Announcements

- Last Class! Exam next Friday!

"Consequently he who wishes to attain to human perfection, must therefore first study Logic, next the various branches of Mathematics in their proper order, then Physics, and lastly Metaphysics."

Maimonides 1135–1204

"Logic is the beginning of wisdom, not the end."

Leonard Nimoy
"Star Trek VI: The Undiscovered Country" 1991

Since Last midterm

- difference lists, definite clause grammars, and natural language interfaces to databases
- computer algebra and calculus
- Triples, knowledge graphs, URIs/IRIs, Ontologies
- You should know what the following mean: RDF, IRI, rdf:type, rdfs:subClassOf, rdfs:subPropertyOf, rdfs:domain, rdfs:range
- Complete knowledge assumption and negation as failure
- Extra-logical predicates
- Substitutions and Unification
- Proofs and answers. Negation with variables.

Today:

- Haskell in Prolog, Prolog in Haskell, Prolog in Prolog

3 implementations of not-equals

- Prolog has 3 different inequalities:

  `\==` `\=` `dif()`

  which give same answers for variable-free queries, or when both sides are identical

  `a \== 3,` `a \= 3,` `dif(a,3)`

  all succceed.

  `a \== a,` `a \= a,` `dif(a,a)`

  all fail.

- They give different answers when there is a free variable.
  `\==` means "not identical". `a \== X` succeeds
  `\=` means "not unifiable". `a \= X` fails
  `dif` is less procedural and more logical

Implementing dif

- *dif*(X, Y)
  - all instances fail when X and Y are identical
  - all instances succeed when X and Y do not unify
  - otherwise some instance succeed and some fail
- To implement *dif*(X, Y) in the body of a clause:
  - Select leftmost clause — unless it is a *dif* which cannot be determined to fail or succeed (delay dif calls)
  - Return the *dif* calls not resolved.
- Consider the calls:

```
dif(X,4), X=7.
dif(X,4), X=4.
dif(X,4), dif(X,7).
```

Example of dif

```
passed_two_courses(S) :-
    dif(C1,C2),
    passed(S, C1),
    passed(S, C2).
passed(S,C) :-
    grade(S,C,M),
    M >= 50.
grade(sam,engl101,87).
grade(sam,phys191,89).
```

Other predicates, such as #<, work similarly.

```
use_module(library(clpfd)).
    % https://www.swi-prolog.org/man/clpfd.html
X #< Y, Y #< Z, Z #< X.
```

Constraint programming systems provide more sophisticated constraint solving. E.g., https://eclipseclp.org.

A functional language in Prolog (funlog .pl)

```
% fib1(N) returns the N'th Fibonacci number
fib1(0) === 1.
fib1(N) === fib2(N-1,1,1).

% fib2(N,F0,F1) returns the N'th Fibonacci number given
%    the current one is F1 and the previous one was F0.
fib2(0,_,F) === F.
fib2(N,F0,F1) === fib2(N-1,F1,F0+F1) :- N>0.
```

Logic Programming in Haskell

- return list of all solutions. Lazy evaluation gives backtracking.
- `unify :: Term -> Term -> Substitution -> [Substitution]`
  `prove :: Term -> Substitution -> [Substitution]`
  `unify` returns empty list or list containing one element.
- Use Clark's completion:

  ```
  append(X,Y,Z) :- X=[], Y=Z
                 ;  X=[H|T], Z=[H|R], append(T,Y,R).
  ```

  = becomes unify
  ; becomes ++
  , means pass each substition to the next call (like "do" but with a list of answers)

see Curry langauge `https://curry.pages.ps.informatik.uni-kiel.de/curry-lang.org/`

Logic Programming in Prolog (meta.pl)

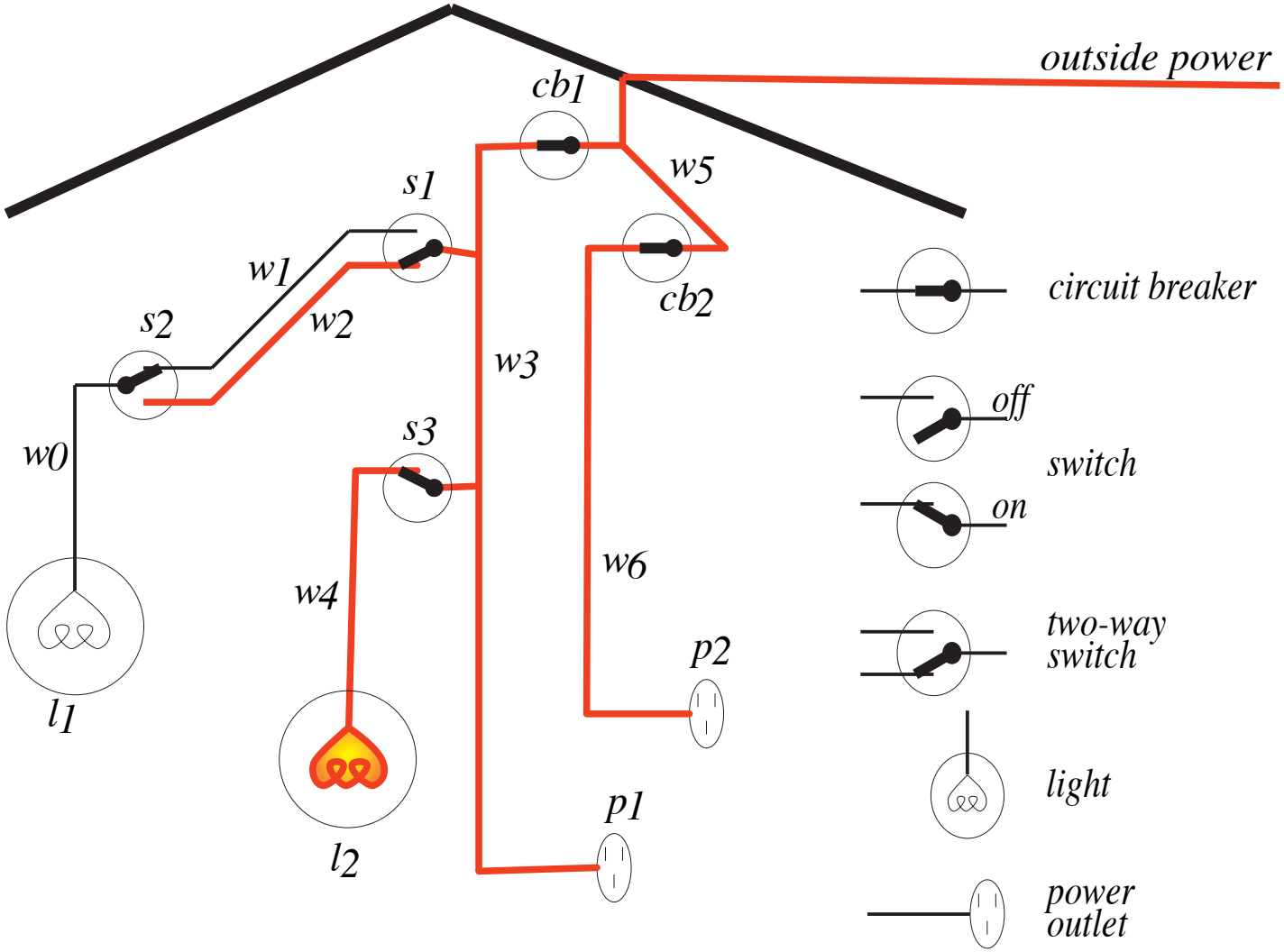
Object level program uses <- and & for "if" and "and".
Meta-level treats <- and & as function symbols. See `meta.pl`.
Variations:

- `bprove.pl` depth-bounded prover
- Delaying prover collects assumptions (abduction).

```
[bprove].
[elect_a].
show(lit(l1)).
```

- Declarative debugging (builds and then traverses a proof tree)

```
[trace2].
[elect_b].
show(lit(l1)).
```


outside power
cb1
w5
s1
w1
w2
cb2
s2
w3
w0
s3
w6
w4
p2
l1
l2
p1
circuit breaker
off
switch
on
two-way
switch
light
power
outlet

Functional and logic programming

Unifying ideas:

- specify what not how
- variables don't change their values once assigned — referential transparency
- each function/clause can be tested and verified modularly.
- high-level specification (e.g., British nationality act, Java VM).

Haskell:

- strong typing
- higher-order functions allow us to create our own abstractions
- lazy computation

Prolog:

- unification allows for powerful pattern matching
- non-determinism through search
- extends relational databases

Declarative programming helps no matter what language you use!