

Knowledge Representation (KR)

- This material is covered in chapters 7—10 (R&N, 2nd ed) and chapters 7– 9 and 12 (R&N, 3rd ed).
- Chapter 7 provides 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 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 10 (2nd ed) (12 in 3rd ed.) covers some of the additional notions that have to be dealt with when using knowledge representation in AI.

What is KNOWLEDGE?

Easier question: how do we talk about it?

- We say “John knows that ...” and fill in the blank
(can be true/false, right/wrong)
- Contrast: “John fears that ...”
(same content, different attitude)

Other forms of knowledge:

- know how, who, what, when, ...
- sensorimotor: typing, riding a bicycle
- affective: deep understanding

Belief: not necessarily true and/or held for appropriate reasons

Here we make no distinction between knowledge and belief

MAIN IDEA

Taking the world to be one way and not another

What is REPRESENTATION?

Symbols standing for things in the world



→ first aid



→ women

“John” → John

“John loves Mary” → the proposition that John loves Mary

Knowledge Representation:

Symbolic encoding of propositions believed
(by some agent)

What is REASONING?

Reasoning:

Manipulation of symbols encoding propositions to
produce new representations of new propositions

Analogy: arithmetic “1011” + “10” → “1101”

↓	↓	↓
eleven	two	thirteen

All men are mortals, socrates is a man. Thus, socrates is mortal

All foobars are bliffs, bebong is a foobar. Thus, bebong is a bliff

WHY Knowledge?

Taking an intentional stance (Daniel Dennet):

For sufficiently complex systems, it is sometimes compelling to describe systems in terms of beliefs, goals, fears, intentions

E.g., in a game-playing program

“because it believed its queen was in danger, but wanted to still control the centre of the board”

More useful than description about actual techniques used for deciding how to move

“because evaluation procedure P using minimax returned a value of +7 for this position.”

Is KR just a convenient way of talking about complex systems?

- sometimes such anthropomorphizing is inappropriate (e.g., thermostat)
- can also be very misleading!

fooling users into thinking a system knows more than it does

WHY Representation?

Taking Dennet's intentional stance says nothing about what is or is not represented symbolically

e.g., in game playing, perhaps the board position is represented but the goal of getting a knight out early is not

KR Hypothesis (Brian Smith)

“Any mechanically embodied intelligent process will be comprised of structural ingredients that a) we as external observers naturally take to represent a propositional account of the knowledge that the overall process exhibits, and b) independent of such external semantic attribution, play a formal but causal and essential role in engendering the behaviour that manifests that knowledge”

Two issues: existence of structures that

- we can interpret propositionally
- determine how the system behaves

KNOWLEDGE-BASED SYSTEM: one designed this way!

Two Examples

Both of the following systems can be described intentionally.

However, only the 2nd has a separate collection of symbolic structures (following the KR Hypothesis)

Example 2 has a knowledge base and thus the system is a knowledge-based system.

Example 1:

```
printColour(snow) :- !, write(``It's white.'').
printColour(grass) :- !, write(``It's green. '').
printColour(sky) :- !, write(``It's yellow. '').
printColour(X) :- !, write(``Beats me. '').
```

Example 2:

```
printColour(X) :- colour(S,Y),!,
~~~~~`write(``It's''), write(Y), write (``. '')
printColour(X) :- !, write(``Beats me. '').
colour(snow,white).
colour(sky,yellow).
colour(X,Y) :- madeof(X,Z), colour(Z,Y).
madeof(grass,vegetation).
colour(vegetation,green).
```

KR and AI

Much of AI involves building systems that are knowledge based

Ability derives in part from reasoning over explicitly represented knowledge

- language understanding,
- planning,
- diagnosis, etc.

Some to a certain extent

game-playing, vision, etc.

Some to a lesser extent

speech, motor control, etc.

Current research question:

how much of intelligent behaviour is knowledge-based?

Why bother?

Why not “compile out” knowledge into specialized procedures?

- distribute KB to procedures that need it (as in Example 1)
- almost always achieves better performance

No need to think. Just do it!

- riding a bike
- driving a car
- playing chess?
- doing math?
- staying alive?

Skills (Hubert Dreyfus)

- novices think, experts *react*
- Compare to “expert systems” which are knowledge based.

Advantage

Knowledge-based system most suitable for open-ended tasks
can structurally isolate *reasons* for particular behaviour

Good for

- explanation and justification
 - “Because grass is a form of vegetation”
- informality: debugging the KB
 - “No the sky is not yellow. It’s blue.”
- extensibility: new relations (*add new info*)
 - “Canaries are yellow”
- extensibility: new applications (*use your KB for different tasks*)
 - Return a list of all the white things
 - Painting pictures

Cognitive Penetrability

Hallmark of knowledge-based system:

The ability to be *told* facts about the world and adjust our behaviour correspondingly E.g.,

read a book about canaries learn they're yellow and add it to the KB

Cognitive penetrability (Zenon Pylyshyn)

Actions that are conditioned by what is currently believed

An example:

we normally leave the room if we hear a fire alarm

we do not leave the room on hearing a fire alarm if we believe that the alarm is being tested/tampered

- *can come to this belief in very many ways*

so this action is cognitively penetrable

A non-example:

blinking reflex

Why reasoning?

Want knowledge to affect action

not do action A if sentence P is in KB

but do action A if world believed in satisfies P

Difference:

P may not be *explicitly* represented

Need to apply what is known in general to the particulars of the situation

Example:

“patient x is allergic to medication m .”

“Anybody allergic to medication m is also allergic to n .”

Is it OK to prescribe n for x ?

Usually need more than just DB-style retrieval of facts in the KB!

Using Logic

No universal language/semantics

- Why not English?
- Different tasks/worlds
- Different ways to carve up the world

No universal reasoning scheme

- Geared to language
- Sometimes want “extralogical” reasoning

Start with first-order predicate calculus (FOL)

- Invented by philosopher Frege for the formalization of mathematics

Why Knowledge Representation?

An Example

An Argument for Logic

Why Knowledge Representation? An Example

- 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: Mother sends them to “seek their fortune”



■ 1st pig builds a house of straw



■ 2nd pig builds a house of sticks



■ 3rd pig builds a house of bricks

Example.

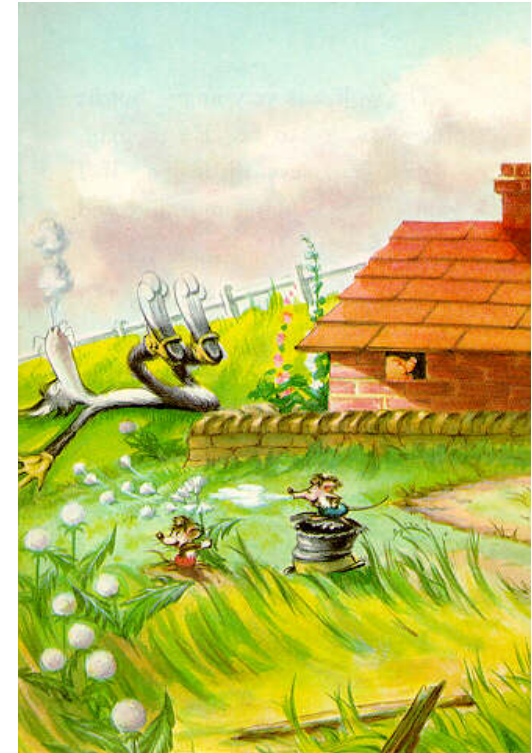
- Three little pigs



■ Wolf blows down the straw house, and eats the pig!



■ Wolf blows down the sticks house, and eats the pig!



■ Wolf cannot huff and poof the brick house!

Example

- Why couldn't the wolf blow down the house made of bricks?
- What background knowledge are we applying to come to that conclusion?
 - Brick structures are stronger than straw and stick structures.
 - Objects, like the wolf, have physical limitations. The wolf can only blow so hard.

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 (achieved by manipulating sentences).

The Knowledge Base

- The Knowledge Base is a set of sentences.
 - Syntactically well-formed
 - Semantically meaningful
- A user can perform two actions to the KB:
 - ▶ Tell the KB a new fact
 - ▶ Ask the KB a question

Syntax of Sentences

English acceptable an one is sentence This.

vs.

This English sentence is an acceptable one.

$$\forall P \neg \wedge Q R$$

vs.

$$P \vee \neg Q \wedge R$$

Semantics of Sentences

This hungry classroom is a jobless moon.

- Why is this syntactically correct sentence not meaningful?

$P \vee (\neg Q \wedge R)$

- Represents a world where either P is true, or Q is not true and R is true.

Entailments

$$\alpha \models \beta$$

- read as “ α entails β ”, or “ β follows logically from α ”
- meaning that in any world in which α is true, β is true as well.

For example

$$(P \wedge Q) \models (P \vee R)$$

Syntactical Derivation

$$\alpha \vdash \beta$$

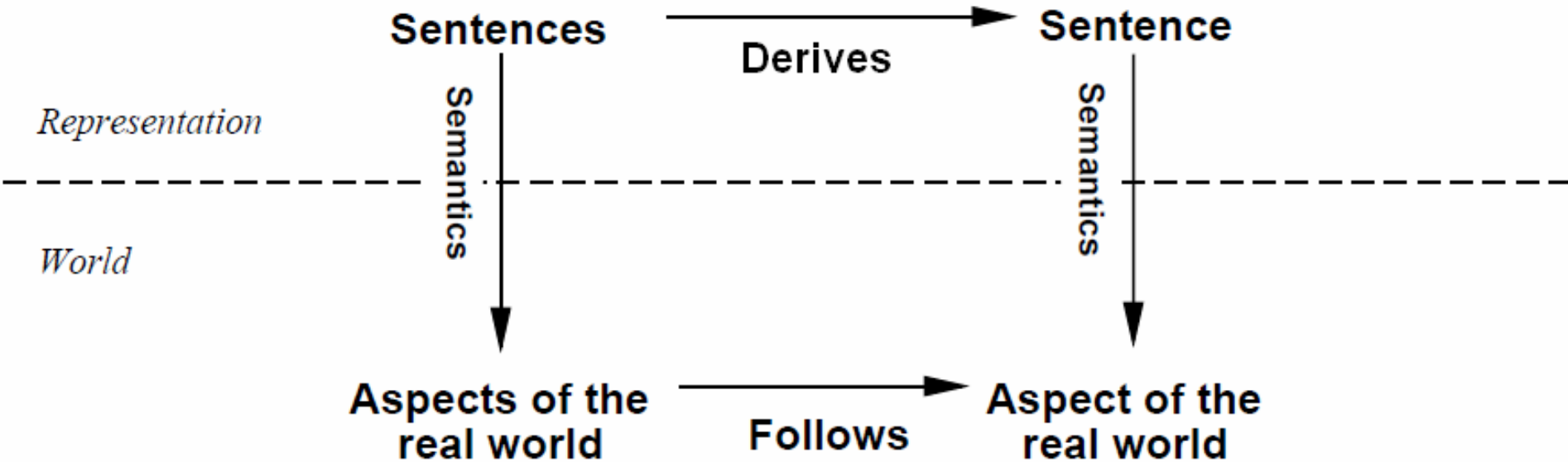
- read as “ α derives β ”
- meaning “from sentence α , following the syntactical derivation rules, we can obtain sentence β ”

For example

$$\neg(A \vee B) \vdash (\neg A \wedge \neg B)$$

KNOWLEDGE REPRESENTATION AND REASONING

The Reasoning Process



Desired Properties of Reasoning

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.

First-Order Logic (FOL)

- Whereas propositional logic assumes the world contains **facts**,
- first-order logic (like natural language) assumes the world contains
 - **Objects**: people, houses, numbers, colors, baseball games, wars, ...
 - **Relations**: red, round, prime, brother of, bigger than, part of, comes between, ...
 - **Functions**: father of, best friend, one more than, plus, ...

First-Order Logic

Syntax: A grammar specifying what are legal syntactic constructs of the representation.

Semantics: A formal mapping from syntactic constructs to set theoretic assertions.

Syntax of FOL: Basic elements

- Constants sheila, 2, blockA,...
- Predicates colour, on, brotherOf, >,...
- Functions sqrt, leftLegOf,...
- Variables x, y, a, b,...
- Connectives \neg , \Rightarrow , \wedge , \vee , \Leftrightarrow
- Equality =
- Quantifiers \forall , \exists

Syntax of Propositional Logic: Basic elements

- ~~Constants~~ sheila, 2, blockA,...
- Propositions a-ison-b, b-ison-c, ...
- ~~Functions~~ sqrt, leftLegOf,...
- ~~Variables~~ x, y, a, b,...
- Connectives $\neg, \Rightarrow, \wedge, \vee, \Leftrightarrow$
- ~~Equality~~ =
- ~~Quantifiers~~ \forall, \exists

First Order Syntax

Start with a set of primitive symbols.

- *constant* symbols.
- *function* symbols.
- *predicate* symbols (for predicates and relations).
- *variables*.

Each function and predicate symbol has a specific arity
(determines the number of arguments it takes).

First Order Syntax—Building up.

A *term* is either:

- a variable
- a constant
- an expression of the form $f(t_1, \dots t_k)$ where
 - (a) f is a function symbol;
 - (b) k is its arity;
 - (c) each t_i is a term

First Order Syntax—Building up.

An *atom* is

An expression of the form $p(t_1, \dots t_k)$ where

- p is a predicate symbol;
- k is its arity;
- each t_i is a term

Note:

- constants are functions taking zero arguments.
- Use UPPER CASE for variables, lower case for function/constant/predicate symbols.

Semantic Intuition (formalized later)

Terms denote individuals:

- constants denote specific individuals;
- functions map tuples of individuals to other individuals
 - bill, jane, father(jane), father(father(jane))
 - **X**, father(**X**), hotel7, rating(hotel7), cost(hotel7)

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)

First Order Syntax—Building up.

Atoms are formulas. (Atomic formulas).

- The negation (**NOT**) of a formula is a new formula

$$\neg f$$

Asserts that f is false.

- The conjunction (**AND**) of a set of formulas is a formula.

$$f_1 \wedge f_2 \wedge \dots \wedge f_n \text{ where each } f_i \text{ is formula}$$

Asserts that each formula f_i is true.

First Order Syntax—Building up.

- The disjunction (OR) of a set of formulas is a formula.

$f_1 \vee f_2 \vee \dots \vee f_n$ where each f_i is formula

Asserts that at least one formula f_i is true.

- Existential Quantification \exists .

$\exists X. f$ where X is a variable and f is a formula.

Asserts there is some individual such that f under than binding will be true.

- Universal Quantification \forall .

$\forall X. f$ where X is a variable and f is a formula.

Assets that f is true for every individual.

First Order Syntax—abbreviations.

- Implication (\rightarrow):
 $f1 \rightarrow f2$ is equivalent to $\neg f1 \vee f2$.

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
 - $\forall X, Y. \text{elephant}(X) \wedge \text{teacup}(Y) \rightarrow \text{largerThan}(X, Y)$
 - $\forall X, Y. \text{teacup}(Y) \wedge \text{elephant}(X) \rightarrow \text{largerThan}(X, Y)$
- To capture this equivalence and to make sense of complex formulas we utilize the semantics.

Semantics.

- A formal mapping from formulas to semantic entities (individuals, sets and relations over individuals, functions over individuals).
- The mapping mirrors the recursive structure of the syntax, so we can give any formula, no matter how complex a mapping to semantic entities.

Semantics—Formal Details

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 symbols.
 - Each symbol p in P has a particular arity.
- V = an infinite set of variables.

Semantics—Formal Details

An **interpretation** (model) is a tuple

$\langle D, \Phi, \Psi, v \rangle$

- D is a non-empty set (domain of individuals)
- Φ is a mapping: $\Phi(f) \rightarrow (D^k \rightarrow D)$
 - maps k -ary **function** symbol f , to a function from k -ary tuples of individuals to individuals.
- Ψ is a mapping: $\Psi(p) \rightarrow (D^k \rightarrow \text{True/False})$
 - maps k -ary **predicate** symbol p , to an indicator function over k -ary tuples of individuals (a subset of D^k)
- v is a **variable** assignment function. $v(X) = d \in D$ (it maps every variable to some individual)

Intuitions: Domain of Discourse

- Domain D : $d \in D$ is an *individual*

E.g., $\{ \underline{craig}, \underline{jane}, \underline{grandhotel}, \underline{le-fleabag}, \underline{rome}, \underline{portofino}, \underline{100}, \underline{110}, \underline{120} \dots \}$

- Underlined symbols denote domain individuals (as opposed to symbols of the first-order language)
- Domains often infinite, but we'll use finite models to prime our intuitions

Intuitions: Φ (individual denoted by fn.)

$$\Phi(f) \rightarrow (D^k \rightarrow D)$$

Given k -ary function f , k individuals,

what individual does $f(d_1, \dots, d_k)$ denote

- 0-ary functions (constants) are mapped to specific individuals in D .
 - $\Phi(\text{client17}) = \underline{\text{craig}}$, $\Phi(\text{hotel5}) = \underline{\text{le-fleabag}}$, $\Phi(\text{rome}) = \underline{\text{rome}}$
- 1-ary functions are mapped to functions in $D \rightarrow D$
 - $\Phi(\text{minquality}) = f_{\text{minquality}}$:
 $f_{\text{minquality}}(\underline{\text{craig}}) = \underline{\text{3stars}}$
 - $\Phi(\text{rating}) = f_{\text{rating}}$:
 $f_{\text{rating}}(\underline{\text{grandhotel}}) = \underline{\text{5stars}}$
- 2-ary functions are mapped to functions from $D^2 \rightarrow D$
 - $\Phi(\text{distance}) = f_{\text{distance}}$:
 $f_{\text{distance}}(\underline{\text{toronto}}, \underline{\text{sienna}}) = \underline{\text{3256}}$
- n -ary functions are mapped similarly.

Intuitions: Ψ (truth or falsity of formula)

$\Psi(p) \rightarrow (D^k \rightarrow \text{True/False})$

Given k -ary **predicate**, k individuals,

does the relation denoted by p hold of these?

$\Psi(p)(\langle d1, \dots dk \rangle) = \text{true?}$

- 0-ary predicates are mapped to true or false.

$\Psi(\text{rainy}) = \text{True} \quad \Psi(\text{sunny}) = \text{False}$

- 1-ary predicates are mapped indicator functions of subsets of D .

- $\Psi(\text{satisfied}) = p_{\text{satisfied}}$:

$p_{\text{satisfied}}(\text{craig}) = \text{True}$

- $\Psi(\text{privatebeach}) = p_{\text{privatebeach}}$:

$p_{\text{privatebeach}}(\text{le-fleabag}) = \text{False}$

Intuitions: Ψ (truth or falsity of formula)

- 2-ary predicates are mapped to indicator functions over D^2
 - $\Psi(location) = p_location$: $p_location(\underline{grandhotel}, \underline{rome}) = True$
 $p_location(\underline{grandhotel}, \underline{sienna}) = False$
 - $\Psi(available) = p_available$:
 $p_available(\underline{grandhotel}, \underline{week29}) = True$
- n-ary predicates..

Intuitions: v

v exists to take care of quantification.

As we will see the exact mapping it specifies will not matter.

Notation: $v[X/d]$ is a **new** variable assignment function.

- Exactly like v , except that it maps the variable X to the individual d .
- Maps every other variable exactly like v :
$$v'(Y) = v[X/d](Y)$$

Semantics—Building up

Given language $L(F,P,V)$, and an interpretation $I = \langle D, \Phi, \Psi, v \rangle$

- a) Constant c (0-ary function) **denotes** an individual
 $I(c) = \Phi(c) \in D$
- b) Variable X denotes an individual
 $I(X) = v(X) \in D$ (variable assignment function).
- c) term $t = f(t_1, \dots, t_k)$ **denotes** an individual
 $I(t) = \Phi(f)(I(t_1), \dots, I(t_k)) \in D$

We recursively find the denotation of each term, then we apply the function denoted by f to get a new individual.

Hence terms always denote individuals under an interpretation I

Semantics—Building up

Formulas

a) atom $a = p(t_1, \dots, t_k)$ has truth value

$$I(a) = \Psi(p)(I(t_1), \dots, I(t_k)) \in \{ \text{True}, \text{False} \}$$

We recursively find the individuals denoted by the t_i , then we check to see if this tuple of individuals is in the relation denoted by p .

Semantics—Building up

Formulas

b) Negated formulas $\neg f$ has truth value

$I(\neg f) = \text{True}$ if $I(f) = \text{False}$

$I(\neg f) = \text{False}$ if $I(f) = \text{True}$

c) And formulas $f_1 \wedge f_2 \wedge \dots \wedge f_n$ have truth value

$I(f_1 \wedge f_2 \wedge \dots \wedge f_n) = \text{True}$ if every $I(f_i) = \text{True}$.

$I(f_1 \wedge f_2 \wedge \dots \wedge f_n) = \text{False}$ otherwise.

d) Or formulas $f_1 \vee f_2 \vee \dots \vee f_n$ have truth value

$I(f_1 \vee f_2 \vee \dots \vee f_n) = \text{True}$ if any $I(f_i) = \text{True}$.

$I(f_1 \vee f_2 \vee \dots \vee f_n) = \text{False}$ otherwise.

Semantics—Building up

Formulas

e) Existential formulas $\exists X. f$ have truth value

$I(\exists X. f) = \text{True}$ if there exists a $d \in D$ such that

$I'(f) = \text{True}$

where $I' = \langle D, \Phi, \Psi, v[X/d] \rangle$

False otherwise.

I' is just like I except that its variable assignment function now maps X to d . “ d ” is the individual of which “ f ” is true.

Semantics—Building up

Formulas

- f) Universal formulas $\forall X.f$ have truth value
 $I(\forall X.f) = \text{True}$ if for all $d \in D$

$$I'(f) = \text{True}$$

where $I' = \langle D, \Phi, \Psi, v[X/d] \rangle$

False otherwise.

Now “f” must be true of every individual “d”.

Hence formulas are always either True or False under an interpretation I

Example

$D = \{\underline{\text{bob}}, \underline{\text{jack}}, \underline{\text{fred}}\}$

$I(\forall X.\text{happy}(X))$

1. $\Psi(\text{happy})(v[X/\underline{\text{bob}}](X)) = \Psi(\text{happy})(\underline{\text{bob}}) = \text{True}$

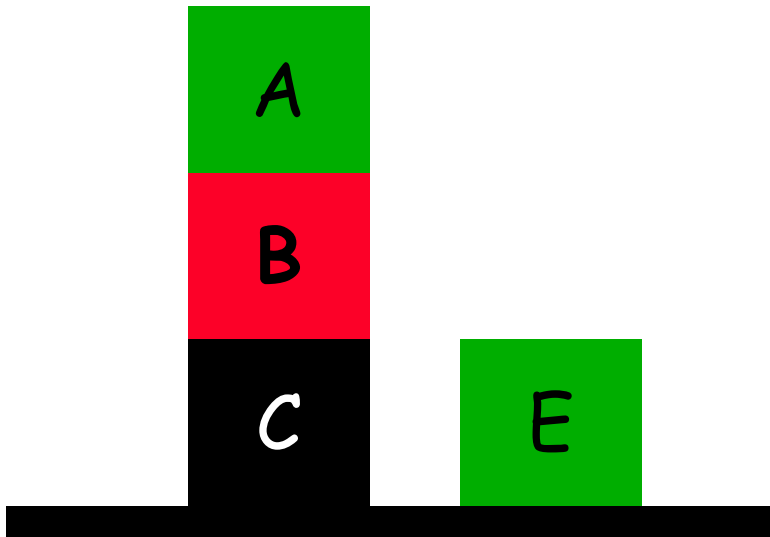
2. $\Psi(\text{happy})(v[X/\underline{\text{jack}}](X)) = \Psi(\text{happy})(\underline{\text{jack}}) = \text{True}$

3. $\Psi(\text{happy})(v[X/\underline{\text{fred}}](X)) = \Psi(\text{happy})(\underline{\text{fred}}) = \text{True}$

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

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)

Think of it as possible way
the world could be

A possible Model I_1 (semantics)

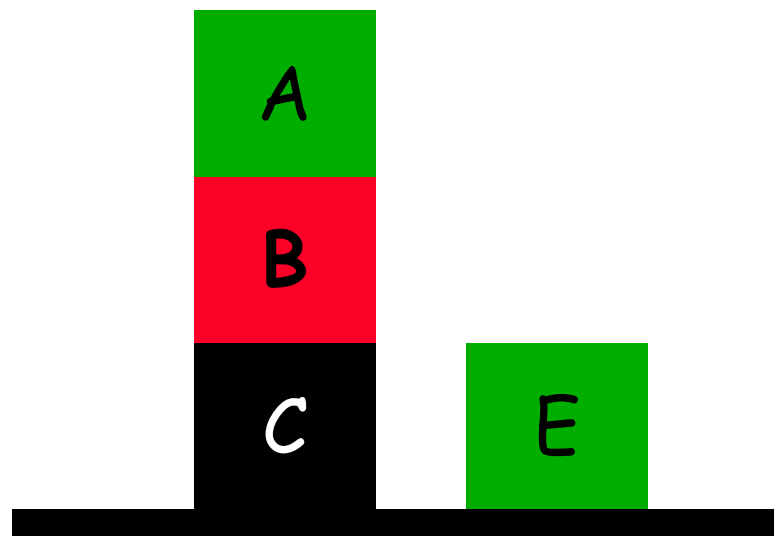
- $D = \{\underline{A}, \underline{B}, \underline{C}, \underline{E}\}$
- $\Phi(a) = \underline{A}, \Phi(b) = \underline{B},$
 $\Phi(c) = \underline{C}, \Phi(e) = \underline{E}.$
- $\Psi(\text{on}) = \{(\underline{A}, \underline{B}), (\underline{B}, \underline{C})\}$
- $\Psi(\text{above}) =$
 $\{(\underline{A}, \underline{B}), (\underline{B}, \underline{C}), (\underline{A}, \underline{C})\}$
- $\Psi(\text{clear}) = \{\underline{A}, \underline{E}\}$
- $\Psi(\text{ontable}) = \{\underline{C}, \underline{E}\}$

Models—Examples.

Model I_1

- $D = \{\underline{A}, \underline{B}, \underline{C}, \underline{E}\}$
- $\Phi(a) = \underline{A}, \Phi(b) = \underline{B},$
 $\Phi(c) = \underline{C}, \Phi(e) = \underline{E}.$
- $\Psi(\text{on}) = \{(\underline{A}, \underline{B}), (\underline{B}, \underline{C})\}$
- $\Psi(\text{above}) =$
 $\{(\underline{A}, \underline{B}), (\underline{B}, \underline{C}), (\underline{A}, \underline{C})\}$
- $\Psi(\text{clear}) = \{\underline{A}, \underline{E}\}$
- $\Psi(\text{ontable}) = \{\underline{C}, \underline{E}\}$

Environment



Aside on Notation.

Model I_1

- $D = \{\underline{A}, \underline{B}, \underline{C}, \underline{E}\}$
- $\Phi(a) = \underline{A}, \Phi(b) = \underline{B},$
 $\Phi(c) = \underline{C}, \Phi(e) = \underline{E}.$
- $\Psi(\text{on}) = \{(\underline{A}, \underline{B}), (\underline{B}, \underline{C})\}$
- $\Psi(\text{above}) =$
 $\{(\underline{A}, \underline{B}), (\underline{B}, \underline{C}), (\underline{A}, \underline{C})\}$
- $\Psi(\text{clear}) = \{\underline{A}, \underline{E}\}$
- $\Psi(\text{ontable}) = \{\underline{C}, \underline{E}\}$

Comment on notation:

To this point we have represented an interpretation as a tuple, $\langle D, \Phi, \Psi, v \rangle$. It is also common practice to abbreviate this and refer to an interpretation function as σ and to denote the application of the interpretation function by a superscripting of σ .

E.g.,

$a^\sigma = A, b^\sigma = B, \text{ etc.}$

$\text{on}^\sigma = \{(\underline{A}, \underline{B}), (\underline{B}, \underline{C})\},$

$\text{above}^\sigma = \{(\underline{A}, \underline{B}), (\underline{B}, \underline{C}), (\underline{A}, \underline{C})\}$

$\text{clear}^\sigma = \{\underline{A}, \underline{E}\}$

Models—Formulas true or false?

Model I_1

- $D = \{\underline{A}, \underline{B}, \underline{C}, \underline{E}\}$
- $\Phi(a) = \underline{A}, \Phi(b) = \underline{B},$
 $\Phi(c) = \underline{C}, \Phi(e) = \underline{E}.$
- $\Psi(\text{on}) = \{(\underline{A}, \underline{B}), (\underline{B}, \underline{C})\}$
- $\Psi(\text{above}) =$
 $\{(\underline{A}, \underline{B}), (\underline{B}, \underline{C}), (\underline{A}, \underline{C})\}$
- $\Psi(\text{clear}) = \{\underline{A}, \underline{E}\}$
- $\Psi(\text{ontable}) = \{\underline{C}, \underline{E}\}$

$\forall X, Y. \text{on}(X, Y) \rightarrow \text{above}(X, Y)$

✓ $X = \underline{A}, Y = \underline{B}$

✓ $X = \underline{C}, Y = \underline{A}$

✓ ...

$\forall X, Y. \text{above}(X, Y) \rightarrow \text{on}(X, Y)$

✓ $X = \underline{A}, Y = \underline{B}$

✗ $X = \underline{A}, Y = \underline{C}$

Models—Examples.

Model I_1

- $D = \{\underline{A}, \underline{B}, \underline{C}, \underline{E}\}$
- $\Phi(a) = \underline{A}, \Phi(b) = \underline{B},$
 $\Phi(c) = \underline{C}, \Phi(e) = \underline{E}.$
- $\Psi(\text{on}) = \{(\underline{A}, \underline{B}), (\underline{B}, \underline{C})\}$
- $\Psi(\text{above}) =$
 $\{(\underline{A}, \underline{B}), (\underline{B}, \underline{C}), (\underline{A}, \underline{C})\}$
- $\Psi(\text{clear}) = \{\underline{A}, \underline{E}\}$
- $\Psi(\text{ontable}) = \{\underline{C}, \underline{E}\}$

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

✓ $X = \underline{A}$

✓ $X = \underline{C}, Y = \underline{B}$

✓ ...

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

✗ $Y = \underline{A} ?$ No! ($X = \underline{C}$)

✗ $Y = \underline{C} ?$ No! ($X = \underline{B}$)

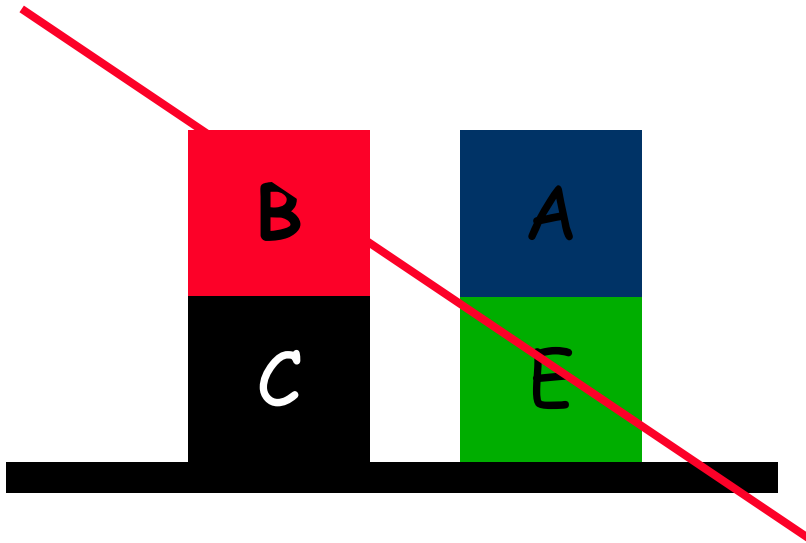
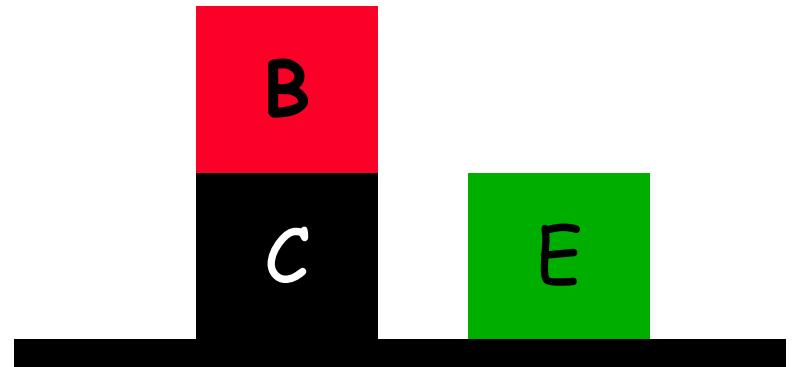
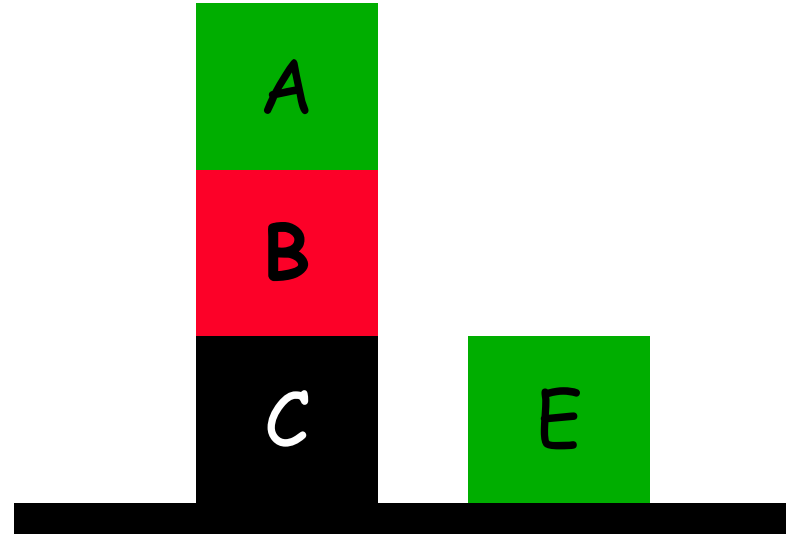
✗ $Y = \underline{E} ?$ No! ($X = \underline{B}$)

✗ $Y = \underline{B} ?$ No! ($X = \underline{B}$)

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 $f \in \text{KB}$ is true under I
- We write $I \models \text{KB}$ if I satisfies KB, and $I \models f$ if f is true under I .

What's Special About Models?

- When we write KB, we have an intended interpretation for it – a way that we think the real world will be.
- 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.,

For all I , $I \models \text{KB} \rightarrow I \models f$.

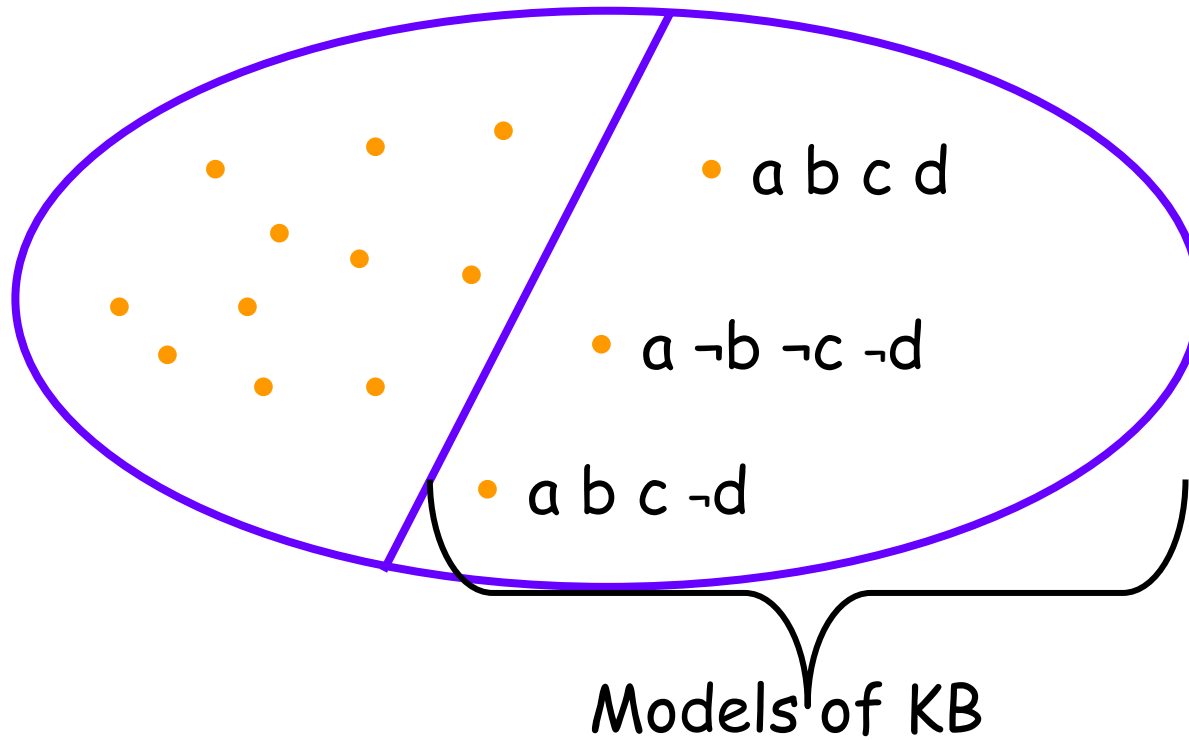
- Then we say that f is a **logical consequence** of KB or that KB **entails** f
 $\text{KB} \models f$.
- Since the real world is a model of KB, f must be true in the real world.
- This means that entailment is a way of finding new true facts that were not explicitly mentioned in KB.

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

Models Graphically (propositional example)

Propositional KB: $a, c \rightarrow b, b \rightarrow c, d \rightarrow b, \neg b \rightarrow \neg c$

Set of All Interpretations



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 are computable, but they might have very high complexity in the worst case. So completeness is not necessarily achievable in practice.

Resolution

- **Clausal form.**
 - Resolution works with formulas expressed in clausal form.
 - A **literal** is an atomic formula or the negation of an atomic formula. $\text{dog}(\text{fido})$, $\neg \text{cat}(\text{fido})$
 - A **clause** is a disjunction of literals:
 - $\neg \text{owns}(\text{fido}, \text{fred}) \vee \neg \text{dog}(\text{fido}) \vee \text{person}(\text{fred})$
 - We write
 $(\neg \text{owns}(\text{fido}, \text{fred}), \neg \text{dog}(\text{fido}), \text{person}(\text{fred}))$
 - A **clausal theory** is a conjunction of clauses.

Resolution

- Prolog Programs

- Prolog programs are clausal theories.
- However, each clause in a Prolog program is **Horn**.
- A horn clause contains at most one positive literal.

- The horn clause

$$\neg q_1 \vee \neg q_2 \vee \dots \vee \neg q_n \vee p$$

is equivalent to

$$q_1 \wedge q_2 \wedge \dots \wedge q_n \Rightarrow p$$

and is written as the following rule in Prolog:

$$p \text{ :- } q_1, q_2, \dots, q_n$$

(However Prolog also uses negation as failure!)

Resolution Rule for Ground Clauses

- The resolution proof procedure consists of only **one simple rule**:
 - From the two clauses
 - (P , Q_1 , Q_2 , ..., Q_k)
 - ($\neg P$, R_1 , R_2 , ..., R_n)
 - We infer the new clause
 - (Q_1 , Q_2 , ..., Q_k , R_1 , R_2 , ..., R_n)
 - Example:
 - ($\neg \text{largerThan}(\text{clyde}, \text{cup})$, $\neg \text{fitsIn}(\text{clyde}, \text{cup})$)
 - ($\text{fitsIn}(\text{clyde}, \text{cup})$)
- Infer
- $\neg \text{largerThan}(\text{clyde}, \text{cup})$

Resolution Proof: Forward chaining

- Logical consequences can be generated from the resolution rule in two ways:

1. Forward Chaining inference (“Consequence Finding”)
 - If we have a sequence of clauses C_1, C_2, \dots, C_k
 - Such that each C_i is either in KB or is the result of a resolution step involving two prior clauses in the sequence.
 - We then have that $KB \vdash C_k$.

Forward chaining is sound so we also have $KB \models C_k$

Resolution Proof: Refutation proofs

2. Refutation proofs.

- We determine if $KB \vdash f$ by showing that a **contradiction** can be generated from $KB \wedge \neg f$.
- In this case a contradiction is an **empty** clause $()$.
- We employ resolution to construct a sequence of clauses C_1, C_2, \dots, C_m such that
 - C_i is in $KB \wedge \neg f$, or is the result of resolving two previous clauses in the sequence.
 - $C_m = ()$ i.e. its the empty clause.

Resolution Proof: Refutation proofs

- If we can find such a sequence $C_1, C_2, \dots, C_m = ()$, we have that
 - $KB \vdash f$.
 - Furthermore, this procedure is sound so
 - $KB \models f$
- And the procedure is also complete so it is capable of finding a proof of any f that is a logical consequence of KB . I.e.
 - If $KB \models f$ then we can generate a refutation from $KB \wedge \neg f$

Resolution Proofs Example

Want to prove $\text{likes}(\text{clyde}, \text{peanuts})$ from:

1. $(\text{elephant}(\text{clyde}), \text{giraffe}(\text{clyde}))$
2. $(\neg \text{elephant}(\text{clyde}), \text{likes}(\text{clyde}, \text{peanuts}))$
3. $(\neg \text{giraffe}(\text{clyde}), \text{likes}(\text{clyde}, \text{leaves}))$
4. $\neg \text{likes}(\text{clyde}, \text{leaves})$

Approach 1: Forward Chaining Proof:

- $3 \& 4 \rightarrow \neg \text{giraffe}(\text{clyde})$ [5.]
- $5 \& 1 \rightarrow \text{elephant}(\text{clyde})$ [6.]
- $6 \& 2 \rightarrow \text{likes}(\text{clyde}, \text{peanuts})$ [7.] ✓

Resolution Proofs Example

1. (elephant(clyde), giraffe(clyde))
2. (\neg elephant(clyde), likes(clyde,peanuts))
3. (\neg giraffe(clyde), likes(clyde,leaves))
4. \neg likes(clyde,leaves)

Approach 2: Refutation Proof:

- \neg likes(clyde,peanuts) [5.]
- 5&2 \rightarrow \neg elephant(clyde) [6.]
- 6&1 \rightarrow giraffe(clyde) [7.]
- 7&3 \rightarrow likes(clyde,leaves) [8.]
- 8&4 \rightarrow () ✓

Resolution Proofs

- Proofs by refutation have the advantage that they are easier to find.
 - They are more focused to the particular conclusion we are trying to reach.
- To develop a complete resolution proof procedure for First-Order logic we need :
 1. A way of converting KB and f (the query) into clausal form.
 2. A way of doing resolution even when we have variables (unification).

Conversion to Clausal Form

To convert the KB into Clausal form we perform the following 8-step procedure:

- 1. Eliminate Implications.**
- 2. Move Negations inwards (and simplify $\neg\neg$).**
- 3. Standardize Variables.**
- 4. Skolemize.**
- 5. Convert to Prenix Form.**
- 6. Distribute disjunctions over conjunctions.**
- 7. Flatten nested conjunctions and disjunctions.**
- 8. Convert to Clauses.**

C-T-C-F: Eliminate implications

We use this example to show each step:

$$\forall X. p(X) \rightarrow \left(\begin{array}{l} \forall Y. p(Y) \rightarrow p(f(X, Y)) \\ \wedge \neg(\forall Y. \neg q(X, Y) \wedge p(Y)) \end{array} \right)$$

1. Eliminate implications: $A \rightarrow B \rightarrow \neg A \vee B$

$$\forall X. \neg p(X) \vee \left(\begin{array}{l} \forall Y. \neg p(Y) \vee p(f(X, Y)) \\ \wedge \neg(\forall Y. \neg q(X, Y) \wedge p(Y)) \end{array} \right)$$

C-T-C-F: Move \neg Inwards

$$\begin{aligned} &\forall X. \neg p(X) \\ &\vee \left(\begin{aligned} &\forall Y. \neg p(Y) \vee p(f(X, Y)) \\ &\wedge \neg(\forall Y. \neg q(X, Y) \wedge p(Y)) \end{aligned} \right) \end{aligned}$$

2. Move Negations Inwards (and simplify $\neg\neg$)

$$\begin{aligned} &\forall X. \neg p(X) \\ &\vee \left(\begin{aligned} &\forall Y. \neg p(Y) \vee p(f(X, Y)) \\ &\wedge \exists Y. q(X, Y) \vee \neg p(Y) \end{aligned} \right) \end{aligned}$$

C-T-C-F: : \neg continue...

Rules for moving negations inwards

- $\neg(A \wedge B) \rightarrow \neg A \vee \neg B$
- $\neg(A \vee B) \rightarrow \neg A \wedge \neg B$
- $\neg\forall X. f \rightarrow \exists X. \neg f$
- $\neg\exists X. f \rightarrow \forall X. \neg f$
- $\neg\neg A \rightarrow A$

C-T-C-F: Standardize Variables

$$\begin{aligned} &\forall X. \neg p(X) \\ &\vee \left(\forall Y. \neg p(Y) \vee p(f(X, Y)) \right. \\ &\quad \left. \wedge \exists Y. q(X, Y) \vee \neg p(Y) \right) \end{aligned}$$

3. Standardize Variables (Rename variables so that each quantified variable is unique)

$$\begin{aligned} &\forall X. \neg p(X) \\ &\vee \left(\forall Y. (\neg p(Y) \vee p(f(X, Y))) \right. \\ &\quad \left. \wedge \exists Z. q(X, Z) \vee \neg p(Z) \right) \end{aligned}$$

C-T-C-F: Skolemize

$$\begin{aligned} &\forall X. \neg p(X) \\ &\vee \left(\begin{aligned} &\forall Y. \neg p(Y) \vee p(f(X, Y)) \\ &\wedge \exists Z. q(X, Z) \vee \neg p(Z) \end{aligned} \right) \end{aligned}$$

4. Skolemize (Remove existential quantifiers by introducing new function symbols).

$$\begin{aligned} &\forall X. \neg p(X) \\ &\vee \left(\begin{aligned} &\forall Y. \neg p(Y) \vee p(f(X, Y)) \\ &\wedge q(X, g(X)) \vee \neg p(g(X)) \end{aligned} \right) \end{aligned}$$

C-T-C-F: Skolemization continue...

Consider $\exists Y.\text{elephant}(Y) \wedge \text{friendly}(Y)$

- This asserts that there is some individual (binding for Y) that is both an elephant and friendly.
- To remove the existential, we **invent** a name for this individual, say **a** . This is a new constant symbol not equal to any previous constant symbols to obtain:
 $\text{elephant}(a) \wedge \text{friendly}(a)$
- This is saying the same thing, since we do not know anything about the new constant **a** .

C-T-C-F: Skolemization continue

- It is essential that the introduced symbol “a” is **new**. Else we might know something else about “a” in KB.
- If we did know something else about “a” we would be asserting more than the existential.
- In original quantified formula we know nothing about the variable “Y”. Just what was being asserted by the existential formula.

C-T-C-F: Skolemization continue

Now consider $\forall X \exists Y. \text{loves}(X, Y).$

- This formula claims that for every X there is some Y that X loves (perhaps a different Y for each X).
- Replacing the existential by a new constant won't work
 $\forall X. \text{loves}(X, a).$

Because this asserts that there is a **particular** individual “ a ” loved by every X .

- To properly convert existential quantifiers scoped by universal quantifiers we must use **functions** not just constants.

C-T-C-F: Skolemization continue

- We must use a function that mentions **every universally quantified variable that scopes the existential**.
- In this case X scopes Y so we must replace the existential Y by a function of X

$$\forall X. \text{loves}(X, g(X)).$$

where g is a **new** function symbol.

- This formula asserts that for every X there is some individual (given by $g(X)$) that X loves. $g(X)$ can be different for each different binding of X .

C-T-C-F: Skolemization Examples

- $\forall XYZ \exists W. r(X, Y, Z, W) \rightarrow \forall XYZ. r(X, Y, Z, h1(X, Y, Z))$
- $\forall XY \exists W. r(X, Y, g(W)) \rightarrow \forall XY. r(X, Y, g(h2(X, Y)))$
- $\forall XY \exists W \forall Z. r(X, Y, W) \wedge q(Z, W)$
 $\rightarrow \forall XYZ. r(X, Y, h3(X, Y)) \wedge q(Z, h3(X, Y))$

C-T-C-F: Convert to prenex

$$\forall X. \neg p(X) \\ \vee \left(\forall Y. \neg p(Y) \vee p(f(X, Y)) \right. \\ \left. \wedge q(X, g(X)) \vee \neg p(g(X)) \right)$$

5. Convert to prenex form. (Bring all quantifiers to the front—only universals, each with different name).

$$\forall X \forall Y. \neg p(X) \\ \vee \left(\neg p(Y) \vee p(f(X, Y)) \right. \\ \left. \wedge q(X, g(X)) \vee \neg p(g(X)) \right)$$

C-T-C-F: disjunctions over conjunctions

$$\begin{aligned} &\forall X \forall Y. \neg p(X) \\ &\quad \vee (\neg p(Y) \vee p(f(X, Y)) \\ &\quad \quad \wedge q(X, g(X)) \vee \neg p(g(X))) \end{aligned}$$

6. Disjunction over Conjunction

$$A \vee (B \wedge C) \rightarrow (A \vee B) \wedge (A \vee C)$$

$$\begin{aligned} &\forall X Y. \neg p(X) \vee \neg p(Y) \vee p(f(X, Y)) \\ &\quad \wedge \neg p(X) \vee q(X, g(X)) \vee \neg p(g(X)) \end{aligned}$$

C-T-C-F: flatten & convert to clauses

7. Flatten nested conjunctions and disjunctions.

$$(A \vee (B \vee C)) \rightarrow (A \vee B \vee C)$$

8. Convert to Clauses

(remove quantifiers and break apart conjunctions).

$$\begin{aligned} \forall XY. \quad & \neg p(X) \vee \neg p(Y) \vee p(f(X,Y)) \\ & \wedge \neg p(X) \vee q(X,g(X)) \vee \neg p(g(X)) \end{aligned}$$

- a) $\neg p(X) \vee \neg p(Y) \vee p(f(X,Y))$
- b) $\neg p(X) \vee q(X,g(X)) \vee \neg p(g(X))$

Unification

- Ground clauses are clauses with no variables in them. For ground clauses we can use syntactic identity to detect when we have a P and $\neg P$ pair.
- What about variables? Can the clauses
 - $(P(\text{john}), Q(\text{fred}), R(X))$
 - $(\neg P(Y), R(\text{susan}), R(Y))$Be resolved?

Unification.

- Intuitively, once reduced to clausal form, all remaining variables are universally quantified. So, implicitly $(\neg P(Y), R(\text{susan}), R(Y))$ represents clauses like
 - $(\neg P(\text{fred}), R(\text{susan}), R(\text{fred}))$
 - $(\neg P(\text{john}), R(\text{susan}), R(\text{john}))$
 - ...
- So there is a “specialization” of this clause that can be resolved with $(P(\text{john}), Q(\text{fred}), R(X))$

Unification.

- We want to be able to match conflicting literals, even when they have variables. This matching process automatically determines whether or not there is a “specialization” that matches.
- We don't want to over specialize!

Unification.

- $(\neg p(X), s(X), q(\text{fred}))$
- $(p(Y), r(Y))$
- Possible resolvants
 - $(s(\text{john}), q(\text{fred}), r(\text{john})) \{Y=\text{john}, X=\text{john}\}$
 - $(s(\text{sally}), q(\text{fred}), r(\text{sally})) \{Y=\text{sally}, X=\text{sally}\}$
 - $(s(X), q(\text{fred}), r(X)) \quad \{Y=X\}$
- The last resolvent is “**most-general**”, the other two are specializations of it.
- We want to keep the most general clause so that we can use it future resolution steps.

Unification.

- **unification** is a mechanism for finding a “most general” matching.
- First we consider **substitutions**.
 - A substitution is a finite set of equations of the form

$$(V = t)$$

where V is a variable and t is a term not containing V .
(t might contain other variables).

Substitutions.

- We can **apply a substitution σ** to a formula f to obtain a new formula $f\sigma$ by simultaneously replacing every variable mentioned in the left hand side of the substitution by the right hand side.

$$p(X, g(Y, Z))[X=Y, Y=f(a)] \rightarrow p(Y, g(f(a), Z))$$

- Note that the substitutions are not applied sequentially, i.e., the first Y is not subsequently replaced by $f(a)$.

Substitutions.

- We can compose two substitutions. θ and σ to obtain a new substitution $\theta\sigma$.

Let $\theta = \{X_1=s_1, X_2=s_2, \dots, X_m=s_m\}$

$\sigma = \{Y_1=t_1, Y_2=t_2, \dots, Y_k=t_k\}$

To compute $\theta\sigma$

1. $S = \{X_1=s_1\sigma, X_2=s_2\sigma, \dots, X_m=s_m\sigma, Y_1=t_1, Y_2=t_2, \dots, Y_k=t_k\}$

we apply σ to each RHS of θ and then add all of the equations of σ .

Substitutions.

1. $S = \{X_1=s_1\sigma, X_2=s_2\sigma, \dots, X_m=s_m\sigma, Y_1=t_1, Y_2=t_2, \dots, Y_k=t_k\}$
2. Delete any identities, i.e., equations of the form $V=V$.
3. Delete any equation $Y_i=s_i$ where Y_i is equal to one of the X_j in θ . Why?

The final set S is the composition $\theta\sigma$.

Composition Example.

$$\theta = \{X=f(Y), Y=Z\}, \sigma = \{X=a, Y=b, Z=Y\}$$

$\theta\sigma$

Substitutions.

- The empty substitution $\varepsilon = \{\}$ is also a substitution, and it acts as an identity under composition.
- More importantly substitutions when applied to formulas are associative:

$$(f\theta)\sigma = f(\theta\sigma)$$

- Composition is simply a way of converting the sequential application of a series of substitutions to a single simultaneous substitution.

Unifiers

- A **unifier** of two formulas f and g is a substitution σ that makes f and g **syntactically identical**.
- Not all formulas can be unified—substitutions only affect variables.

$$p(f(X),a) \quad p(Y,f(w))$$

- This pair cannot be unified as there is no way of making $a = f(w)$ with a substitution.

MGU

- A substitution σ of two formulas f and g is a **Most General Unifier (MGU)** if
 1. σ is a unifier.
 2. For every other unifier θ of f and g there must exist a third substitution λ such that
$$\theta = \sigma\lambda$$
- This says that every other unifier is “more specialized than σ ”. The MGU of a pair of formulas f and g is unique up to renaming.

MGU

$$p(f(X),Z) \quad p(Y,a)$$

1. $\sigma = \{Y = f(a), X=a, Z=a\}$ is a unifier.

$$\begin{array}{l} p(f(X),Z)\sigma = \\ p(Y,a)\sigma = \end{array}$$

But it is not an MGU.

2. $\theta = \{Y=f(X), Z=a\}$ is an MGU.

$$\begin{array}{l} p(f(X),Z) \theta = \\ p(Y,a) \theta = \end{array}$$

MGU

$$p(f(X), Z) \quad p(Y, a)$$

3. $\sigma = \theta\lambda$, where $\lambda = \{X=a\}$

$$\sigma = \{Y = f(a), X=a, Z=a\}$$

$$\lambda = \{X=a\}$$

$$\text{Recall } \theta = \{Y=f(X), Z=a\}$$

$$\text{Thus } \theta\lambda =$$

MGU

- The MGU is the “least specialized” way of making clauses with universal variables match.
- We can compute MGUs.
- Intuitively we line up the two formulas and find the first sub-expression where they disagree. The pair of subexpressions where they **first** disagree is called the **disagreement set**.
- The algorithm works by successively fixing disagreement sets until the two formulas become syntactically identical.

MGU

To find the MGU of two formulas f and g .

1. $k = 0$; $\sigma_0 = \{\}$; $S_0 = \{f, g\}$
2. If S_k contains an identical pair of formulas stop, and return σ_k as the MGU of f and g .
3. Else find the disagreement set $D_k = \{e_1, e_2\}$ of S_k
4. If $e_1 = V$ a variable, and $e_2 = t$ a term not containing V (or vice-versa) then let
 $\sigma_{k+1} = \sigma_k \{V=t\}$ (Compose the additional substitution)
 $S_{k+1} = S_k \{V=t\}$ (Apply the additional substitution)
 $k = k+1$
GOTO 2
5. Else stop, f and g cannot be unified.

MGU Example 1

$$S_0 = \{p(f(a), g(X)) ; p(Y, Y)\}$$

MGU Example 2

$$S_0 = \{p(a, X, h(g(Z))) ; p(Z, h(Y), h(Y))\}$$

MGU Example 3

$$S_0 = \{p(X,X) \ ; \ p(Y,f(Y))\}$$

Non-Ground Resolution

- Resolution of non-ground clauses. From the two clauses

$(L, Q_1, Q_2, \dots, Q_k)$

$(\neg M, R_1, R_2, \dots, R_n)$

Where there exists σ a MGU for L and M .

We infer the new clause

$(Q_1\sigma, \dots, Q_k\sigma, R_1\sigma, \dots, R_n\sigma)$

Non-Ground Resolution Example

1. $(p(X), q(g(X)))$
2. $(r(a), q(Z), \neg p(a))$

$L=p(X); M=p(a)$
 $\sigma = \{X=a\}$

3. $R[1a,2c]\{X=a\} (q(g(a)), r(a), q(Z))$

The notation is important.

- “R” means resolution step.
- “1a” means the **first** (**a**-th) literal in the first clause i.e. $p(X)$.
- “2c” means the **third** (**c**-th) literal in the second clause, $\neg p(a)$.
 - 1a and 2c are the “clashing” literals.
- $\{X=a\}$ is the substitution applied to make the clashing literals identical.

Resolution Proof Example

“Some patients like all doctors. No patient likes any quack. Therefore no doctor is a quack.”

Resolution Proof Step 1.

Pick symbols to represent these assertions.

$p(X)$: X is a patient

$d(X)$: X is a doctor

$q(X)$: X is a quack

$l(X, Y)$: X likes Y

Resolution Proof Example

Resolution Proof Step 2.

Convert each assertion to a first-order formula.

1. Some patients like all doctors.

F1.

Resolution Proof Example

2. No patient likes any quack

F2.

3. Therefore no doctor is a quack.

Query.

Resolution Proof Example

Resolution Proof Step 3.

Convert to Clausal form.

F1.

F2.

Negation of Query.

Resolution Proof Example

Resolution Proof Step 4.

Resolution Proof from the Clauses.

1. $p(a)$
2. $(\neg d(Y), I(a, Y))$
3. $(\neg p(Z), \neg q(R), \neg I(Z, R))$
4. $d(b)$
5. $q(b)$

Answer Extraction.

- The previous example shows how we can answer true-false questions. With a bit more effort we can also answer “fill-in-the-blanks” questions (e.g., what is wrong with the car?).
- As in Prolog we use free variables in the query where we want to fill in the blanks. We simply need to keep track of the binding that these variables received in proving the query.
 - `parent(art, jon)` –is art one of jon’s parents?
 - `parent(X, jon)` -who is one of jon’s parents?

Answer Extraction.

- A simple bookkeeping device is to use an predicate symbol **answer(X,Y,...)** to keep track of the bindings automatically.
- To answer the query `parent(X,jon)`, we construct the clause
 $(\neg \text{parent}(X,\text{jon}), \text{answer}(X))$
- Now we perform resolution until we obtain a clause consisting of only answer literals (previously we stopped at empty clauses).

Answer Extraction: Example 1

1. $\text{father}(\text{art}, \text{jon})$
2. $\text{father}(\text{bob}, \text{kim})$
3. $(\neg \text{father}(Y, Z), \text{parent}(Y, Z))$
i.e. *all fathers are parents*
4. $(\neg \text{parent}(X, \text{jon}), \text{answer}(X))$
i.e. the query is: who is parent of jon?

Here is a resolution proof:

5. $R[4, 3b]\{Y=X, Z=\text{jon}\}$
 $(\neg \text{father}(X, \text{jon}), \text{answer}(X))$
6. $R[5, 1]\{X=\text{art}\}$ **answer(art)**
so **art** is parent of jon

Answer Extraction: Example 2

1. $(\text{father}(\text{art}, \text{jon}), \text{father}(\text{bob}, \text{jon}))$ //either bob or art is parent of jon
2. $\text{father}(\text{bob}, \text{kim})$
3. $(\neg \text{father}(Y, Z), \text{parent}(Y, Z))$ //i.e. all fathers are parents
4. $(\neg \text{parent}(X, \text{jon}), \text{answer}(X))$ //i.e. query is $\text{parent}(X, \text{jon})$

Here is a resolution proof:

5. $R[4, 3b]\{Y=X, Z=\text{jon}\} (\neg \text{father}(X, \text{jon}), \text{answer}(X))$
6. $R[5, 1a]\{X=\text{art}\} (\text{father}(\text{bob}, \text{jon}), \text{answer}(\text{art}))$
7. $R[6, 3b]\{Y=\text{bob}, Z=\text{jon}\}$
 $(\text{parent}(\text{bob}, \text{jon}), \text{answer}(\text{art}))$
8. $R[7, 4]\{X=\text{bob}\} (\text{answer}(\text{bob}), \text{answer}(\text{art}))$

A disjunctive answer: either bob or art is parent of jon.

Factoring

1. $(p(X), p(Y))$ $// \forall X. \forall Y. \neg p(X) \rightarrow p(Y)$
2. $(\neg p(V), \neg p(W))$ $// \forall V. \forall W. p(V) \rightarrow \neg p(W)$

- These clauses are intuitively contradictory, but following the strict rules of resolution only we obtain:

3. $R[1a, 2a](X=V) (p(Y), \neg p(W))$
Renaming variables: $(p(Q), \neg p(Z))$
4. $R[3b, 1a](X=Z) (p(Y), p(Q))$

No way of generating empty clause!

Factoring is needed to make resolution complete, without it resolution is incomplete!

Factoring (optional)

- If two or more literals of a clause C have an mgu θ , then $C\theta$ with all duplicate literals removed is called a **factor** of C .
- $C = (p(X), p(f(Y)), \neg q(X))$
 $\theta = \{X=f(Y)\}$
 $C\theta = (p(f(Y)), p(f(Y)), \neg q(f(Y))) \rightarrow (p(f(Y)), \neg q(f(Y)))$ is a factor

Adding a factor of a clause can be a step of proof:

1. $(p(X), p(Y))$
2. $(\neg p(V), \neg p(W))$
3. $f[1ab]\{X=Y\} p(Y)$
4. $f[2ab]\{V=W\} \neg p(W)$
5. $R[3,4]\{Y=W\} ()$.

Prolog

- Prolog search mechanism (*without not and cut*) is simply an instance of resolution, except
 1. Clauses are Horn (only one positive literal)
 2. Prolog uses a specific depth first strategy when searching for a proof. (Rules are used first mentioned first used, literals are resolved away left to right).

Prolog

- Append:

1. `append([], Z, Z)`
2. `append([E1 | R1], Y, [E1 | Rest]) :-
 append(R1, Y, Rest)`

Note:

- The second is actually the clause
 `(append([E1|R1], Y, [E1|Rest]) , ¬append(R1,Y,Rest))`
- `[]` is a constant (the empty list)
- `[X | Y]` is `cons(X,Y)`.
- So `[a,b,c]` is short hand for `cons(a,cons(b,cons(c,[])))`

Prolog: Example of proof

- Try to prove : $\text{append}([a,b], [c,d], [a,b,c,d])$:
 1. $\text{append}([], Z, Z)$
 2. $(\text{append}([E1|R1], Y, [E1|Rest]), \neg \text{append}(R1, Y, Rest))$
 3. $\neg \text{append}([a,b], [c,d], [a,b,c,d])$
 4. $R[3,2a]\{E1=a, R1=[b], Y=[c,d], Rest=[b,c,d]\}$
 $\neg \text{append}([b], [c,d], [b,c,d])$
 5. $R[4,2a]\{E1=b, R1=[], Y=[c,d], Rest=[c,d]\}$
 $\neg \text{append}([], [c,d], [c,d])$
 6. $R[5,1]\{Z=[c,d]\} ()$

Review: One Last Example!

Consider the following English description

- Whoever can read is literate.
 - Dolphins are not literate.
 - Flipper is an intelligent dolphin.
-
- Who is intelligent but cannot read.

Example: pick symbols & convert to first-order formula

- Whoever can read is literate.
 $\forall X. \text{read}(X) \rightarrow \text{lit}(X)$
- Dolphins are not literate.
 $\forall X. \text{dolph}(X) \rightarrow \neg \text{lit}(X)$
- Flipper is an intelligent dolphin
 $\text{dolph}(\text{flipper}) \wedge \text{intell}(\text{flipper})$
- Who is intelligent but cannot read?
 $\exists X. \text{intell}(X) \wedge \neg \text{read}(X).$

Example: convert to clausal form

- $\forall X. \text{read}(X) \rightarrow \text{lit}(X)$
 $(\neg \text{read}(X), \text{lit}(X))$
- Dolphins are not literate.
 $\forall X. \text{dolph}(X) \rightarrow \neg \text{lit}(X)$
 $(\neg \text{dolph}(X), \neg \text{lit}(X))$
- Flipper is an intelligent dolphin.
 $\text{dolph}(\text{flipper})$
 $\text{intell}(\text{flipper})$
- who are intelligent but cannot read?
 $\exists X. \text{intell}(X) \wedge \neg \text{read}(X).$
Negated Query $\rightarrow \forall X. \neg \text{intell}(X) \vee \text{read}(X)$
Clausal Form $\rightarrow (\neg \text{intell}(X), \text{read}(X), \text{answer}(X))$

Example: do the resolution proof

1. $(\neg \text{read}(X), \text{lit}(X))$
 2. $(\neg \text{dolp}(X), \neg \text{lit}(X))$
 3. $\text{dolp}(\text{flip})$
 4. $\text{intell}(\text{flip})$
 5. $(\neg \text{intell}(X), \text{read}(X), \text{answer}(X))$

 6. $R[5a,4] \text{ } X=\text{flip}. (\text{read}(\text{flip}), \text{answer}(\text{flip}))$
 7. $R[6,1a] \text{ } X=\text{flip}. (\text{lit}(\text{flip}), \text{answer}(\text{flip}))$
 8. $R[7,2b] \text{ } X=\text{flip}. (\neg \text{dolp}(\text{flip}), \text{answer}(\text{flip}))$
 9. $R[8,3] \text{ } \text{answer}(\text{flip})$
- so flip is intelligent but cannot read!