### Three languages
- The Hoare axioms are logical rules just like operational rules
- Programs will be annotated with (logical) assertions about program state
- We need languages:
  - For representing programs
  - For representing arithmetic expressions
  - For representing assertions

### Arithmetic expressions
- Expressions are composed of:
  - Variables
  - Values
  - Relational operators
- Their semantics could be given operationally, but we could also be more primitive

### Basic Number Theory
- We could be more “primitive” and use an axiomatization of arithmetic, but this is hard, and largely unnecessary for what we need
- For instance, starting with the number 0 and two operators `succ` and `pred`, we could represent the integers by:
  - 1 is shorthand for `succ(0)`
  - 2 is shorthand for `succ(succ(0))` and so on
  - -1 is shorthand for `pred(0)`
  - -2 is shorthand for `pred(pred(0))`
- Axioms can then define higher operators like + and *
- Eventually we could build the whole of arithmetic
- We choose not to do this (see proof later for an example)
**Abstract syntax of expressions**

- $E_i ::= n \mid \text{id} \mid o_a(E_i) \mid (E'_1 \circ_o E_i)$
- $o_a ::= - \mid \text{abs}$
- $o_b ::= + \mid - \mid * \mid ** \mid / \mid \text{mod}$
- $n ::= \ldots -2 \mid -1 \mid 0 \mid 1 \mid 2 \ldots$
- $\text{id} ::= x \mid y \mid z \mid \ldots$

- Expressive enough for most programming languages e.g. $((x-1)\ast2+z)/(u+v)$
- Note that these variables are not programming language variables

**Substitution**

- We will need a way to represent the changes in program state
- Since arithmetic expressions are just symbolic, we use substitution to change them
- Notation: $E[E'\text{id}]$ (substitute $E'$ for $\text{id}$ in $E$)
- E.g. $(x+y^3)((z-1)y)$ is $(x+(z-1)^3)$
- Also $E[E'\setminus E'']$ for substituting whole expressions
- E.g. $((x^2+1)/(x^2))[(y-1)/(x^2)]$ is $((y-1)+1/(y-1))$
- Why this is necessary will be seen in the assignment axiom.

**Assertions**

- To the arithmetic expression language we add relational operators and logic
  
  - $A ::= (E_i \ r E_i') \mid \neg A \mid (A \land A') \mid \forall X.A \mid \exists X.A$
  
  - $r ::= = \mid \neq \mid > \mid < \mid \leq$
  
  - $\text{cb} ::= \land \mid \lor \mid \Rightarrow$

- Now we can express constraints like $x>0 \land y>1$
- The quantifier variables (upper case $X$, $Y$, $Z$, ...) are logical variables; now we have three kinds of variables: program, arithmetic, logical

**Natural deduction**

- The annotated program will produce a proof using natural deduction techniques
- Sometimes there is “gluing” to do between commands
- Have to prove $A_{n-1} \Rightarrow A_n$
- This can be hard
Example “simple” proof from axiomatization of arithmetic

- To prove: ∀X.0+X=X
- Given axioms:
  - ∀X.X+0=X (we do not have ∀X,Y.X+Y=Y+X)
  - ∀X,Y.X+(Y+1) = (X+Y)+1 (associativity of +)
  - Substitution axioms of identity – can always substitute equals for equals, and transitivity of equals

Proof

- Induction: from a base expression B, prove B(0), then prove ∀X.B(X)⇒B(X+1)
- Choose expression from goal statement: B(x) is 1. 0+x=x
- Use transitive axiom:
  1. x+(y+1)=(x+y)+1 (remove quantifiers – x and y are arbitrary)
  2. 0+(x+1)=(0+x)+1
  3. (0+x)+1=(0+x)+1 (identity)
  4. (0+x)+1=x+1 (from B(x))
- 5. 0+(x+1)=x+1 (from 5 and 3)
- 6. B(x+1) ⇒ B(x+1) (conditional proof)
- 7. ∀X.B(X)⇒B(X+1) (since x is not special)
- QED

Justifications

- We will not prove every step, especially the “glue”
- Instead we will rely on algebra and other simple reasoning methods
- e.g. if we are trying to prove: \( x>y \land y=0 \Rightarrow x>0 \)
  - we can substitute 0 for y in \( x>y \) and we get the right hand side, so it is true, but not really proved
- Other techniques include mechanical proof checking and even “creative” theorem proving programs