# Functional Languages

## Functional Languages (Applicative, value-oriented)

Importance?

- In their pure form they dispense with notion of assignment
  - claim is: it's easier to program in them
  - also: easier to reason about programs written in them
- FPL's encourage thinking at higher levels of abstraction
  - support modifying and combining existing programs
  - thus, FPL's encourage programmers to work in units larger than statements of conventional languages: "programming in the large"
- FPL's provide a paradigm for parallel computing
  - absence of assignment (single assignment) } provide basis
  - independence of evaluation order } for parallel
  - ability to operate on entire data structures } functional programming

## Importance of Functional Languages…

- Valuable in developing executable specifications and prototype implementations
  - Simple underlying semantics
    - rigorous mathematical foundations
    - ability to operate on entire data structures
      => ideal vehicle for capturing specifications
- Utility to AI
  - Most AI done in func langs (extensibility. symbolic manipulation)
- Functional Programming is tied to CS theory
  - provides framework for viewing decidability questions
    - (both programming and computers)
  - Good introduction to Denotational Semantics
    - functional in form

## Expressions

- Key purpose of functional programming:
  - to extend the advantages of expressions (over statements) to an entire programming language
- Backus ('78 FP paper) has said that expressions and statements come from two different worlds.
  - expressions: $(a + b) * c$ arithmetic
    $(a + b) = 0$ relational
    $\neg(a \vee b)$ boolean
  - statements: the usual assortment with assignment singled out
  - *assignments alter the state of a computation* (ordering is important)
    e.g. a:= a * i; i:= i + 1
- In contrast, ordering of expressions is not side-effecting and therefore not order dependent (Church-Rosser property /Church Diamond)

# More Expressions

- With Church-Rosser
  - reasoning about expressions is easier
  - order independence supports fine-grained parallelism
  - Diamond property is quite useful
- Referential transparency
  - In a fixed context, the replacement of a subexpression by its value is completely independent of the surrounding expression
    - having once evaluated an expression in a given context, shouldn't have to do it again.

    Alternative: referential transparency is the universal ability to substitute equals for equals (useful in common subexpression optimizations and mathematical reasoning)

# Hoare's Principles of Structuring

(1973, "Hints on Programming Language Design," Stanford Tech Rep)

1) Transparency of meaning
   - Meaning of whole expression can be understood in terms of meanings of its subexpressions.

2) Transparency of Purpose
   - Purpose of each part consists solely of its contribution to the purpose of the whole. ➡ No side effects.

3) Independence of Parts
   - Meaning of independent parts can be understood completely independently.
     - In E + F, E can be understood independently of F.