CS424 - Programming Languages and Semantics

Lecture 24 18th December 2014

**Prolog:**

*Difference Lists:*

* Data structure that comes naturally in Prolog, but clunky to implement in other languages.
  + Only convenient to implement if you’ve logic variables available.

List of elements: a b c

*Usual Method:*

*Difference List:*

Idea: [a,b,c]=[a,b,c,d,e,f] – [d,e,f]

*Difference Lists in Prolog:*

dl-eq( L , X-Y )

* L= Standard List, X-Y=Difference List.
* Returns true if they match.

dl-eq( [] , X-X ) for any list X

dl-eq( [A|L] , [A|M]-N ) :- dl-eq( L , M-N )

dl-append( Xhead – Xtail , Yhead –Ytail , Zhead – Ztail )

* X appended to Y gives us Z.

Idea: Zhead == Xhead

Yhead == Xtail

Ztail == Ytail

*Usage of Difference Lists:*

* Used a lot in Parsing.

**Semantics:**

*Axiomantic Semantics:*

* Hoare Logic/Semantics: As seen in CS357 – Software Verification.

*Operational Semantics:*

* Interpreters (e.g. an interpreter written in Scheme for Scheme).
* Define the semantics of your programming language by writing an interpreter.
* Issue/Problem: Not great for writing compilers, as these require a greater level of precision.

*Denotational Semantics:*

* Relates/links to compilers.
* Mathematical description given a piece of code, what does it mean.
* Properties:
  + Compositional etc.
* Prove properties of programming languages.
* Prove axioms from axiom semantics.
* Prove properties of implementations.
* Big Issue: Defining what you mean by recursion can be difficult in math.
  + Scott Domains
  + Defined and smallest #inputs and still match definition.