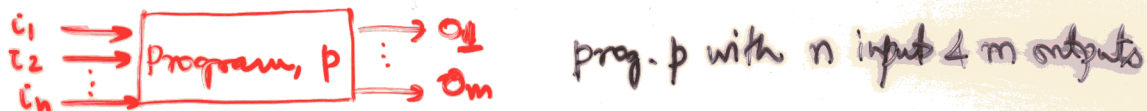


Specifying Program for Correctness

- There are two notions of correctness of a program
 - syntactical: program has correct syntax so it will compile and run
 - semantical: program correctly implements the intended specification

Example: "Tree can move" is syntactically correct but not semantically correct.

- Semantic correctness of program requires
 - termination of program, and
 - correctness of function implemented
- (ii) alone called **partial correctness**; (i) & (ii) together called **total correctness**
- Termination of a program on a given input is in general undecidable. (popularly known as halting problem), and so is proving functional correctness. (Meaning both require human reasoning).
- We discuss how to specify (and verify) for functional correctness. 1st order logic can be used to specify functional correctness



- **Precondition**, $\text{pre}(i_1, \dots, i_n)$: 1st order formula specifying property of inputs prior to P 's execution
 - **Postcondition**, $\text{post}(i_1, \dots, i_n, o_1, \dots, o_m)$: 1st order formula specifying property of inputs & outputs after P 's execution
- $\text{pre}(\vec{i}) \{ P \} \text{post}(\vec{i}, \vec{o})$ ← known as "Hoare triple"

- Function correctness specified by pre & post conditions read as:
If $\text{pre}(\vec{i})$ true before execution, then $\text{post}(\vec{i}, \vec{o})$ after execution.