CS 330 Lecture 3

- Outline
  - Quick intro to symbol tables
  - Intro to functional programming - why SML ?
  - Assignment I

Schedule link in webpage: Suggested readings and exercises

Names and attributes

- const int n = 5;
  - name n
  - type attribute = const int
  - value attribute = 5
- double f(int n) { .....}
  - name f
  - type attribute = function of one integer argument that returns a double
  - body attribute = the actual code

Binding

- Associate attribute with a name
- Static binding
  - Translation
  - Linking
  - Loading
- Dynamic binding

Symbol Table or Environment

- Function that expresses the bindings of attributes to names
- Compilers – Symbol Table
- Interpreters - Environment
- Variable dictionary
  - Insert, Lookup, Delete
- Scope
  - region of program where binding is maintained

Functional Languages

- Black box view
- Function y=f(x) f: X->Y
  - domain X, range Y
  - x = independent variable, y = dependent variable
  - partial vs total function
- Function definition, application

Functional Languages

- Scheme, ML, Haskell
- AI, prototyping, proof-systems
- Advantages
  - Uniform view of programs as functions
  - Automatic memory management
  - Great flexibility, conciseness of notation and simple semantics
- Drawback (used to be)
  - Inefficiency

Then why people don't use them

- Persistence of established technology
- More abstract and mathematical
- Object-oriented programming mirrors everyday experience so for simple programs it is easier (that's why it doesn't work very well :-))
- Less libraries although they are catching up

Side-effects – the enemy

- In pure functional languages there are no assignments only bindings.
- Referential transparency
  - Function that its value depends only on the values of its arguments
- Value semantics
- Functions are first class citizens

Why SML I ?

- Very powerful in expressing structured data
- Simple interactive interface
- Higher-order functions
- Polymorphism
- Side-effect freedom
- Strong typing
- Abstract data types

Why SML II ?

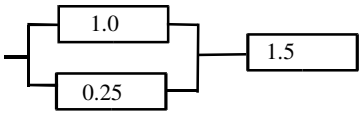
- Recursion
- Rule-based programming
- Complete, formal semantics
- Combination of the “best IMHO” features of
  - Lisp
  - Pascal, Modula-2
  - Prolog
  - C++, Smalltalk

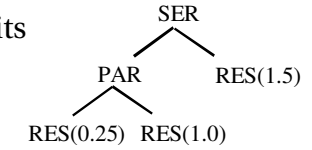
FP in the real world

- Phone calls in the European Parliament
  - Erlang (Ericsson's functional language)
- CDs shipped by Polygram in Europe using Software AG's Natural Expert
- HOL theorem prover
  - helped design the HP 9000 line of multiprocessors

Assignment I

- Binary resistance circuits





Prefix input notation: - | 1.0 0.25 1.5

| is used to denote parallel connection
- is used to denote series connection

Resistance of combinations

$r(c_1 - c_2) = r(c_1) + r(c_2)$

$r(c_1 \mid c_2) = 1 / (1/r(c_1) + 1/r(c_2))$