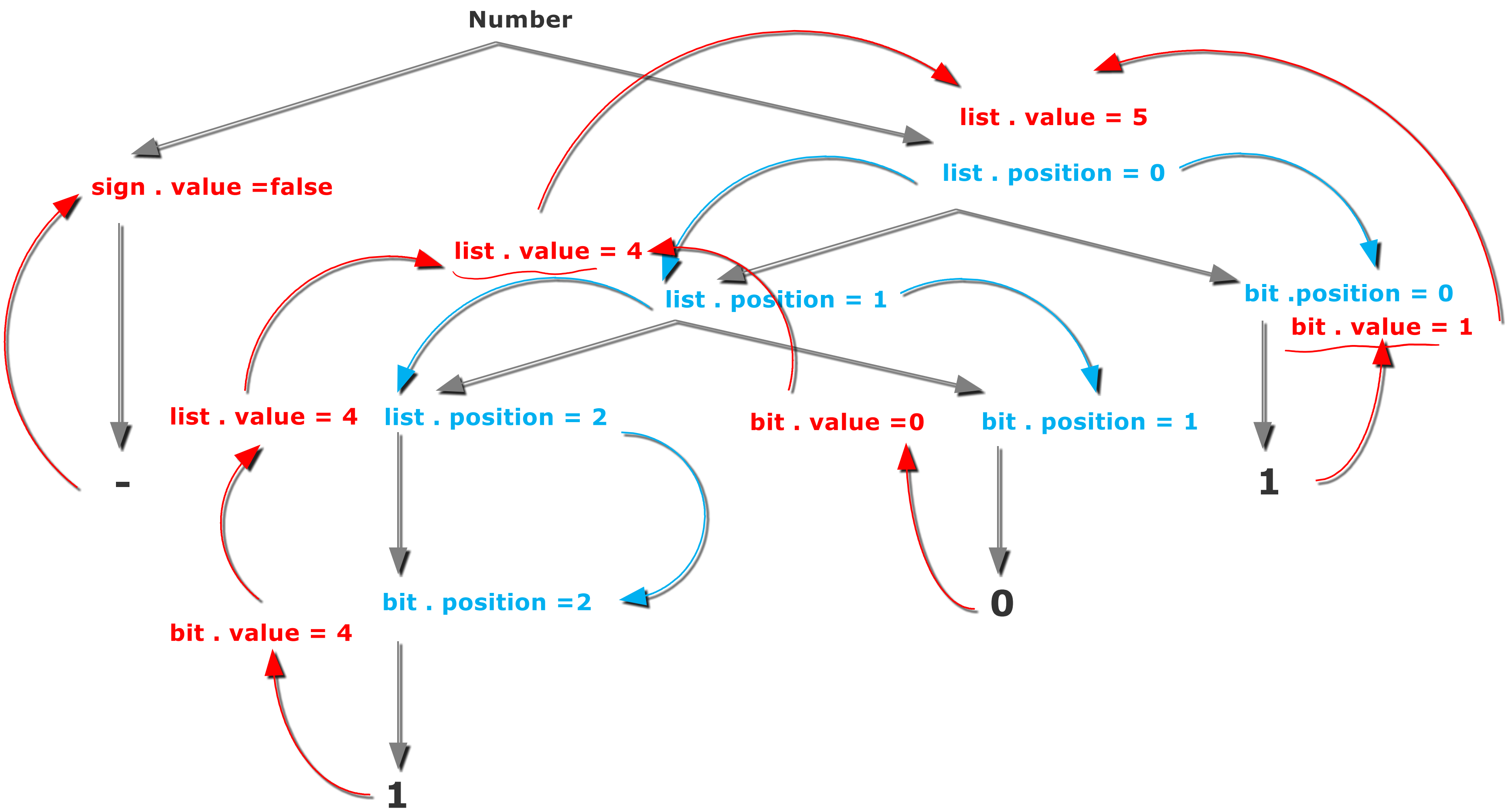
# Parse Tree and Dependence Graph



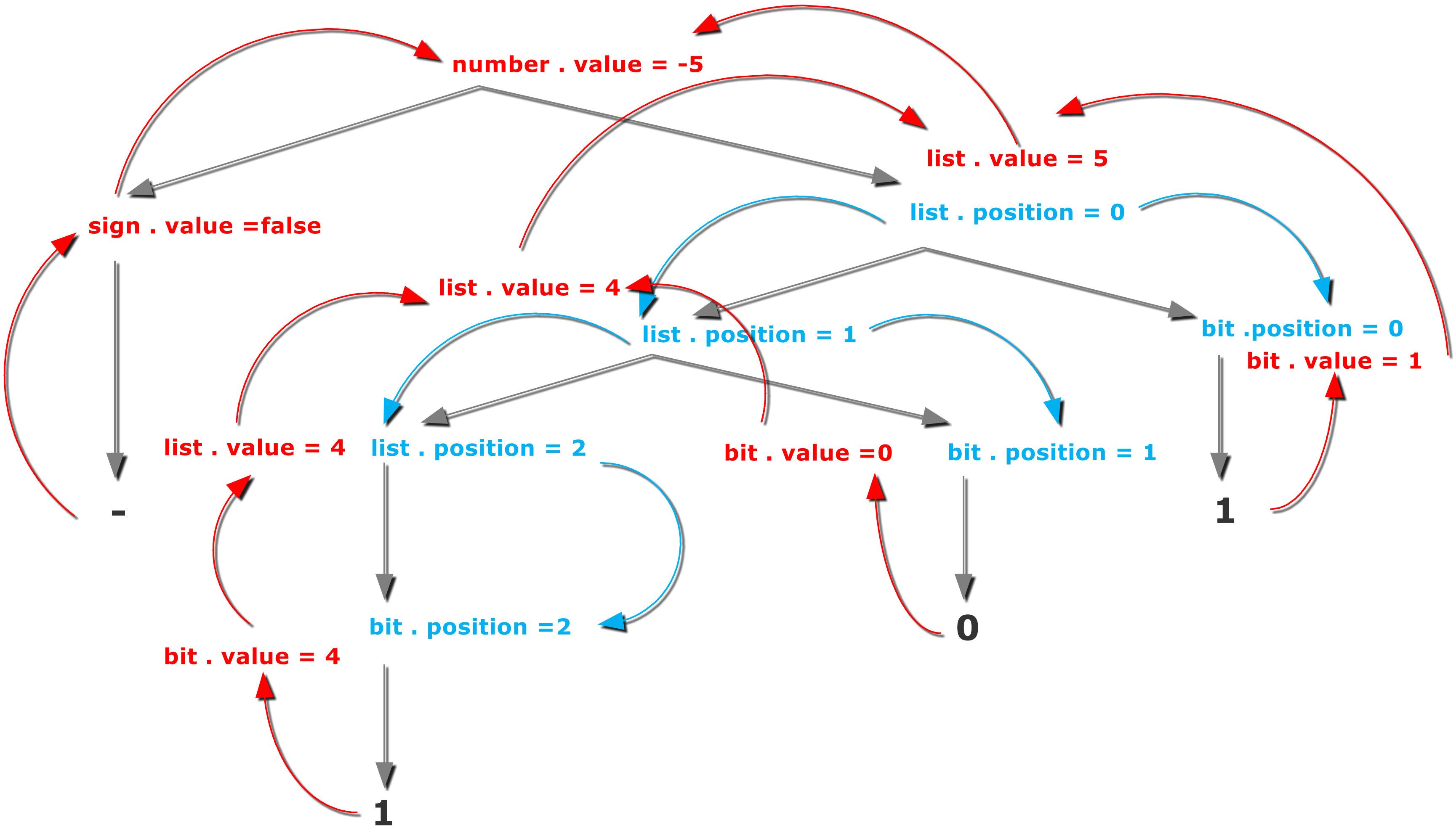
# Parse Tree and Dependence Graph



# Parse Tree and Dependence Graph



# Parse Tree and Dependence Graph



# SDD

**Production Rules**

number -> sign list

**Semantic Rules**

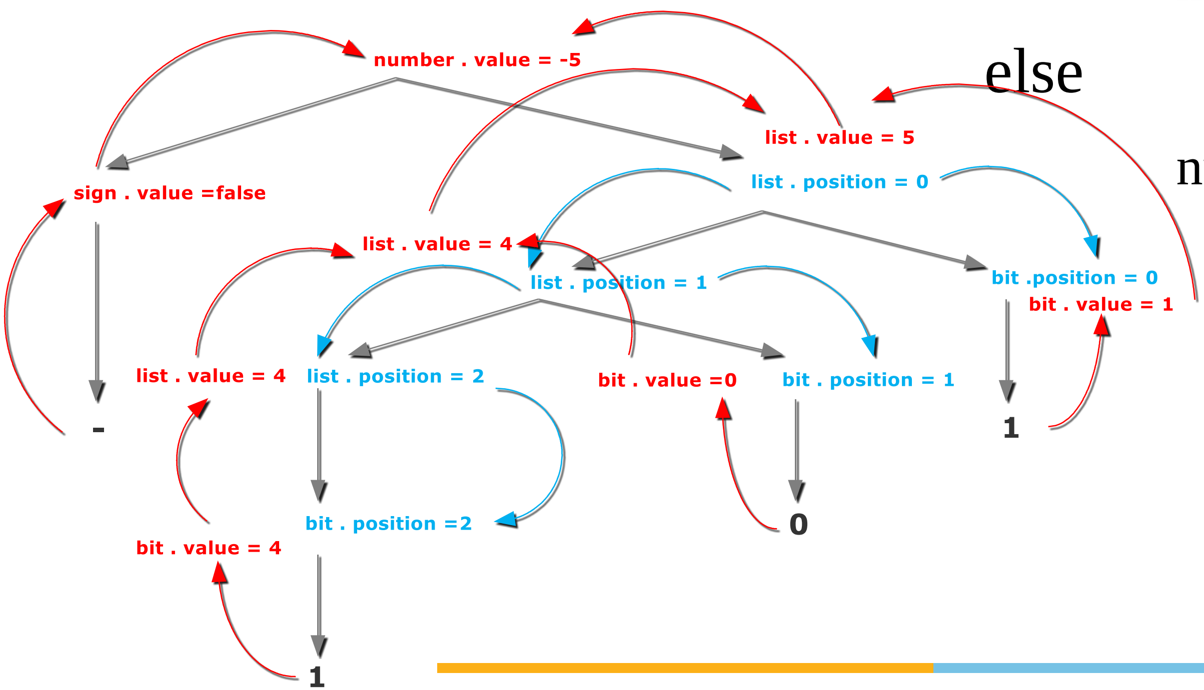
list.position = 0

if (sign.value)

number.value = list.value

else

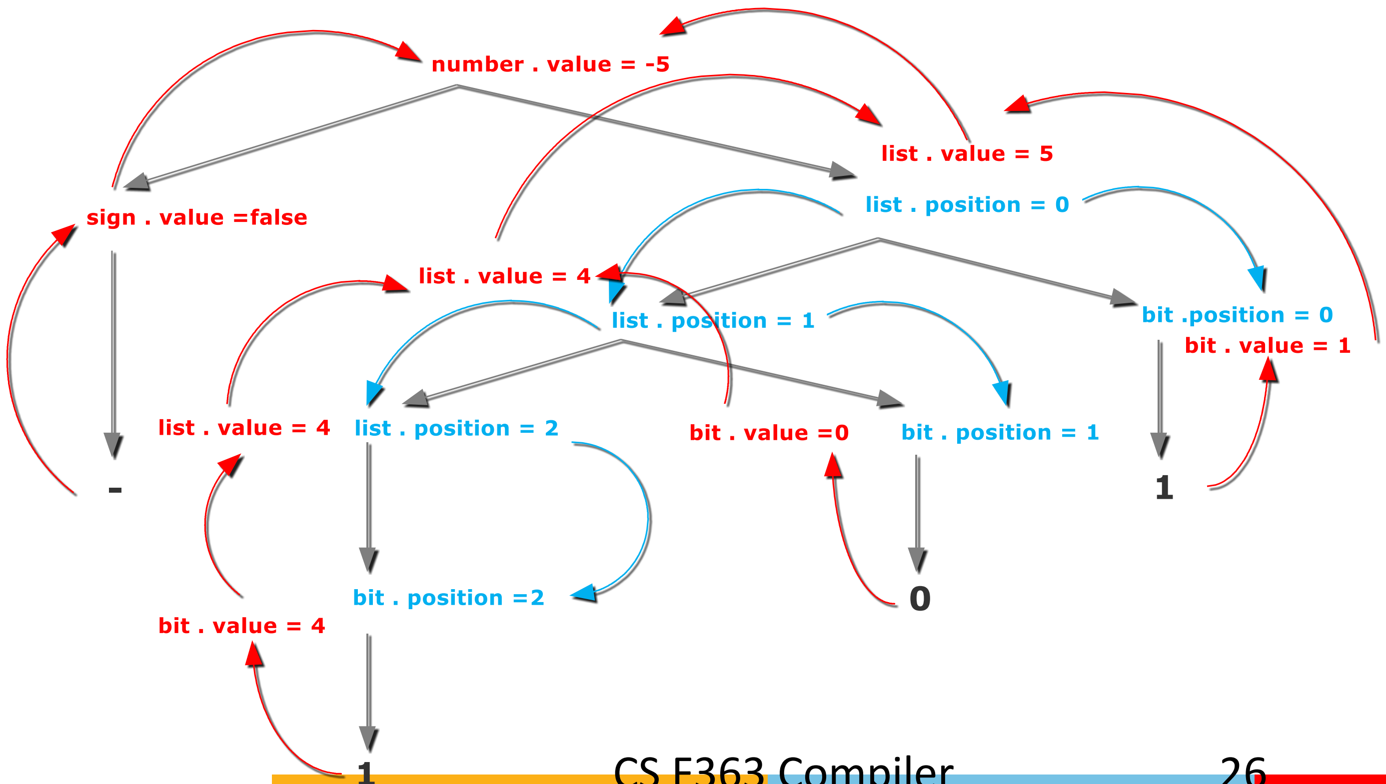
number.value = -list.value



# Example

**Production Rules** | **Semantic Rules**

| **Production Rules** | **Semantic Rules** |
| --- | --- |
| sign -> + | sign.value = true |



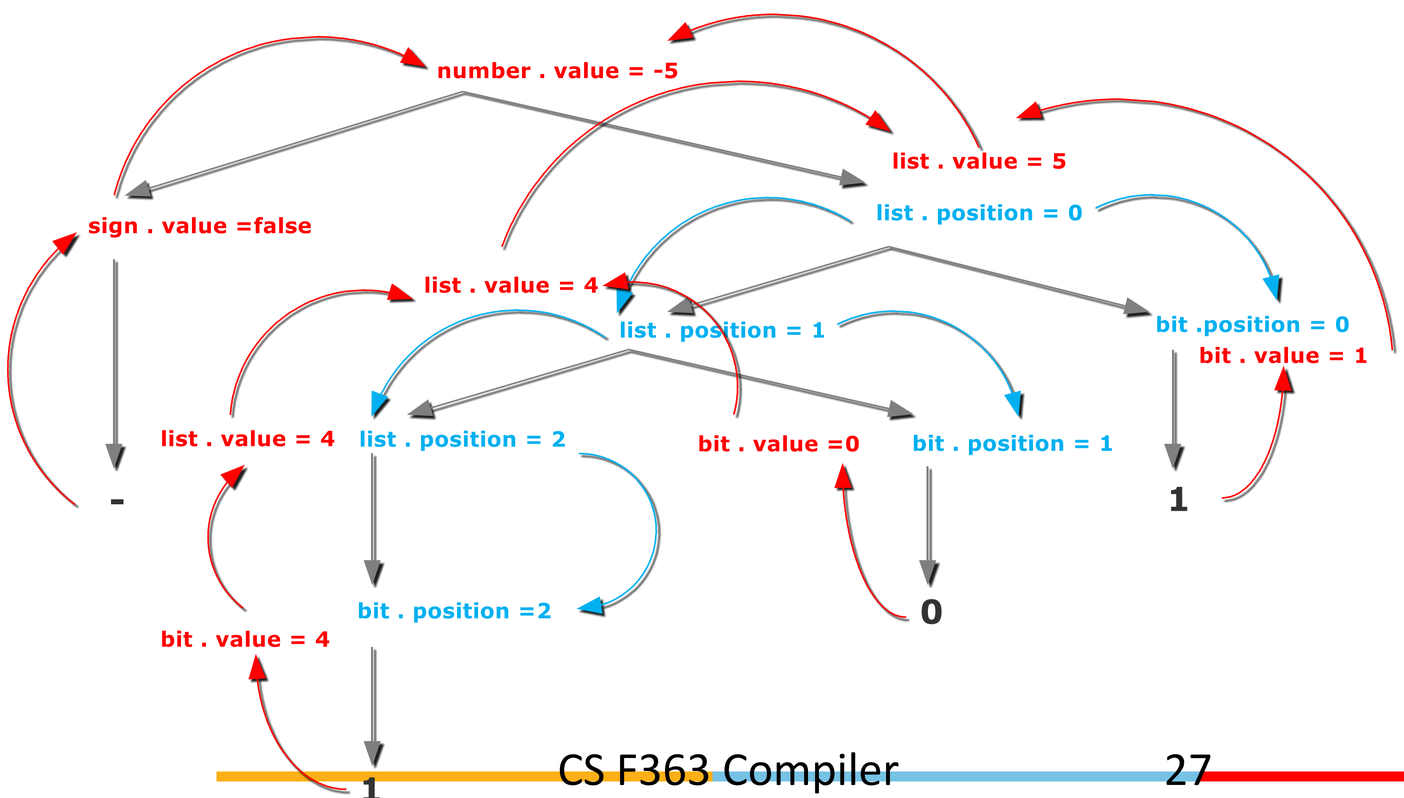
# Example

**Production Rules**

sign -> -

**Semantic Rules**

sign.value = false



# Example

**Production Rules**

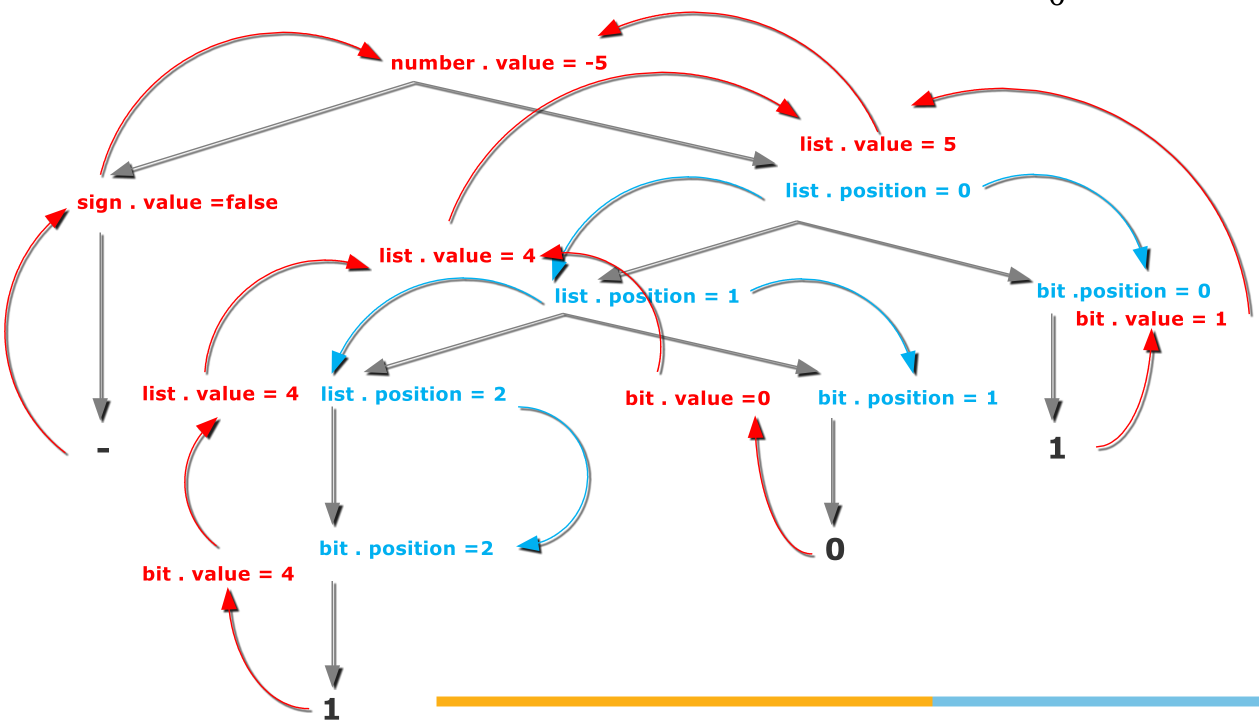
$list_0 \text{ -> } list_1 \text{ bit}$

**Semantic Rules**

$list_1.position = list_0.position + 1$

$bit.position = list_0.position$

$list_0.value = list_1.value + bit.value$



# Example

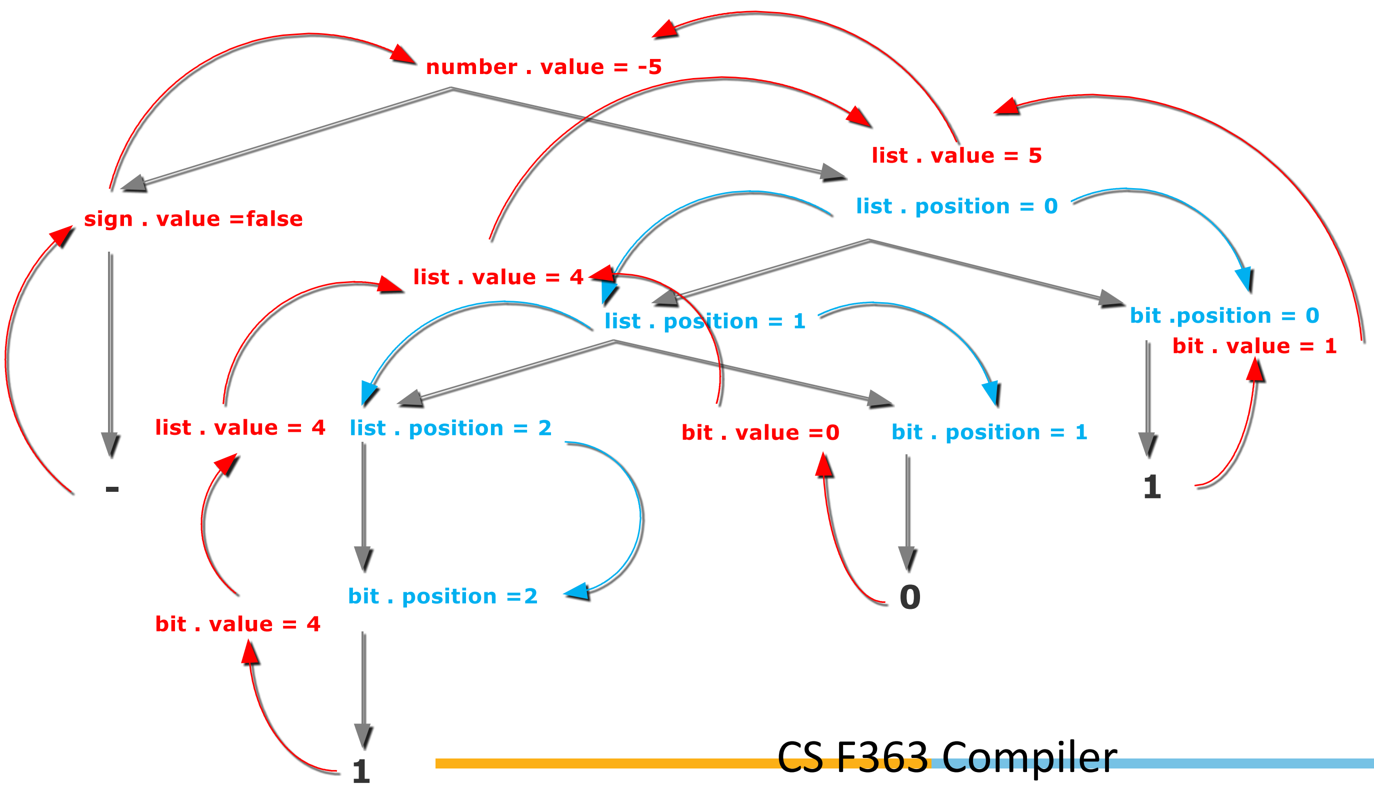
**Production Rules**

list -> bit

**Semantic Rules**

bit.position = list.position

list.value = bit.value



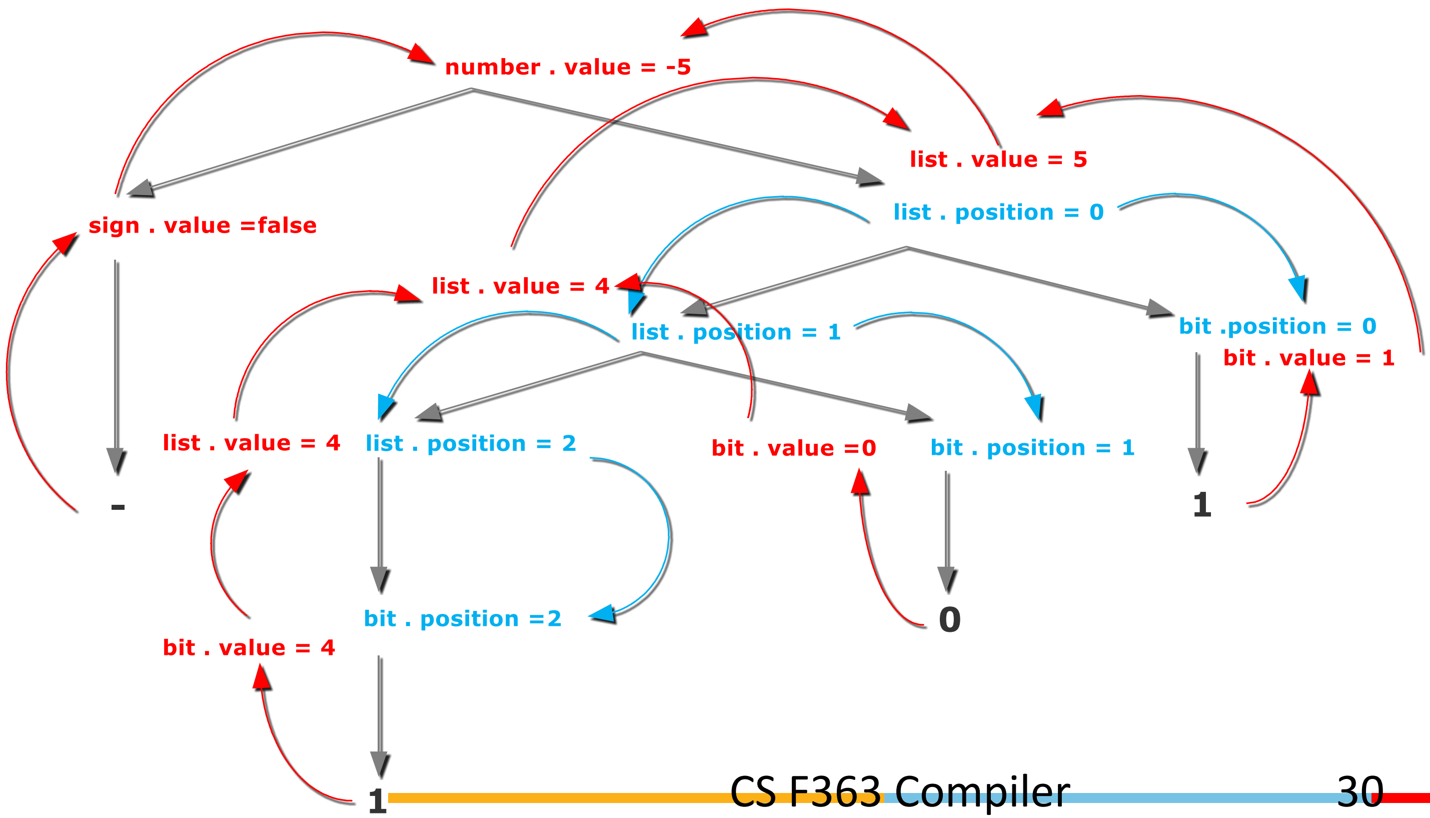
# Example

**Production Rules**

bit -> 0

**Semantic Rules**

bit.value = 0



# Example

## Production Rules

bit -> 1

## Semantic Rules

$\text{bit.value} = 2^{\text{bit.position}}$