CS131 Week 7

Week 7 Lecture 1

==========

Prolog (finishing up)
Scope(“ “)
Storage management

Last time: \+(x) :- x, !, fail.
\+(_).

negation as failure
“Anything I can’t prove, must be false.”

pr(cs31, cs131).
pr(cs31, cs32).
pr(cs131, cs132).
…

~(dance101, cs131).

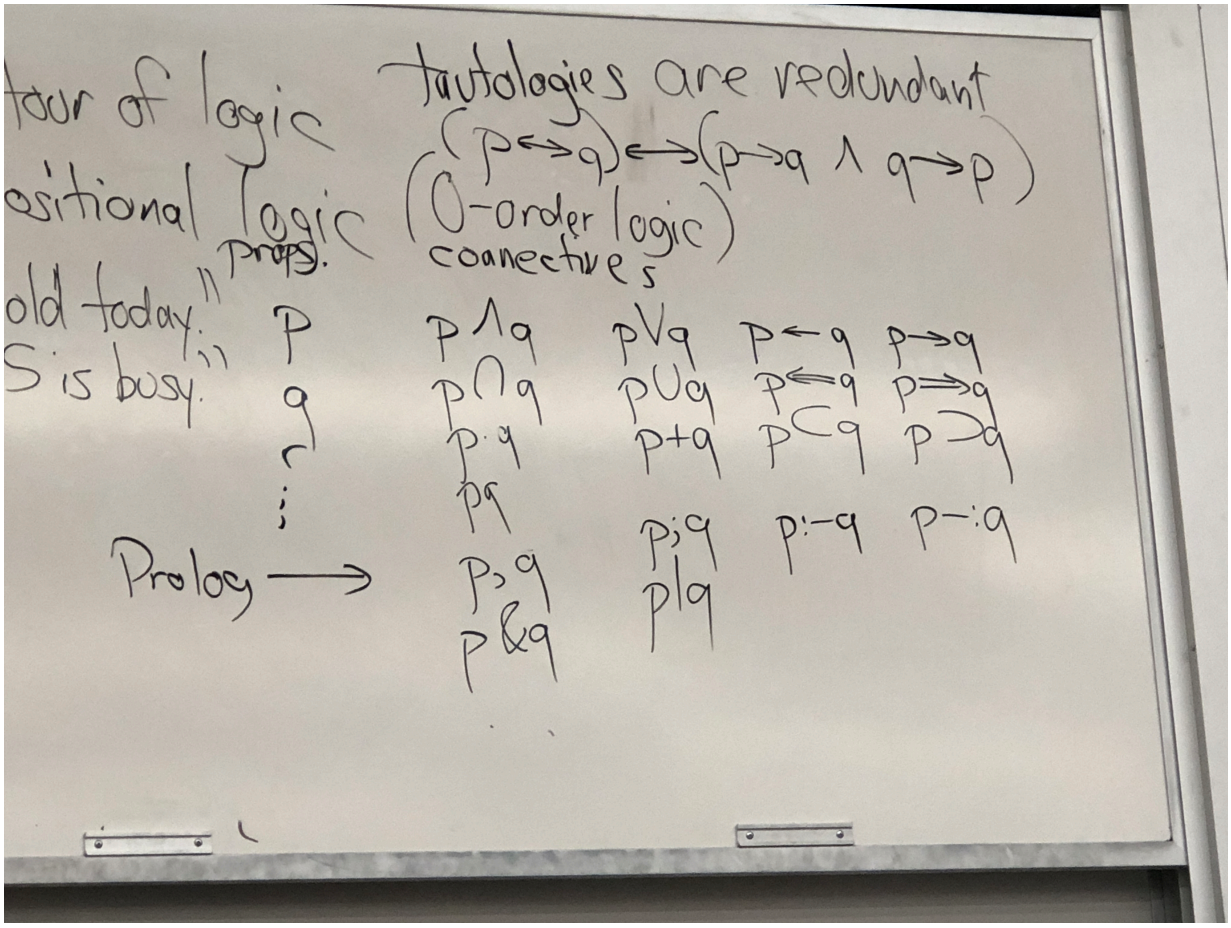
Closed Word Assumption (CWA)

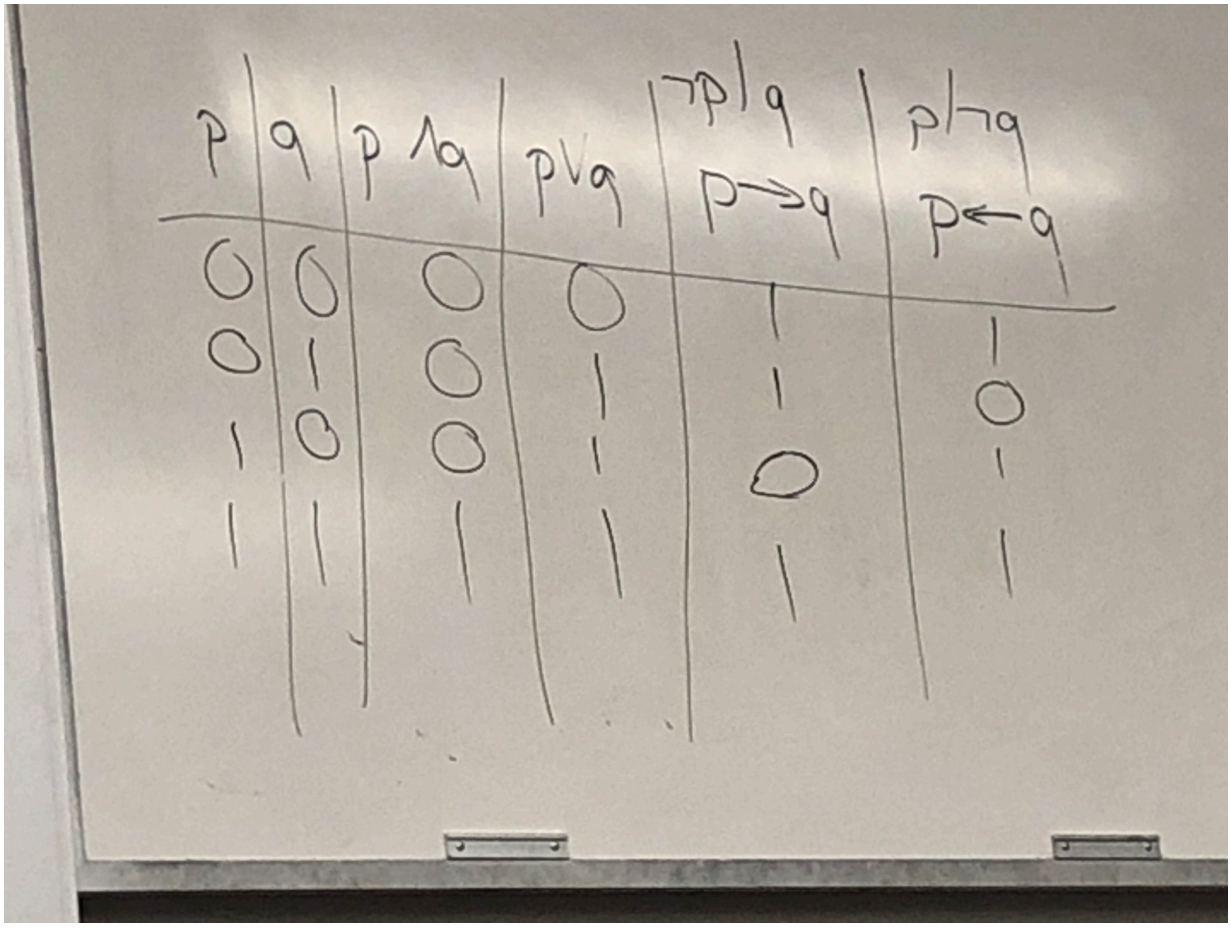
Little tour of logic propositional logic
“It is cold today.” p
“The 405 is busy.” q
… r

tautologies

- redundant
- **(p <—> q) <—> (p —> q ^ q —> p)**

0 order logic


tour of logic
ositional logic
old today."
S is busy."
Prolog
Props.
P
q
r
tautologies are redundant
(P↔q)↔(p→q ∧ q→p)
(0-order logic)
connectives
p∧q
p∩q
p·q
pq
p,q
p&q
pVq
pUq
p+q
p;q
p|q
p←q
p⇐q
p⊂q
p:-q
p→q
p⇒q
p⊃q
p-:q

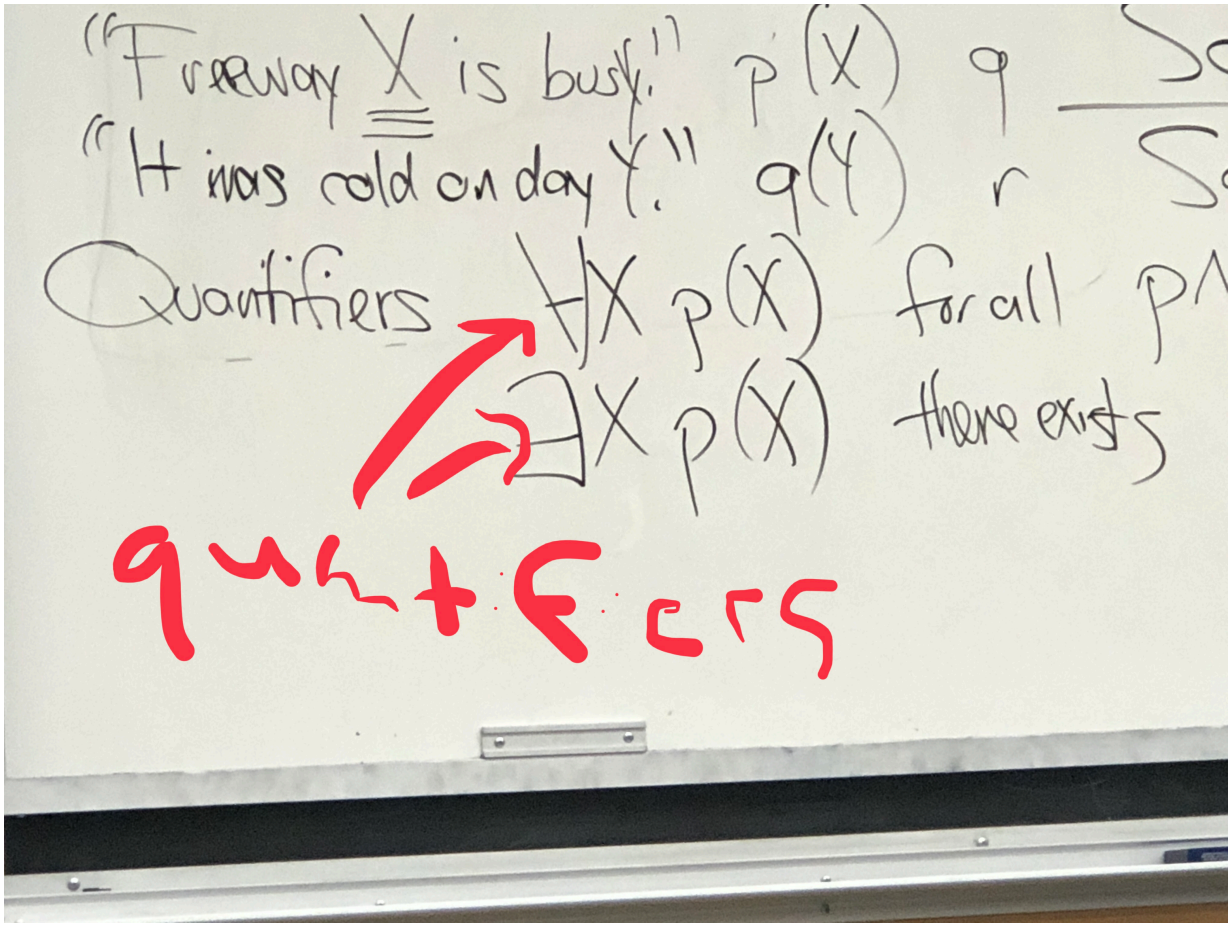


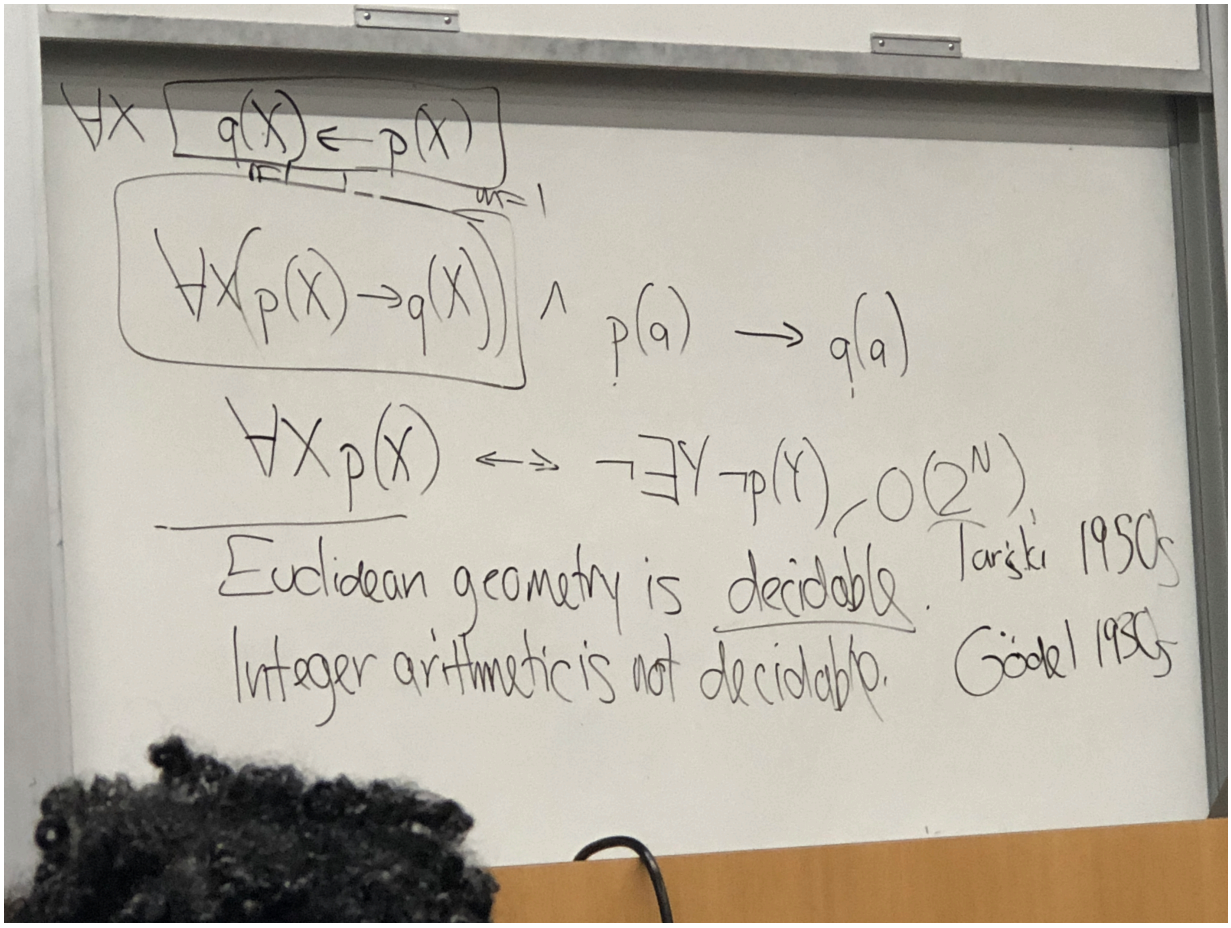
First order logic (predicate calculus)
Propositions have arguments
"Freeway X is busy." p(X) <- notation to show we're using first order logic
"It was cold on day Y." q(Y)

Can't formulate r using p and q in Example below
- not p & q, p | q, etc.

Example:
- All men are mortal. p
- Socrates is a man. q
- Socrates is mortal. r

Quantifiers:

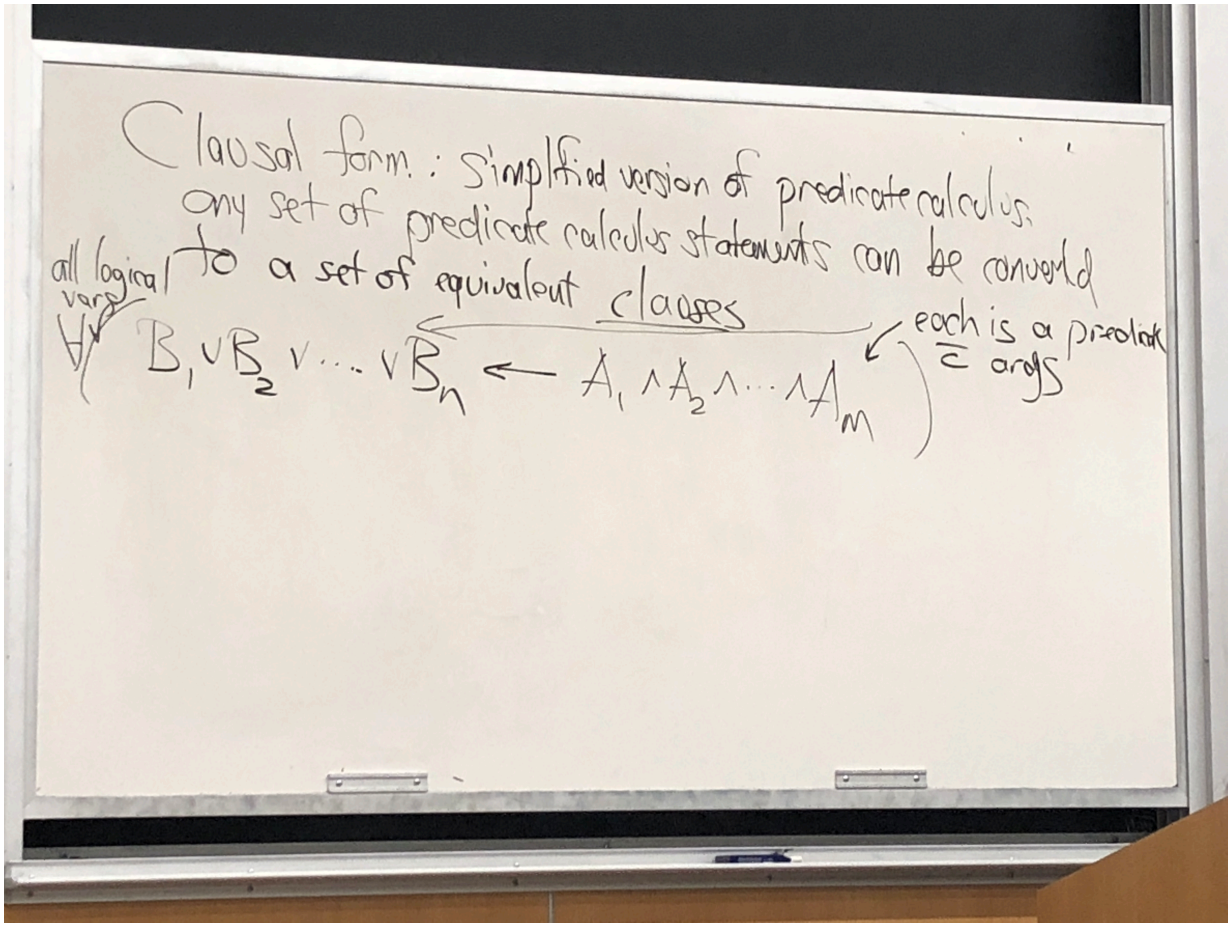

"Freeway X is busy." p(X) q
"It was cold on day Y." q(Y) r
Quantifiers ∀X p(X) for all
∃X p(X) there exists
quantifiers



Can always convert logical form into clausal form

Clausal form:

- simplified version of predicate calculus
- Any set of predicate calculus statements can be converted to a set of equivalent clauses



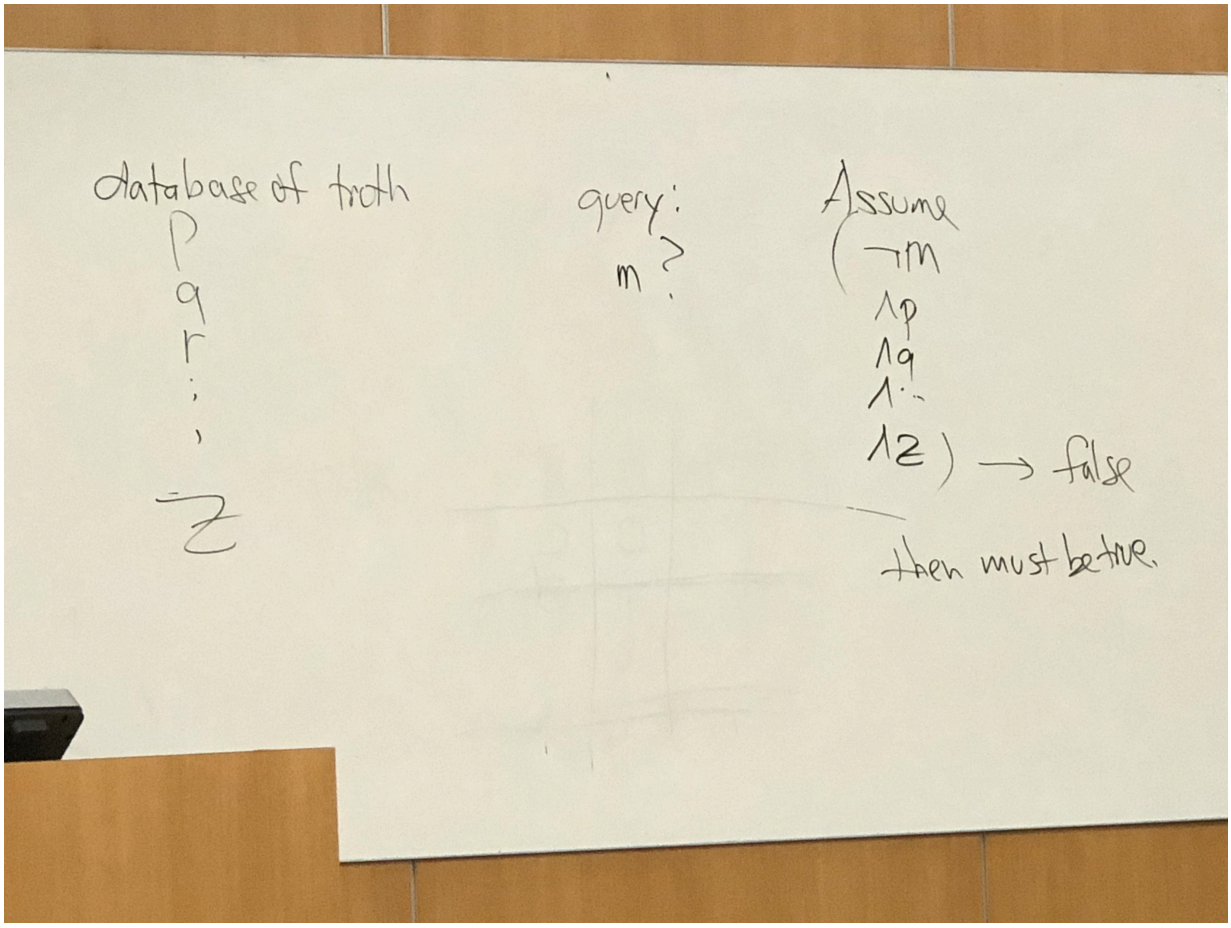
Approximations

Horn: Simplify by assuming n <= 1
Horn clause: (3 possibilities)

- n = 1, m = 0. Prolog fact.
  - man(socrates)
- n = 1, m > 0. Prolog rule.
  - mortal(X) :- man(X)
- n = 0. Prolog query.
  - ?- mortal(socrates).
  - This is backwards: is same as false <- mortal(socrates)
    - Then q = false, p = ?
    - with truth table p—>q, p must be true
    - this means true—>false... BACKWARDS???
    - equivalent to ~mortal(socrates).

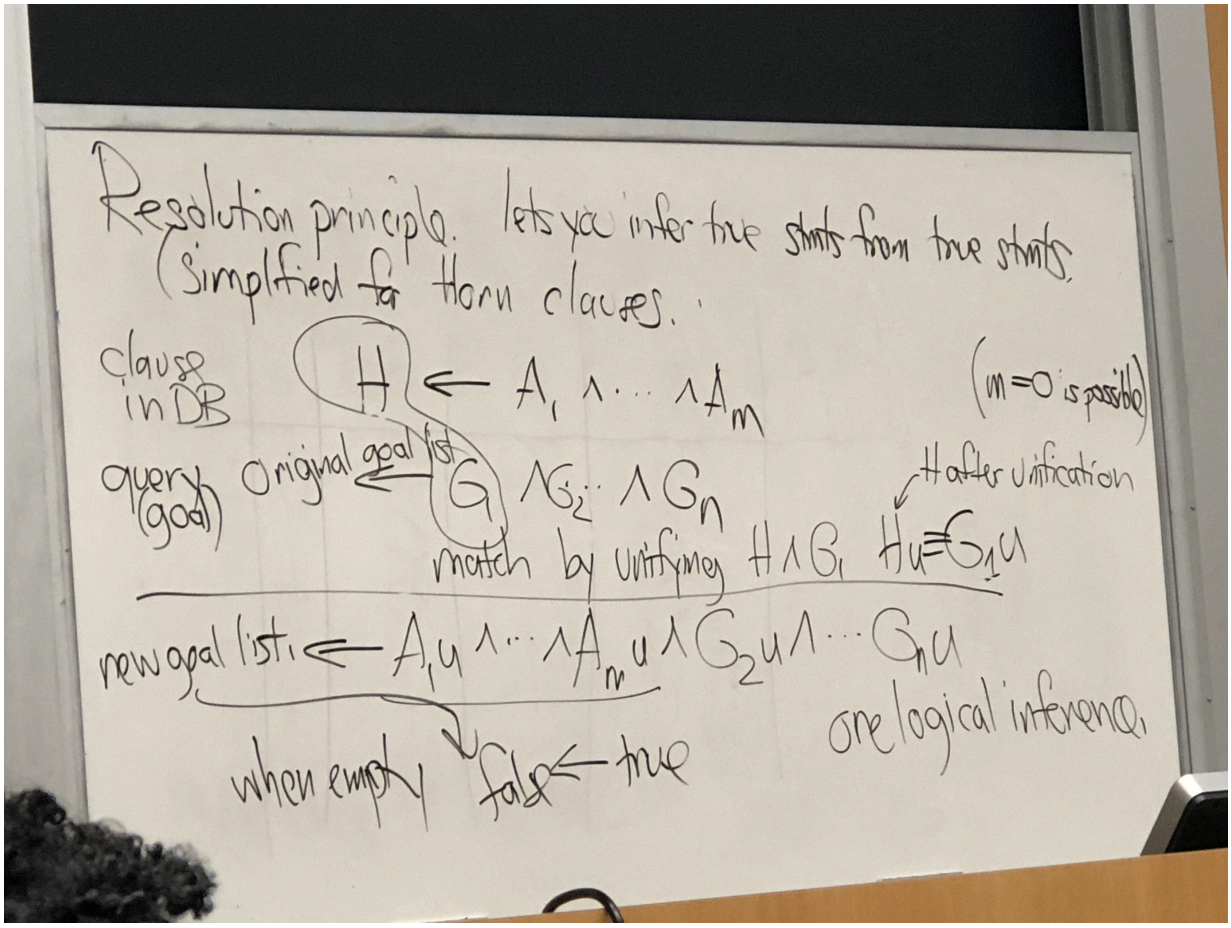
Prolog Interpreter
Works because Prolog interpreter uses proof by contradiction.

- true implies false means contradiction



Resolution Principle: lets you infer true stmts from true stmts
- simplified for Horn clauses.

clause in DB: H <- A1 & ... & Am (m = 0 is possible)
query (goal) <- G1 & ... & Gn



He could ask us how to figure out how the OCaml interpreter works

- slightly simpler: not logic proof by contradiction

Scope

```
{
	int n = &n == &n;
	int m = n + 1;
	{
		int n = m + 3;
		...
	}
}

struct x {
	int v;
	struct x *next;
```

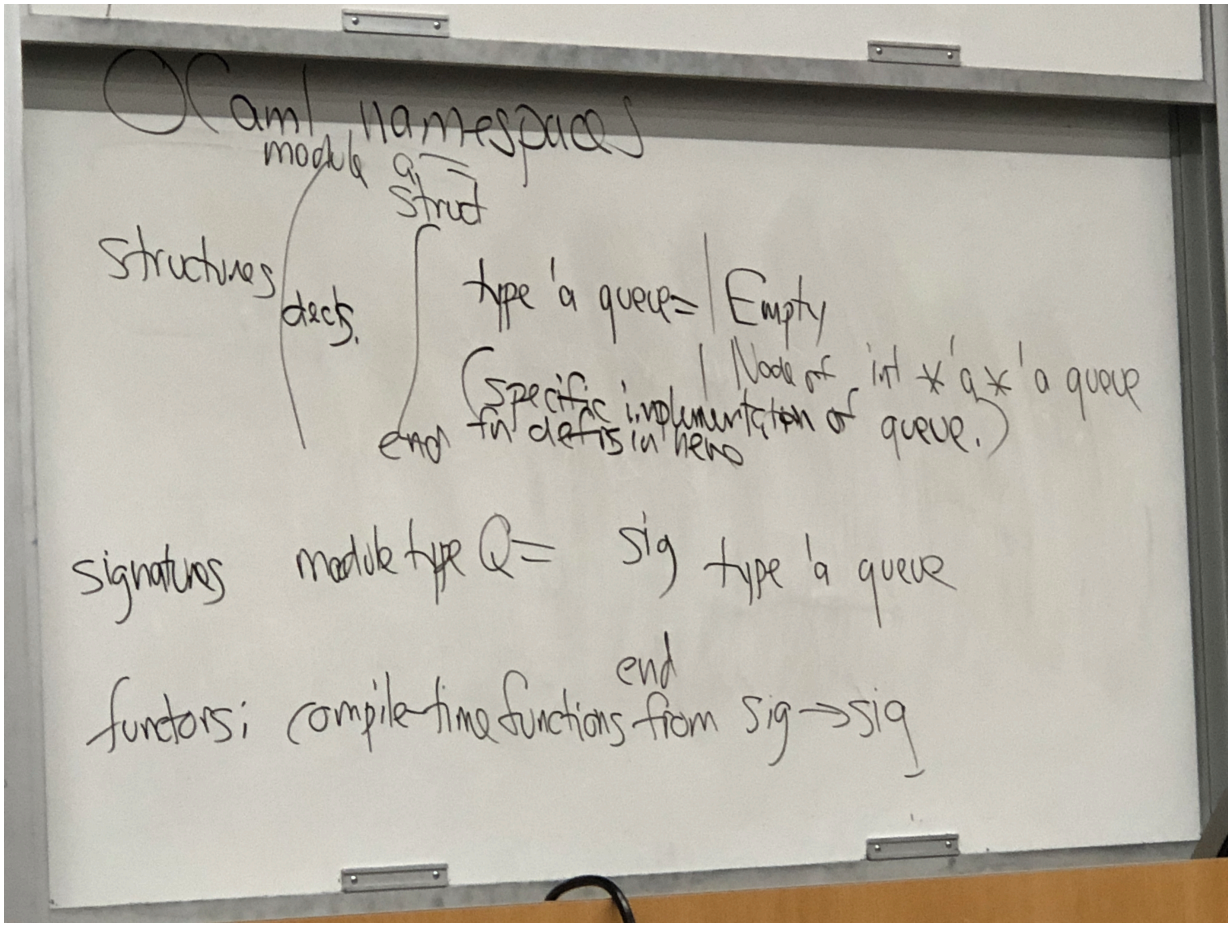
```
}
struct x v = {0, &v};
```

visibility modifiers in Java, visible in

	current class	another class in same package	other subclasses outside package	elsewhere
private	y			
default	y	y		
protected	y	y	y	
public	y	y	y	y

```
class c {
        int n;              // default visibility
}
```

Ocaml namespaces



Signature: like interface
Structures: the actual type variable implementing interface
Functors: compile time functions from sig —> sig

- more powerful than signature
- generates compile time functions from signatures

Storage management
warmup. arrays.

All the following need storage:

- string literals
- objects computed dynamically
  - new
- local vars \
- stack for recursion
- pointers to stack locations
- functions (machine code, closures)
- VNCZ flags (etc.)
- function args
- return addresses
- buffers for I/O
- memory manager

Closures
functions packed in other functions that carry information about the function returned
(fun x -> fun y -> x + y) 12
getchar()