CSC384: Lecture 2

- Last time
  - RRSs and Syntax of DCL
- Today
  - semantics of DCL; models and queries; variables; start on proof procedures
- Readings:
  - Today: Ch.2.5, 2.6, 2.7 (excl. SLD/top-down proofs)
  - Next week: 2.7 (rest); 2.8 (details in tutorial), Ch.3 (we'll discuss some)

Semantics of DCL

- Semantics:
  - how do attach *meaning* to sentences in KB?
  - to computer, just symbols; to us, correspond to world
- We want to associate symbols with entities in our domain of interest (real world)
  - constants/terms: *individuals*; atoms/etc: *facts*
- Purpose:
  - so we can interpret KB
  - so we can tell if KB facts are legitimate (true)
  - form basis for deciding *consequences* of KB

Interpretations

- Formally (mathematically) tricky
  - how do you capture relationship with “real world”?
- **Interpretations**: mathematical abstractions of real world, that talk about relevant aspects of the world in precise way
- Basic idea: we have an intended interpretation corresponding to our view of the world. We make sure KB consists only of sentences true in the intended interpretation
- From this: we’ll formally justify new conclusions

Interpretations (Defn)

- Assume language *L(F,P,V)*, where
  - *F = F(0), F(1), F(2),…* (function symbols of each arity)
  - *P = P(0), P(1), …* (predicate symbols of each arity)
  - *V* is set of variable symbols
- An interpretation $I = \langle D, \phi, \pi \rangle$ where
  - *D* is a non-empty set (domain of individuals)
  - ϕ is a mapping $\phi : F(k) \times D^k \rightarrow D$
  - π is a mapping $\pi : P(k) \times D^k \rightarrow \{T, F\}$

Intuitions: Domain

- Domain *D*: $d \in D$ is an *individual*
- E.g., { *craig*, *jane*, *grandhotel*, *lefleabag* *rome*, *siena* *100*, *110*, *120* …}
- Underlined symbols denote domain individuals (as opposed to language elements)
- Domains usually infinite, but we'll use finite models to prime our intuitions

Intuitions: Φ

- $\phi : F(k) \times D^k \rightarrow D$
  - given *k*-ary function, *k* individuals, what individual does *f(d1, … dk)* denote
- *F(0)*: take symbol from *F(0)*=*C*, (nb. *D0* = *{}*)
  - *Φ(client17) =* craig *Φ(hotel5) =* lefleabag *Φ(rome) =* rome
- *F(1)*: take symbol from *F(1)*, element $d \in D$
  - *Φ(minquality,* craig*) =* 3stars
  - *Φ(rating,* grandhotel*) =* 5stars
- *F(2)*: symbol from *F(2)*, two elements $d \in D$
  - *Φ(distance,* toronto*,*siena*) =* 3256km

Intuitions: π

- $\pi : P(k) \times D^k \rightarrow \{T, F\}$
  - given *k*-ary predicate, *k* individuals, does relation denoted by P hold of these? is *P(d1, … dk)* true?
  - what facts are made true by the interpretation?
- *P(0)*: take symbol from *P(0)*, (nb. *D0 = {}*)
  - *π(rainy) = T* *π(sunny) = F*
- *P(1)*: take symbol from *P(1)*, element *d∈D*
  - *π(satisfied, craig) = T* *π(privatebeach, lefleabag) = F*
- *P(2)*: symbol from *P(2)*, two elements *d∈D*
  - *π(location, grandhotel, rome) = T*
  - *π(location, grandhotel, siena) = F*
  - *π(available, grandhotel, week29) = T*

What Language Elements Denote

- Given language *L(F,P,V)*, interpret. $I = \langle D, \phi, \pi \rangle$

(a) Constant $c \in C$ *denotes* individual $I(c) = \Phi(c)$

(b) *Ground* term $t = f(t_1, \ldots t_k)$ *denotes* $I(t) \in D$:

where $I(t) = \phi(f, \langle d_1, \cdots d_k \rangle)$

and $I(t_k) = d_k, \ \forall k \leq n$

(c) *Ground* atom $a = p(t_1, \ldots t_k)$ has *truth value* $I(a) \in \{T,F\}$, where:

$I(a) = \pi(p, \langle d_1, \cdots d_k \rangle) \qquad I(t_k) = d_k, \ \forall k \leq n$

Example

I(happy(father(client15))) = π(happy, I(father(client15)))

I(father(client15)) = Φ(father, I(client15))
= Φ(father, Φ(client15))
= Φ(father, craig)
= bill

I(happy(father(client15))) = π(happy, bill)
= TRUE

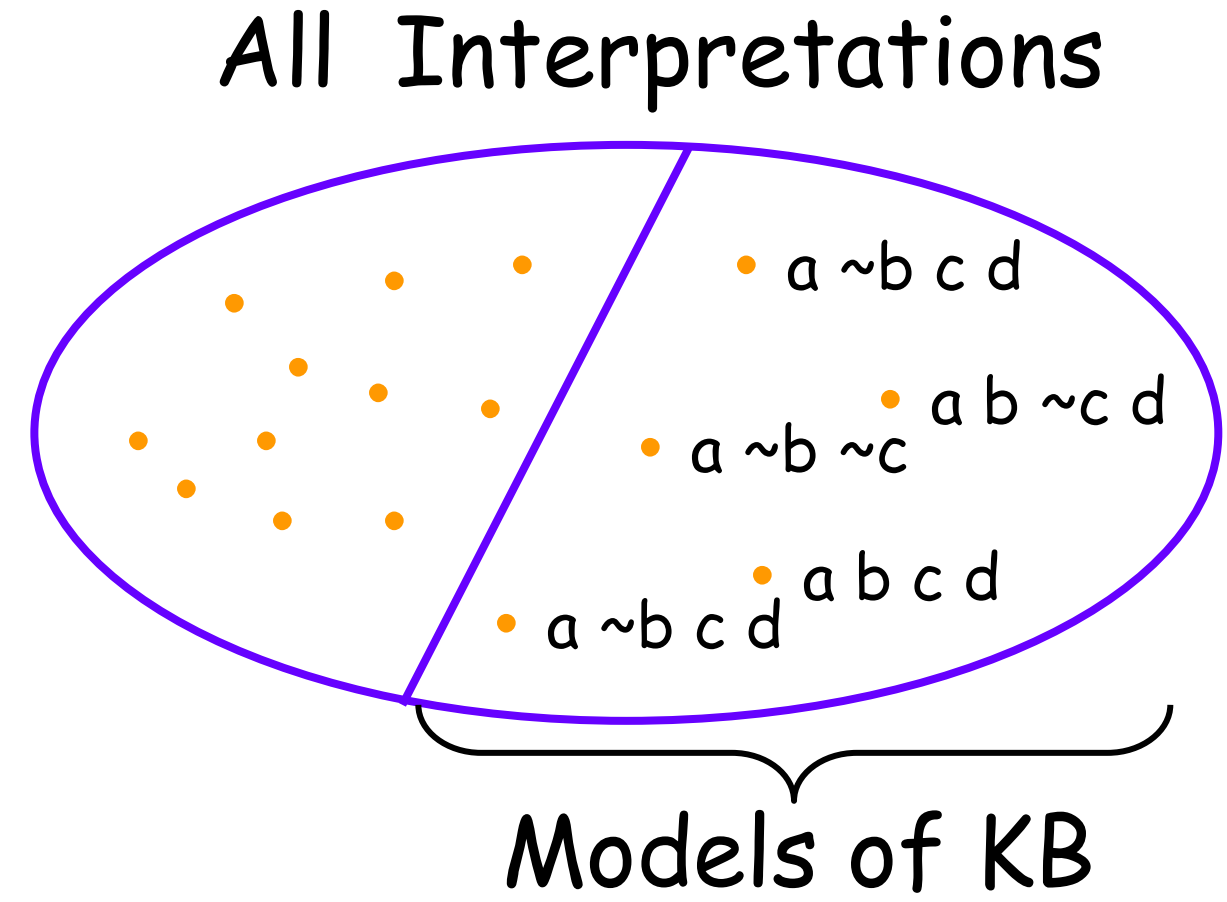
Models

- A *ground body* $a_1 \& a_2 \& \ldots \& a_n$ is true under *I* iff each a_i is true under *I*
- A *ground rule* $a \leftarrow$ <*body*> is true under *I* iff its head *a* is true or its body is false
  - "if body then head": material implication
- A KB is true under *I* iff each fact/rule in KB is true
- In this case, we say *I* is a *model* of KB or that I *satisfies* KB
- We write *I* |= *KB* (and *I* |= *a* and *I* |= *rule*)

What's Special About Models?

- When we write KB, we intend that the real world (or our abstraction of it) is one of its models
- Suppose fact *f* is not mentioned in KB, but is *true in every model of KB*; i.e., $I \models f$ for each model *I* of KB.
- Since real world is a model, *f* must be true in the real world: *f* is a *logical consequence* of KB
- KB *entails* f (written $KB \models f$) iff $I \models f$ for each model *I* of KB; (*same applies to bodies, rules)*
  - *??? If KB doesn't entail f, is f false in real world?*

Models Graphically



Consequences?

Queries

- A query is evaluated wrt a KB:
  - "Must the query be true, given that the KB is true?"

?b is answered YES iff $KB \models b$

?b is answered NO iff $KB \not\models b$

Models, Interpretations, Queries

- You include sentences in KB. The more sentences you include, the fewer models (satisfying interpretations) there are. The more you write down (as long as it's all true!), the "closer" you get to the "real world"! Each sentence in KB rules out certain unintended interpretations.
- This is called *axiomatizing the domain*
- A query is true iff it is true in all of these models, hence, in the intended interpretation.

What are some consequences?

1. busy(craig) <- teaches(craig,384) & teaches(craig,148) .

2. busy(craig) <- teaches(craig,384) & teaches(craig,324) .

3. smart(craig) <- teaches(craig,384) .

4. teaches(craig,384) .

5. sad(craig) <- busy(craig) & sunny .

---

6. teaches(craig, 148) .

---

7. rainy. (or sunny .)

Variables

- We've defined semantics for ground KBs
  - what terms, atoms, rules, denote; what they entail
- *Variables*: allow universally quantified facts/rules
  - Rule $c(X) \leftarrow a(X) \,\&\, b(X)$: $\forall X.a(X) \wedge b(X) \supset c(X)$
- *E.g.*,

```
happy(C) <- desires(C,Date) & gets(C, H, Date).

busy(Z) <- teaches(Z,X) & teaches(Z,Y)
           & distinct(X,Y).
```

Semantics of Variables

happy(C) <- desires(C,Date) & gets(C, H, Date).

- Interpretation: no matter what individuals in *D* are assigned to *C*, *Date, H*, the rule is true
  - In example, *I* only satisfies rule if *each* client who gets a hotel on desired date is happy (regardless of hotel)
- Formally, satisfaction is defined using the notion of *variable assignments* and *uniform substitution* of individuals for vars in a clause (see text)

Variables in Queries

- Suppose we have a query with a var: *?happy(C)*
  - *could* interpret this as “Are all individuals happy?”
  - i.e., is *∀C.happy(C)* a logical consequence of KB
- *Instead* interpret as “Tell me those individuals who are busy”
  - unfortunately, individuals require a specific domain (in a specific interpretation)
  - so we want terms, *specifically ground terms t,* such that *happy(t)* is a logical consequence of KB

Variables in Queries

KB:

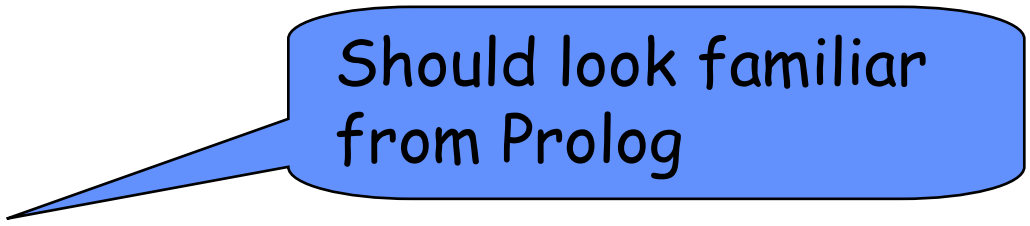
```
busy(Z) <- teaches(Z,X) & teaches(Z,Y)
            & distinct(X,Y).
teaches(craig, 384).   teaches(craig, 2534).
teaches(kyros, 384).   teaches(kyros, 2501).
teaches(suzanne, 324).
distinct(384, 2534).  distinct(384, 2501). distinct…
```

Query: ? busy(X).

X = craig

X = kyros

and that's it.



Proof Procedures

- How do we get computer to determine logical consequences of a KB?
  - certainly can't expect it to enumerate models (infinite)
- A **proof procedure** is an algorithmic procedure for deriving logical consequences of a KB,
  - *without (usually) recourse to model enumeration*
  - derivation by syntactic means (generally)
- Given KB, query ?q, proof procedure:
  - returns YES iff $KB \vdash q$
  - returns NO iff $KB \nvdash q$

Derivations with variables

- $KB \vdash q$ refers to the fact that our (specific) proof procedure can *derive* q from KB
- For a nonground query $?q(X_1, \ldots, X_k)$, procedure returns *set* of tuples $(t_1, \ldots, t_k)$ of ground terms such that $KB \vdash q(t_1, \ldots, t_k)$

```
?teaches(X, Y).
    X = craig, Y = 2534
    X = kyros, Y = 384
    X = craig, Y = 384
    etc.
```

Soundness and Completeness

- Two important properties of proof procedures
- **Soundness**: If $KB \vdash q$, then $KB \models q$
  - proof procedure gives *only* correct answers (says YES only to genuine logical consequences)
  - what is simplest possible sound procedure?
- **Completeness**: If $KB \models q$, then $KB \vdash q$
  - proof procedure gives *all* correct answers (says YES to every genuine logical consequence)
  - what is simplest possible complete procedure?
- Ideally, we have a sound and complete proof pr.
  - is a sound and complete procedure always possible?

Bottom Up Proof Procedure -- for ground KBs and queries

- Assume everything ground
  - we can focus only on prop. atoms (ignore terms)
- Bottom-up procedure motivated as follows:
  - if atom *p* is in *KB*, then *p* is true in all models of *KB*
  - if rule $R = h \leftarrow p_1 \,\&\, \ldots \&\, p_k$ is in *KB*, and $p_1, \ldots, p_k$ are true in all models of *KB,* then so is atom *h*
- This is equivalent to *modus ponens*
- For simplicity, treat fact *h* as a rule $h \leftarrow .$
  - i.e., a fact is a rule with an empty body

BUPP: Algorithm

1. Let C = ∅ (C is the set of logical consequences)
2. Until no clause is selectable
   (a) Select a clause $h \leftarrow p_1 \,\&\, \ldots \,\&\, p_k$ such that $p_1 \in C, \ldots, p_k \in C$ and $h \notin C$
   (b) Add h to C

- A clause is selectable if you've proven it's body
  - so adding it's head is justified (head is provable)
- When no clause selectable, C is a *fixed point*
- Selection is nondeterministic (no order imposed)

BUPP: Example

KB: (1) a <- b & c.
(2) b <- d & e.
(2') b <- c.
(3) b <- g & e.
(4) c <- e.
(5) d.
(6) e.
(7) f <- a & g.

Consequences (selecting first selectable clause in order they occur in KB)

C = ∅
C = {d} (clause 5)
C = {d, e} (6)
C = {d,e,b} (2)
C = {d,e,b,c} (4)
C = {d,e,b,c,a} (1)

- No more clauses selectable
- g not provable (not in any head)
- (3), (7) could never be selected
- (2') could have been selected if used different order

Some Considerations on BUPP

- Can the order of selection matter?
  - *Can selecting one clause prevent you from selecting another?*
- Starvation: must eventually *consider* each clause
  - Once you select a clause, never consider it again
- How to answer a query?
  - Generate set of consequences C and ask if $q \in C$
  - For *specific* query, can stop early if we generate it
- Exercise: prove soundness (easy; see text)
  - proving completeness harder (text)
  - relies on notion of a minimal model
- Complexity: $O(|KB|^2)$
  - no more than |KB| iterations
  - naïve search for selectable clause: O(|KB|)

Variables in BUPP (no functions)

- With vars, goal is to derive *ground instances* of atoms; i.e., substitute ground terms for variables
- If no functions, only ground terms are constants
  - So plug constant symbols into argument slots
- Assume KB is finite, only finite # of constant symbols mentioned in KB and query
- Must be careful if we have KB elements like p(X). May need to assume finite language.

Constant Substitutions

- A *constant substitution* is an assignment of constants to variables
  - We write this as {X/a, Y/b}, etc.
- The *application of constant substitution* to any expression: uniformly substitute that constant for every occurrence of the var in that expression

Applying: {X/craig, Y/384}
to: happy(X) <- teaches(X,Y) & fun(Y).
yields: happy(craig) <- teaches(craig,384) & fun(384).

- Note that we create a ground instance of a clause by substituting ground terms for each var

BUPP with variables: Algorithm

1. Let C = ∅ (C is the set of logical consequences)

2. Until no clause is selectable

   (a) Select a clause $h(\mathbf{Z}) \leftarrow p_1(\mathbf{Z}_1) \,\&\, \ldots \,\&\, p_k(\mathbf{Z}_k)$
   and a constant subst. $\{X_1/c_1, \ldots, X_n/c_n\}$
   s.t. $p_1(\mathbf{c}_1) \in C, \ \ldots, \ p_k(\mathbf{c}_k) \in C$

   (b) Add $h(\mathbf{c})$ to C

- Here boldface type refers to vectors (sets)
  - the vars X_i refer to all vars among the $\mathbf{Z}_i$
  - the constant tuples $\mathbf{c}_i$ refer to subst. applied to $\mathbf{Z}_i$

BUPP with variables: Example

KB: (1) p(X,Y).
(2) s(Y) <- p(X,Y) & t(Y).
(3) t(m).
(4) t(g).

Language has constants: m, n, g

All derivable facts:

p(m,n).	p(g,n).	p(n,n).	t(g).	t(m).
p(n,m).	p(m,g).	p(n,g).	s(m).	
p(m,m).	p(g,m).	p(g,g).	s(g).	

Some Notes

- Datalog (DCL w/ no function symbols)
  - BUPP is sound and complete
  - Completeness depends on termination, but since a finite number of constants, only a finite number of substitutions to consider
  - If language is infinite language? Only need to consider “mentioned” constants (KB, query)
- What if function symbols?
  - need to consider ground terms (not just constants)
  - how many ground terms can we substitute???