CSC384h: Intro to Artificial Intelligence

- Knowledge Representation

  - This material is covered in chapters 7—9 and 12 of the text.
  - Chapter 7 provides a useful motivation for logic, and an introduction to some basic ideas. It also introduces propositional logic, which is a good background for first-order logic.
  - What we cover here is mainly covered in Chapters 8 and 9. However, Chapter 8 contains some additional useful examples of how first-order knowledge bases can be constructed. Chapter 9 covers forward and backward chaining mechanisms for inference, while here we concentrate on resolution.
  - Chapter 12 covers some of the additional notions that have to be dealt with when using knowledge representation in AI.

Knowledge Representation

- What is knowledge?
- Information we have about the world we inhabit
  - Both the physical and mental world.
  - We have knowledge about many abstract mental constructs and ideas
- Besides knowledge we have various other mental attitudes and feelings about our environment.
  - John knows "…"
  - John fears "…"
  - Then things get complex: John knows that he fears "…"
  - So knowledge can take a variety of forms, some quite complex

Knowledge Representation

- What is Representation?
- Symbols standing for things in the world

CSC 384

Words we use in language

Symbols we use in mathematics

- Can all knowledge be symbolically represented?

Knowledge Representation

- Can all knowledge be symbolically represented?
- No - we do not symbolically represent the "pixels" that we perceive at the back of our retina.
- So intelligent agents also perform a great deal of low level "non-symbolic reasoning" over their perceptual inputs.
- But higher level "symbolically represented" knowledge also seems to be essential
  - This is the kind of knowledge that we learn in school, by reading, etc.
- In this module we study symbolically represented knowledge

Reasoning

- What is reasoning (in the context of symbolically represented knowledge) ?
  - Manipulating our symbols to produce new symbols that represent new knowledge.

    Deriving a new sentence
  - Typically "symbols" are sequences of symbols, e.g., words in language sequenced together to form sentences.
  - So we will develop methods for manipulating "sentences" to produce new "sentences"

Reasoning

- In language we can make up a huge variety of sentences.
- Each of these sentences makes some sort of claim or assertion about our world (mental or physical).
- These claims could be true or false.
    - I am anxious, so the sentence "I feel calm and relaxed." Is false
- Reasoning aims to be **TRUTH PRESERVING**.
- If we use reasoning to manipulate a collection of **TRUE** sentences, we want the newly derived sentences to also be **TRUE**

- If our reasoning is truth preserving we say that is is **SOUND**

Reasoning

- A more subtle idea is **COMPLETENESS**.
- Completeness says that our reasoning system is powerful enough to produce **ALL** sentences that must be true given one current collection of true sentences.
- Completeness requires a formal characterization of "sentence" in order to answer the question of if we have produced **ALL** true sentences

Knowledge Representation

- Consider the task of understanding a simple story.
- How do we test understanding?
- Not easy, but understanding at least entails some ability to answer simple questions about the story.

Example.

- Three little pigs

Example.

- Three little pigs

Example.

- Why couldn’t the wolf blow down the house made of bricks?

- What background knowledge are we applying to come to that conclusion?

Why Knowledge Representation?

- Large amounts of knowledge are used to understand the world around us, and to communicate with others.
- We also have to be able to reason with that knowledge.
  - Our knowledge won't be about the blowing ability of wolfs in particular, it is about physical limits of objects in general.
  - We have to employ reasoning to make conclusions about the wolf.
  - More generally, reasoning provides an exponential or more compression in the knowledge we need to store. I.e., without reasoning we would have to store a infeasible amount of information: e.g., Elephants can't fit into teacups.

Logical Representations

- AI typically employs logical representations of knowledge.

- Logical representations useful for a number of reasons:

Logical Representations

- They are mathematically precise, thus we can analyze their limitations, their properties, the complexity of inference etc.
- They are formal languages, thus computer programs can manipulate sentences in the language.
- They come with both a formal syntax and a formal semantics.
- Typically, have well developed proof theories: formal procedures for reasoning at the syntactic level (achieved by manipulating sentences).
- In this module we will study First-Order logic, and a reasoning mechanism called resolution that operates on First-Order logic.

First Order Logic (FOL)

- Two components: Syntax and Semantics.
  - In a programming language we have a syntax for an if statement: “if <boolean condition>:<expressions>”
  - The if statement also has semantics: if <boolean condition> evaluates to TRUE then we execute <expressions>.
- Syntax gives the grammar or rules for forming proper sentences.
- Semantics gives the meaning.

Basic Semantic entities of FOL

- We have a set of objects **D.** These are objects in the world that are important for our application.
  - Often we will want to form tuples of objects, e.g., (d1,d2) where d1$\in$ **D** and d2 $\in$ **D** are a pair of objects
  - A k-ary tuple is a subset of **D^k = D X D X … X D** the k-wise Cartesian product of **D**
- We can identify special sets of objects (subsets of **D**) that have some property in common. These sets are called **properties (predicates)**.
  - E.g., **female, male, children, adult** could each nee subsets that we identify as being useful in our application. If an object **d** is in the set **male**, we can say that d has the property male : **male(d).**

Basic Semantic entities of FOL

- Sometimes individual objects are not sufficient, we want to identify special groups (tuples) of objects that are related to each other. We call these sets **relations**.
  - E.g. **married** might be a special subset of pairs that we wish to keep track of in out application.
- Finally, we might want to keep track of functions over our objects. f: **D** ➔ **D**
  - E.g. for d ∈ student, we might want a function faculty(d) that gives the faculty the student is registered in.
  - More generally we might want f: $\mathbf{D}^k$ ➔ **D**, i.e., a function of many arguments mapping **D**.

Basic Syntactic symbols of FOL

- The syntax starts off with a different symbol for each basic semantic entity (objects, functions, predicates, relations) that we have decided to utilize.
- We get to decide what symbols we use (but of course want to use symbols that are easy to understand
- These user specified symbols are called the **primitive symbols**

Syntax	Semantics
Constant symbols	A particular object $d \in \mathbf{D}$
Function symbols	Some function f: $\mathbf{D}^k \rightarrow \mathbf{D}$
Predicate symbols	Some subset of $\mathbf{D}$
Relation symbols	Some subset of $\mathbf{D}^k$

Basic Syntactic symbols of FOL

- In addition we introduce some additional symbols that we will use to connect our basic symbols into sentences.

Syntax	Semantics
Constant symbols	A particular object $d \in \mathbf{D}$
Function symbols	Some function f: $\mathbf{D}^k \rightarrow \mathbf{D}$
Predicate symbols	Some subset of **D**
Relation symbols	Some subset of $\mathbf{D}^k$
Equality (commonly used relation)	Subset of $\mathbf{D}^2 = \{(d,d) \mid d \in \mathbf{D}\}$
Variables (as many as we need) x,y,z,…	An object $d \in \mathbf{D}$ (which particular object can vary)
Logical connectives: $\land, \lor, \neg, \rightarrow$	…defined below…
Quantifiers: $\forall, \exists$	…defined below…

Example

- Teaching CSC384, want to represent knowledge that would be useful for making the course a successful learning experience. So we might choose syntactic symbols like
- Objects:
  - students,subjects,assignments,numbers.
- Predicates:
  - difficult(*subject*), CSMajor(*student*)
- Relations:
  - handedIn(*student,assignment*)
- Functions:
  - Grade(*student,assignment*) → *number*

First Order Syntax (the grammar)

- We start with out basic syntactic symbols constants, functions, predicates, relations and variables.
  - Note: the function and relation symbols each have specific arities (the number of arguments it takes)

- From these we can build upon **terms** and **sentences(formulas)**. Terms are ways of applying functions to build up new "names" for objects. Formulas, are denoting true/false assertions about terms.

First Order Syntax - Terms

- Terms are used as names (perhaps complex nested names) for objects in the domain.

Terms	
Constants	c, john, mary
Variables	x, y, z, …
Function application	$f(t_1, t_2, \ldots, t_k)$ t_i are already constructed terms

- 5 is a constant term: a symbol representing the number 5. john is a term — a symbol representing the person John.
- +(5,5) is a function application term — a new symbol representing the number 10.

First Order Syntax - Terms

- **Note**: constants are the same as functions taking zero arguments.
- Terms are names for objects (things in the world):
  - Constants denote specific objects
  - Functions map tuples of objects to other objects
    - bill, jane, father(jane), father(father(jane))
    - X, father(X), hotel7, rating(hotel7), cost(hotel7)
  - Variables like X are not yet determined, but they will eventually denote particular objects.

First Order Syntax - Sentences.

- Once we have terms we can build up *sentences* (*formulas*) Terms represent objects, *formulas* represent true/false assertions about these objects

First Order Syntax - Sentences.

Formula	
Atomic formula	**p(t)** or **r(t_1, t_2, …, t_k)** p is a predicate symbol, r is a k-ary relation symbol, t_i are terms
Negation	**¬f** f is a fomula
Conjunction	$f_1 \land f_2 \land \ldots \land f_k$ f_i are formulas
Disjunction	$f_1 \lor f_2 \lor \ldots \lor f_k$
Implication	$f_1 \rightarrow f_2$ f_1 and f_2 are fomulas f_1 often calles the antecedent, f_2 the consequence
Existential	**∃X.f** f is a formula X is a variable
Universal	**∀X.f**

Intuition (formalized later).

- Atoms denote facts that can be true or false about the world
  - father_of(jane,bill), female(jane), system_down()
  - satisfied(client15), satisfied(C)
  - desires(client15,rome,week29), desires(X,Y,Z)
  - rating(hotel7,4), cost(hotel7,125)
- Other formulas generate more complex assertions by composing these atomic formulas.
  - Their truth is dependent on the truth of the atomic formulas in them

Semantics

- Formulas (syntax) can be built up recursively, and can become arbitrarily complex

- Intuitively, there are various distinct formulas (viewed as strings) that really are asserting the same thing
  - $\forall X,Y.\ elephant(X) \wedge teacup(Y) \rightarrow largerThan(X,Y)$
  - $\forall X,Y.\ teacup(Y) \wedge elephant(X) \rightarrow largerThan(X,Y)$

- To capture this equivalence and to make sense of complex formulas we utilize the semantics

Semantics

- A formal mapping from formulas to true/false assertions about our semantic entities (individuals, sets and relations over individuals, functions over individuals).

- The mapping mirrors the recursive structure of the syntax, so we can map any formula to a composition of assertions about the semantic entities.

Syntax - The language

- First, we must fix the particular first-order language we are going to provide semantics for. The **primitive** symbols included in the syntax defines the particular language.

L(F,P,V)

F = set of function (and constant symbols)

Each symbol f in F has a particular arity.

P = set of predicate and relation symbols.

Each relation symbol r $\in$ *P has a particular arity. (The predicate symbols always have arity 1)*

V = an infinite set of variables.

Semantics - Primitive Symbols

- An interpretation (model) specifies the mapping from the primitive symbols to semantic entities. It is a tuple $\langle$ D,$\boldsymbol{\Phi}$,$\boldsymbol{\Psi}$,V $\rangle$
  - D is a non-empty set of objects (domain of discourse)
  - $\boldsymbol{\Phi}$ specifies the meaning of each primitive function symbol
    - Also handles the primitive constant symbols (these can be viewed as being zero-arity functions.
  - $\boldsymbol{\Psi}$ specifies the meaning of each primitive predicate and relation symbol.
  - V specifies the meaning of the variables.

- Note, the semantic entities that a syntactic symbol maps to is often called the meaning of the symbol or the denotation of the symbol

Semantics - Primitive Symbols

Symbol	Semantics
Constant Symbol c	Φ(c) $\in$ **D** (some particular object)
K-ary function symbol f	Φ(f) Some particular function $\mathbf{D}^k \rightarrow \mathbf{D}$
Predicate symbol p	Ψ(p) Some particular subset of **D**
K-ary relation symbol r	Ψ(r) Some particular subset of $\mathbf{D}^k$
Variable x	V(x) $\in$ **D** (some particular object)

Intuitions: Domain

- Domain D: $d \in D$ is an *individual*
- E.g. {craig, jane, grandhotel, marriot, rome, portofino, 100, 110, 120 …}
- We use underlined symbols to talk about domain individuals (syntactic symbols of the first-order language are not underlined)
- Domains often infinite, but we'll use finite models to prime our intuitions

Intuitions: Φ

- Given *k*-ary function f and *k* individuals d1…dk, what individual does *f(d1,…,dk)* denote
  - Constants (0-ary functions) are mapped to individuals in **D**.
    - Φ(client17) = craig, Φ(hotel5) = marriot, Φ(rome) = rome
  - 1-ary functions are mapped to particular functions in **D**→**D**
    - Φ(rating) = f_rating:
      - f_rating(grandhotel) = 5stars
  - 2-ary functions are mapped to functions from $\mathbf{D}^2 \rightarrow \mathbf{D}$
    - Φ(distance) = f_distance:
      - f_distance(toronto, sienna) = 3256
  - N-ary functions are mapped similarily

Intuitions: Ψ

- Given k-ary relation r, what does r denote
- 0-ary predicates are mapped to true or false.
  Ψ(rainy) = True   Ψ(sunny) = False
- 1-ary predicates are mapped to subsets of **D**.
  - *Ψ(*privatebeach) = p_privatebeach*:* (the subset of hotels that have a private beach)
    e.g. p_privatebeach = {grandhotel, fourseasons}
- 2-ary predicates are mapped to subsets of D^2 (sets of pairs of individuals)
  - *Ψ(location) = p_location: p_location(grandhotel, rome) = True*
    p_location(grandhotel, sienna) = False
  - *Ψ(available) = p_available:*
    p_available(grandhotel, week29) = True
- n-ary predicates..subsets of Dn

Intuitions: v

- V exists to take care of quantification. As we will see the exact mapping it specifies will not matter.

Semantics — Terms

- Given language L(F,P,V), and an interpretation **I** = ⟨ D,Φ,Ψ,V ⟩ and a term **t**. **I(t)** is the denotation of **t** under **I**.

Term	Semantics
Constant Symbol c	$I(c) = \Phi(c) \in \mathbf{D}$ (some particular object)
Variable x	$I(x) = V(x) \in \mathbf{D}$ (some particular object)
Function application $f(t_1,t_2,\ldots,t_k)$	$I(f(t_1,t_2,\ldots,t_k)) = \Phi(f)(\, I(t_1), I(t_2),\ldots,I(t_k))$ First we obtain the denotation of each argument under I, then we apply the function Φ(f) to these interpreted terms

- Hence the terms always denote individuals under interpretation I

Semantics — Formulas

- Formulas will always be True or False under any interpretation **I**.

Formula	Semantics
Atomic formula $r(t_1,t_2,\ldots,t_k)$	$I(r(t_1,t_2,\ldots,t_k))$ = True if $(I(t_1), I(t_2),\ldots,I(t_k)) \in \Psi(r)$ False otherwise First we obtain the denotation of each argument under I. Then we check if this tuple of interpreted terms is in the set of tuples $\Psi(r)$

- Ψ Maps r to a subset of D^k (a subset of k-ary tuples of individuals). So the atomic formula is true if its arguments are in the stated relation.

Semantics — Formulas

Formula	Semantics
$\neg f$	$I(\neg f) =$ True if $I(f) =$ False False otherwise
$f_1 \wedge f_2 \wedge \ldots \wedge f_k$	$I(f_1 \wedge f_2 \wedge \ldots \wedge f_k) =$ True if $I(f_i) =$ True for every i False otherwise
$f_1 \vee f_2 \vee \ldots \vee f_k$	$I(f_1 \vee f_2 \vee \ldots \vee f_k) =$ True if $I(f_i) =$ True for any i False otherwise
$f_1 \rightarrow f_2$	$I(f_1 \rightarrow f_2) =$ True if $I(f_1) =$ False or $I(f_2) =$ True False otherwise

- Standard rules for proposition logic that you would have seen before (check chap 7 if not)

Semantics — Formulas

Formula	Semantics
$\exists X.f$	$I(f) =$ True if for some $d \in \mathbf{D}$, $I'(f) =$ True $I' = \langle \mathbf{D}, \Phi, \Psi, V[X=d] \rangle$ False Otherwise
$\forall X.f$	$I(f) =$ True if for all $d \in \mathbf{D}$, $I'(f) =$ True $I' = \langle \mathbf{D}, \Phi, \Psi, V[X=d] \rangle$ False Otherwise

- Quantifiers. Exists checks if f is true under some different variable mapping for the variable X. Forall checks if f is true under all possible mappings of the variable X.

Example

D = {bob, jack, fred}

I(happy = {bob, jack, fred}

I($\forall$X.happy(X))

1. Ψ(happy)(X) ,(V[X = bob]) = Ψ(happy)(bob) = True

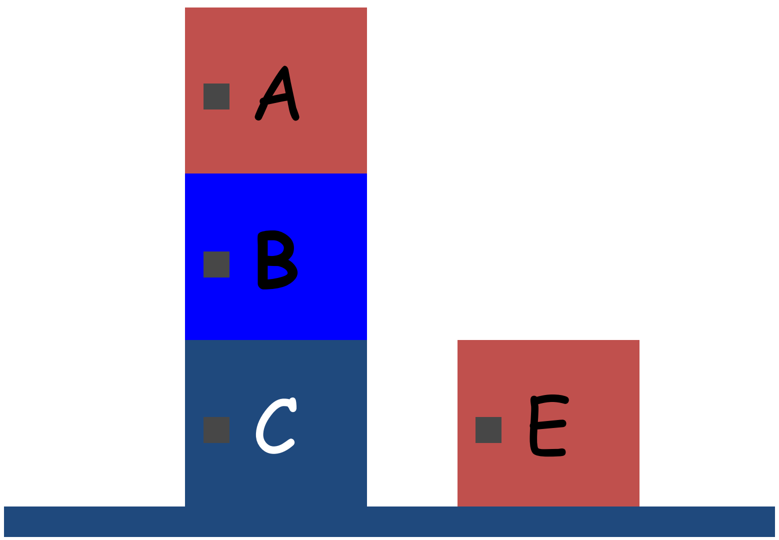
2. Ψ(happy)(X), (V[X = jack]) = Ψ(happy)(jack) = True

3. Ψ(happy)(X), (V[X = fred]) = Ψ(happy)(fred) = True

Therefore I($\forall$X.happy(X)) = True.

Models—Examples.

Environment



Language (Syntax)

- Constants: a,b,c,e
- Functions:
  - No function
- Predicates:
  - on: binary
  - above: binary
  - clear: unary
  - ontable: unary

Models—Examples.

Language (syntax)

- Constants: a,b,c,e
- Predicates:
  - on (binary)
  - above (binary)
  - clear (unary)
  - ontable(unary)

A possible Model I_1 (semantics)

- D = {A, B, C, E}
- Φ(a) = A, Φ(b) = B, Φ(c) = C, Φ(e) = E.
- Ψ(on) = {(A,B),(B,C)}
- Ψ(above)=

  {(A,B),(B,C),(A,C)}
- Ψ(clear)={A,E}
- Ψ(ontable)={C,E}

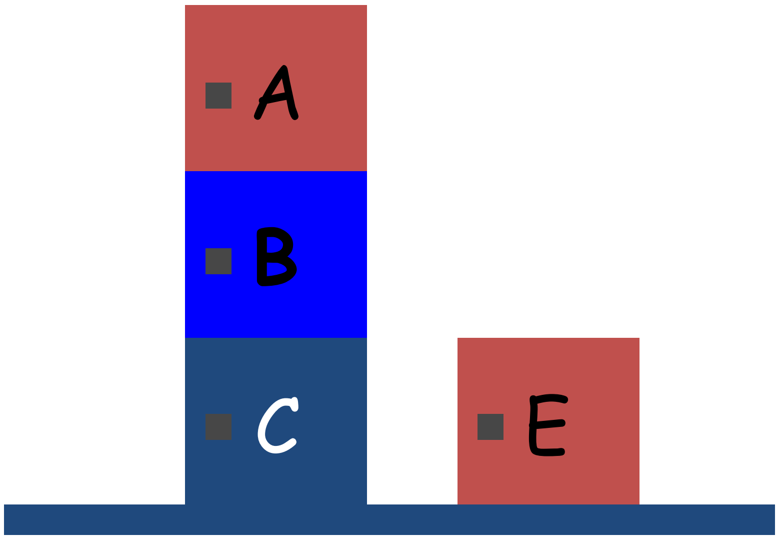
Models—Examples.

Model I_1

- D = {A, B, C, E}
- Φ(a) = A, Φ(b) = B, Φ(c) = C, Φ(e) = E.
- Ψ(on) = {(A,B),(B,C)}
- Ψ(above)=

  {(A,B),(B,C),(A,C)}
- Ψ(clear)={A,E}
- Ψ(ontable)={C,E}

Environment



Models—Formulas true or false?

Model I_1

- D = {A, B, C, E}
- Φ(a) = A, Φ(b) = B, Φ(c) = C, Φ(e) = E.
- Ψ(on) = {(A,B),(B,C)}
- Ψ(above)=

  {(A,B),(B,C),(A,C)}
- Ψ(clear)={A,E}
- Ψ(ontable)={C,E}

$\forall$X,Y. on(X,Y)→above(X,Y)

X=A, Y=B. ?

X=C, Y=A ?

…

$\forall$X,Y. above(X,Y)→on(X,Y)

X=A, Y=B ?

X=A, Y=A ?

X=A, Y=C ?

Models—Examples.

Model I_1

- D = {A, B, C, E}
- Φ(a) = A, Φ(b) = B, Φ(c) = C, Φ(e) = E.
- Ψ(on) = {(A,B),(B,C)}
- Ψ(above)=

  {(A,B),(B,C),(A,C)}
- Ψ(clear)={A,E}
- Ψ(ontable)={C,E}

$\forall X \exists Y. (clear(X) \vee on(Y,X))$

X=A

X=C, Y=B

…

$\exists Y \forall X. (clear(X) \vee on(Y,X))$

Y=A ?

Y=C?

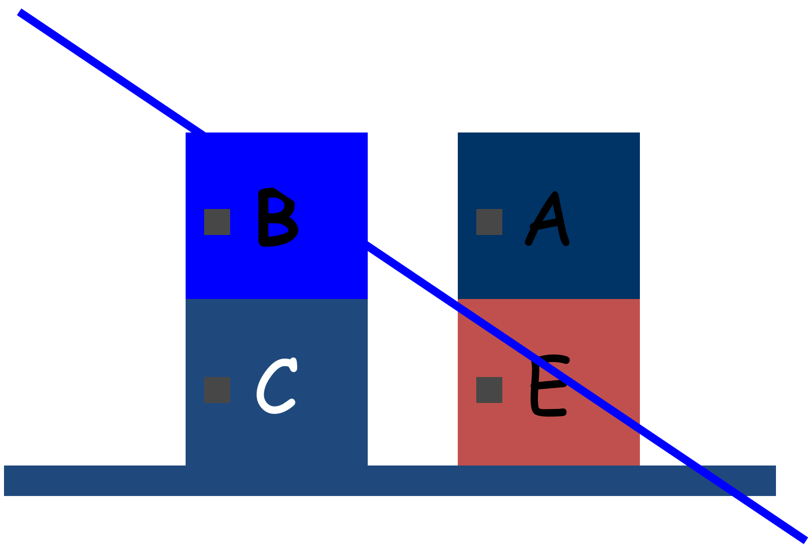
Y=E?

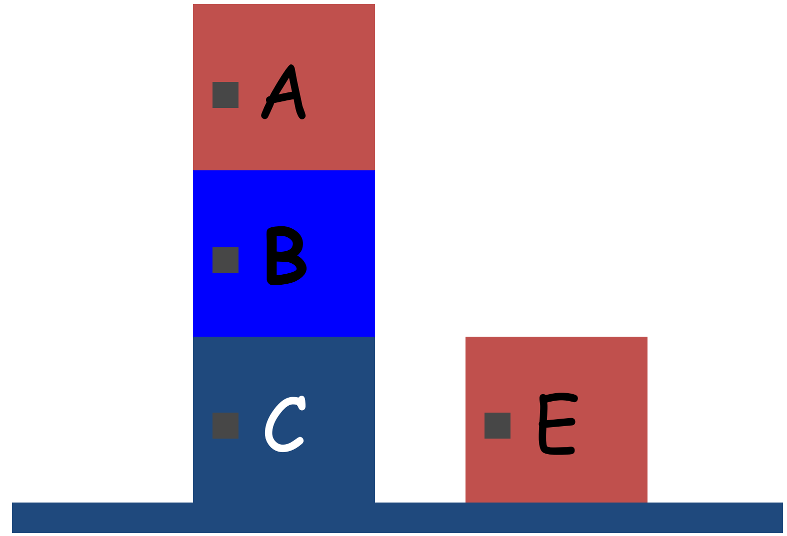
Y=B ?

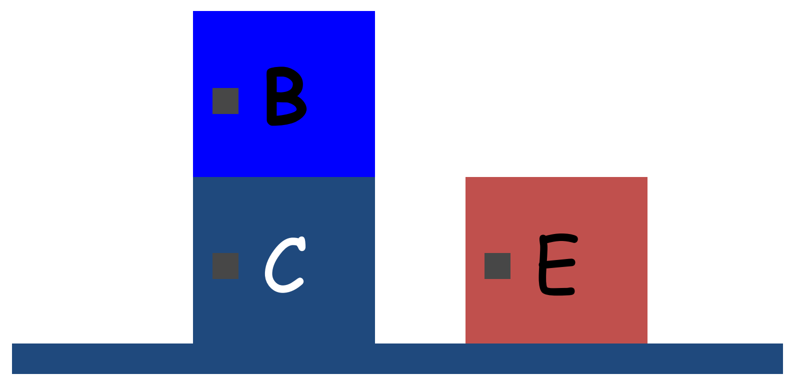
KB — many models

KB

1. on(b,c)
2. clear(e)







Models

- Let our Knowledge base KB, consist of a set of formulas.
- We say that I is a model of KB or that I satisfies KB
  - If, every formula $f \in$ KB is true under I
- We write I $\models$ KB if I satisfies KB, and I $\models f$ if f is true under I.

What’s Special About Models?

- When we write KB, we intend that the real world (i.e. our set theoretic abstraction of it) is one of its models.
- This means that every statement in KB is true in the real world.
- Note however, that not every thing true in the real world need be contained in KB. We might have only incomplete knowledge.

Models support reasoning.

- Suppose formula f is not mentioned in KB, but is true in every model of KB; i.e.,
  $$I \models KB \rightarrow I \models f.$$
- Then we say that f is a logical consequence of KB or that KB entails f .
- Since the real world is a model of KB, f must be true in the real world.
- This means that entailment is a way of finding new true facts that were not explicitly mentioned in KB.

??? If KB doesn't entail f, is f false in the real world?

Logical Consequence Example

- elephant(clyde)
  - the individual denoted by the symbol *clyde* in the set denoted by *elephant* (has the property that it is an *elephant*).

- **teacup(cup)**
  - *cup* is a teacup.

- Note that in both cases a unary predicate specifies a set of individuals. Asserting a unary predicate to be true of a term means that the individual denoted by that term is in the specified set.

Logical Consequence Example

- $\forall X,Y.elephant(X) \wedge teacup(Y) \rightarrow largerThan(X,Y)$

  - For all pairs of individuals if the first is an elephant and the second is a teacup, then the pair of objects are related to each other by the *largerThan* relation.

  - For pairs of individuals who are not elephants and teacups, the formula is immediately true.

Logical Consequence Example

- $\forall X,Y.largerThan(X,Y) \rightarrow \neg fitsIn(X,Y)$

  - For all pairs of individuals if X is larger than Y (the pair is in the largerThan relation) then we cannot have that X fits in Y (the pair cannot be in the fitsIn relation).

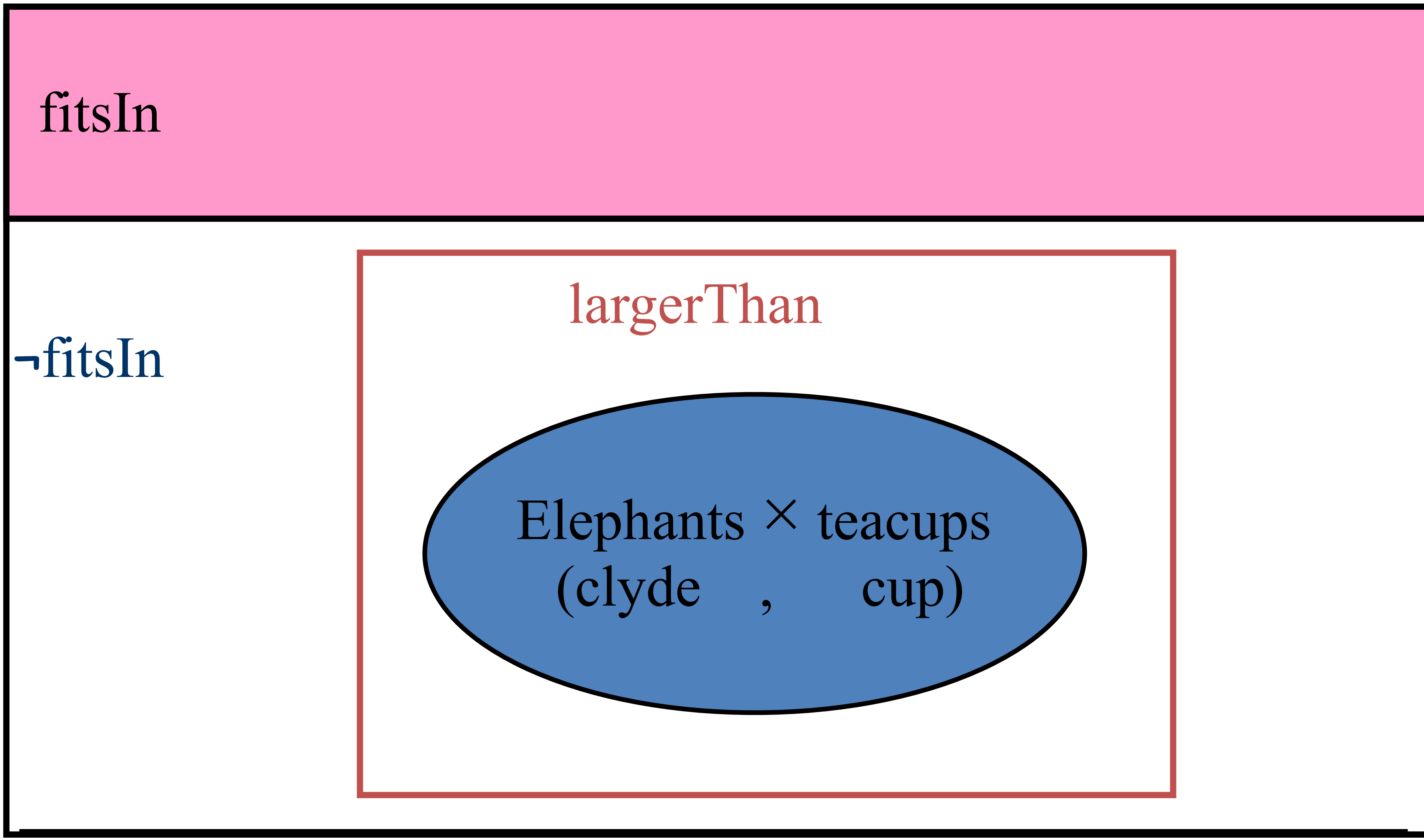
  - (The relation largerThan has an empty intersection with the fitsIn relation).

Logical Consequences

- ¬fitsIn(clyde,cup)

- We know largerThan(clyde,teacup) from the first implication. Thus we know this from the second implication.

Logical Consequences



Logical Consequence Example

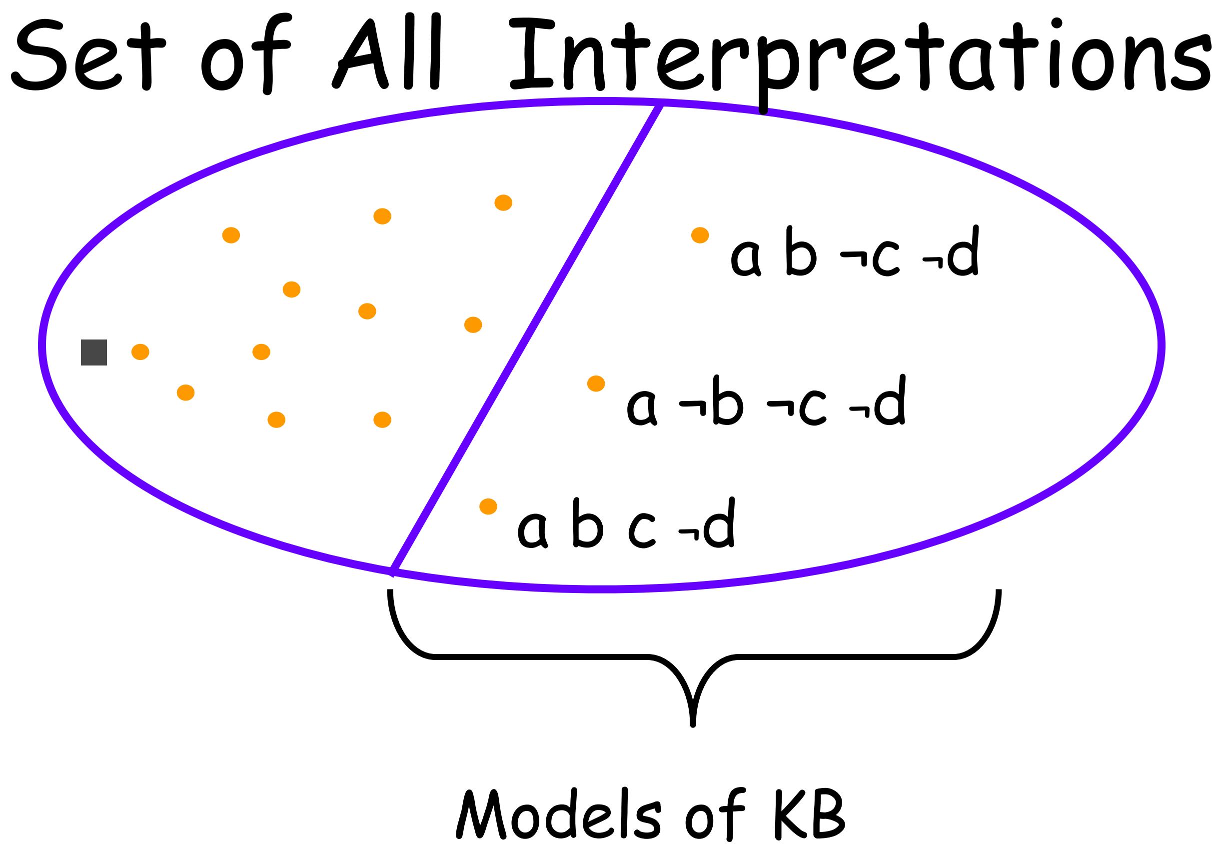
- If an interpretation satisfies KB, then the set of pairs *elephant* X *teacup* must be a subset of *largerThan*, which is disjoint from *fitsIn*.

- Therefore, the pair (*clyde,cup)* must be in the complement of the set *fitsIn*.

- Hence, ¬fitsIn(clyde,cup) must be true in every interpretation that satisfies KB.

- ¬fitsIn(clyde,cup) is a logical consequence of KB.

Models Graphically



a, b, c, and d are atomic formulas

Consequences? a, c → b, b → c, d → b, ¬b → ¬c

Models and Interpretations

- the more sentences in KB, 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"! Because Each sentence in KB rules out certain unintended interpretations.

- This is called axiomatizing the domain

Computing logical consequences

- We want procedures for computing logical consequences that can be implemented in our programs.

- This would allow us to reason with our knowledge
  - Represent the knowledge as logical formulas

  - Apply procedures for generating logical consequences

- These procedures are called proof procedures.

Proof Procedures

- Interesting, proof procedures work by simply manipulating formulas. They do not know or care anything about interpretations.
- Nevertheless they respect the semantics of interpretations!
- We will develop a proof procedure for first-order logic called resolution.
  - Resolution is the mechanism used by PROLOG

Properties of Proof Procedures

- Before presenting the details of resolution, we want to look at properties we would like to have in a (any) proof procedure.

- We write $KB \vdash f$ to indicate that f can be proved from KB (the proof procedure used is implicit).

Properties of Proof Procedures

- Soundness
  - $KB \vdash f \rightarrow KB \models f$

  i.e all conclusions arrived at via the proof procedure are correct: they are logical consequences.

- Completeness
  - $KB \models f \rightarrow KB \vdash f$

  i.e. every logical consequence can be generated by the proof procedure.

- Note proof procedures for FOL have very high complexity in the worst case. So completeness is not necessarily achievable in practice.