Knowledge Representation and Reasoning

from the book of the same name

by

Ronald J. Brachman

and

Hector J. Levesque

Morgan Kaufmann Publishers, San Francisco, CA, 2004

1.

Introduction

What is knowledge?

Easier question: how do we talk about it?

We say "John knows that ..." and fill the blank with a <u>proposition</u>

- 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

and weaker yet: "John suspects that ..."

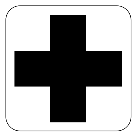
Here: no distinction

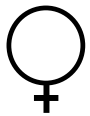
the main idea

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:

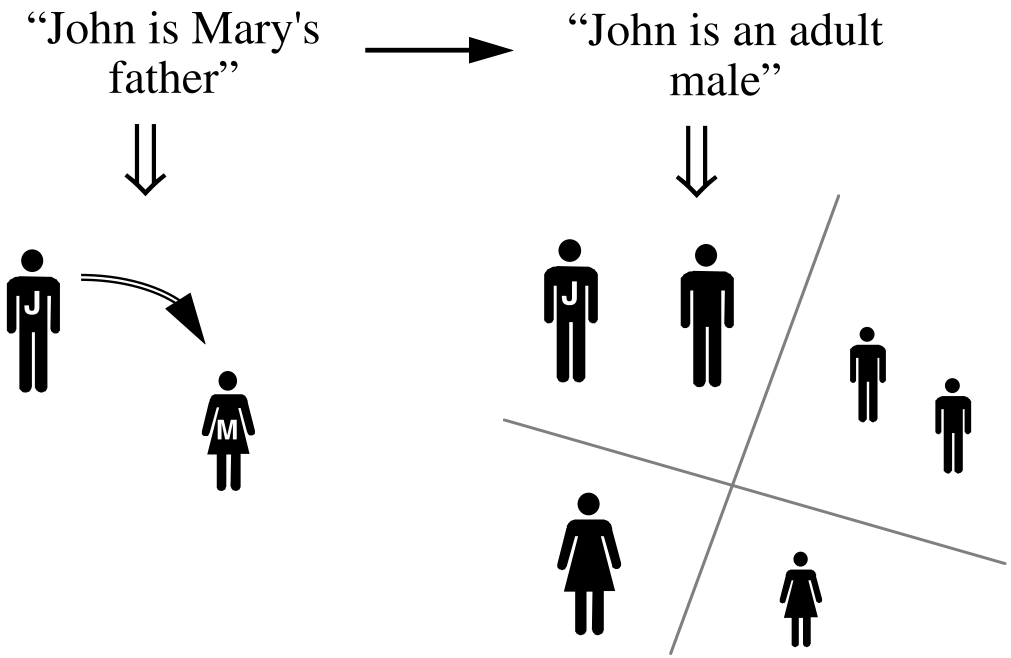
symbolic encoding of propositions believed
(by some agent)

What is reasoning?

Manipulation of symbols encoding propositions to produce representations of new propositions

Analogy: arithmetic

“1011”	+	“10”	$\rightarrow$	“1101”
$\Downarrow$		$\Downarrow$		$\Downarrow$
eleven		two		thirteen



Why knowledge?

For sufficiently complex systems, it is sometimes useful 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 center 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

= taking an intentional stance (Dan Dennett)

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

- sometimes anthropomorphizing is inappropriate

  e.g. thermostats

- can also be very misleading!

  fooling users into thinking a system knows more than it does

Why representation?

Note: 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

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(X,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).
```

Both systems can be described intentionally.

Only the 2nd has a separate collection of symbolic structures à la KR Hypothesis

its <u>knowledge base</u> (or KB)

∴ a small knowledge-based system

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,
- "expert systems", etc.

Some, to a certain extent

game-playing, vision, etc.

Some, to a much lesser extent

speech, motor control, etc.

Current research question:

how much of intelligent behaviour is knowledge-based?

Challenges: connectionism, others

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 current "expert systems":

  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."
- informability: debugging the KB
  - "No the sky is not yellow. It's blue."
- extensibility: new relations
  - "Canaries are yellow."
- extensibility: new applications
  - returning 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

for example: read a book about canaries or rare coins

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 a given situation

Example:

"Patient x is allergic to medication m."

"Anybody allergic to medication m is also allergic to m'."

Is it OK to prescribe m' for x ?

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

Entailment

Sentences $P_1, P_2, ..., P_n$ <u>entail</u> sentence P iff the truth of P is implicit in the truth of $P_1, P_2, ..., P_n$.

If the world is such that it satisfies the P_i then it must also satisfy P.

Applies to a variety of languages (languages with truth theories)

Inference: the process of calculating entailments

- sound: get only entailments
- complete: get all entailments

Sometimes want unsound / incomplete reasoning

for reasons to be discussed later

Logic: study of entailment relations

- languages
- truth conditions
- rules of inference

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
- but will consider subsets / supersets and very different looking representation languages

Knowledge level

Allen Newell's analysis:

- Knowledge level: deals with language, entailment
- Symbol level: deals with representation, inference

Picking a logic has issues at each level

- Knowledge level:
  expressive adequacy,
  theoretical complexity, ...
- Symbol level:
  architectures,
  data structures,
  algorithmic complexity, ...

Next: we begin with FOL at the knowledge level

2.

The Language of First-order Logic

Declarative language

Before building system

before there can be learning, reasoning, planning, explanation ...

need to be able to express knowledge

Want a precise declarative language

- declarative: believe *P* = hold *P* to be <u>true</u>

  cannot believe *P* without some sense of what it would mean for the world to satisfy *P*

- precise: need to know exactly

  what strings of symbols count as sentences

  what it means for a sentence to be true

  (but without having to specify which ones are true)

Here: language of first-order logic

again: not the only choice

Alphabet

Logical symbols:

- Punctuation: (,), .
- Connectives: $\neg$, $\wedge$, $\vee$, $\forall$, $\exists$, $=$
- Variables: $x, x_1, x_2, \ldots, x', x'', \ldots, y, \ldots, z, \ldots$

  Fixed meaning and use

  like keywords in a programming language

Non-logical symbols

- Predicate symbols (like Dog)
- Function symbols (like bestFriendOf)

  Domain-dependent meaning and use

  like identifiers in a programming language

Note: not treating $=$ as a predicate

Have <u>arity</u>: number of arguments

arity 0 predicates: propositional symbols

arity 0 functions: constant symbols

Assume infinite supply of every arity

Grammar

Terms

1. Every variable is a term.
2. If $t_1, t_2, ..., t_n$ are terms and f is a function of arity n, then $f(t_1, t_2, ..., t_n)$ is a term.

Atomic wffs (well-formed formula)

1. If $t_1, t_2, ..., t_n$ are terms and P is a predicate of arity n, then $P(t_1, t_2, ..., t_n)$ is an atomic wff.
2. If t_1 and t_2 are terms, then $(t_1 = t_2)$ is an atomic wff.

Wffs

1. Every atomic wff is a wff.
2. If α and β are wffs, and v is a variable, then $\neg\alpha$, $(\alpha\wedge\beta)$, $(\alpha\vee\beta)$, $\exists v.\alpha$, $\forall v.\alpha$ are wffs.

The propositional subset: no terms, no quantifiers

Atomic wffs: only predicates of 0-arity: $(p \wedge \neg(q \vee r))$

Notation

Occasionally add or omit (,), .

Use [,] and {,} also.

Abbreviations:

$(\alpha \supset \beta)$ for $(\neg\alpha \vee \beta)$

safer to read as disjunction than as "if ... then ..."

$(\alpha \equiv \beta)$ for $((\alpha\supset\beta) \wedge (\beta\supset\alpha))$

Non-logical symbols:

- Predicates: mixed case capitalized

  Person, Happy, OlderThan

- Functions (and constants): mixed case uncapitalized

  fatherOf, successor,
  johnSmith

Variable scope

Like variables in programming languages, the variables in FOL have a <u>scope</u> determined by the quantifiers

Lexical scope for variables

$$P(x) \wedge \exists x[P(x) \vee Q(x)]$$

free bound occurrences of variables

A <u>sentence</u>: wff with no free variables (closed)

Substitution:

$\alpha[v/t]$ means α with all free occurrences of the v replaced by term t

Note: written α^v_t elsewhere (and in book)

Also: $\alpha[t_1,\ldots,t_n]$ means $\alpha[v_1/t_1,\ldots,v_n/t_n]$

Semantics

How to interpret sentences?

- what do sentences claim about the world?
- what does believing one amount to?

Without answers, cannot use sentences to represent knowledge

Problem:

cannot fully specify interpretation of sentences because non-logical symbols reach outside the language

So:

make clear dependence of interpretation on non-logical symbols

Logical interpretation:

specification of how to understand predicate and function symbols

Can be complex!

DemocraticCountry, IsABetterJudgeOfCharacterThan, favouriteIceCreamFlavourOf, puddleOfWater27

The simple case

There are objects.

some satisfy predicate P; some do not

Each interpretation settles extension of P.

borderline cases ruled in separate interpretations

Each interpretation assigns to function f a mapping from objects to objects.

functions always well-defined and single-valued

The FOL assumption:

this is all you need to know about the non-logical symbols to understand which sentences of FOL are true or false

In other words, given a specification of

- what objects there are
- which of them satisfy P
- what mapping is denoted by f

it will be possible to say which sentences of FOL are true

Interpretations

Two parts: $\mathfrak{I} = \langle D, I \rangle$

D is the domain of discourse

can be *any* non-empty set

not just formal / mathematical objects

e.g. people, tables, numbers, sentences, unicorns, chunks of peanut butter, situations, the universe

I is an interpretation mapping

If P is a predicate symbol of arity n,

$I[P] \subseteq D \times D \times \ldots \times D$

an n-ary relation over D

If f is a function symbol of arity n,

$I[f] \in [D \times D \times \ldots \times D \to D]$

an n-ary function over D

for propositional symbols,

$I[p] = \{\}$ or $I[p] = \{\langle\rangle\}$

for constants, $I[c] \in D$

In propositional case, convenient to assume

$\mathfrak{I} = I \in [\textit{prop. symbols} \to \{\text{true, false}\}]$

Denotation

In terms of interpretation $\mathfrak{I}$, terms will denote elements of the domain D.

will write element as $\|t\|_{\mathfrak{I}}$

For terms with variables, the denotation depends on the values of variables

will write as $\|t\|_{\mathfrak{I},\mu}$

where $\mu \in [\mathit{Variables} \rightarrow D]$,
called a <u>variable</u> <u>assignment</u>

Rules of interpretation:

1. $\|v\|_{\mathfrak{I},\mu} = \mu(v)$.
2. $\| f(t_1, t_2, ..., t_n) \|_{\mathfrak{I},\mu} = H(d_1, d_2, ..., d_n)$

   where $H = I[f]$

   and $d_i = \|t_i\|_{\mathfrak{I},\mu}$, recursively

Satisfaction

In terms of an interpretation $\mathfrak{I}$, sentences of FOL will be either true or false.

Formulas with free variables will be true for some values of the free variables and false for others.

Notation:

will write as $\mathfrak{I},\mu \models \alpha$ "α is satisfied by $\mathfrak{I}$ and μ"

where $\mu \in [\mathit{Variables} \rightarrow D]$, as before

or $\mathfrak{I} \models \alpha$, when α is a sentence

"α is true under interpretation $\mathfrak{I}$"

or $\mathfrak{I} \models S$, when S is a set of sentences

"the elements of S are true under interpretation $\mathfrak{I}$"

And now the definition...

Rules of interpretation

1. $\mathfrak{I},\mu \models P(t_1, t_2, ..., t_n)$ iff $\langle d_1, d_2, ..., d_n \rangle \in R$
   where $R = I[P]$
   and $d_i = \| t_i \|_{\mathfrak{I},\mu}$, as on denotation slide
2. $\mathfrak{I},\mu \models (t_1 = t_2)$ iff $\| t_1 \|_{\mathfrak{I},\mu}$ is the same as $\| t_2 \|_{\mathfrak{I},\mu}$
3. $\mathfrak{I},\mu \models \neg\alpha$ iff $\mathfrak{I},\mu \not\models \alpha$
4. $\mathfrak{I},\mu \models (\alpha \wedge \beta)$ iff $\mathfrak{I},\mu \models \alpha$ and $\mathfrak{I},\mu \models \beta$
5. $\mathfrak{I},\mu \models (\alpha \vee \beta)$ iff $\mathfrak{I},\mu \models \alpha$ or $\mathfrak{I},\mu \models \beta$
6. $\mathfrak{I},\mu \models \exists v\alpha$ iff for some $d \in D$, $\mathfrak{I},\mu\{d;v\} \models \alpha$
7. $\mathfrak{I},\mu \models \forall v\alpha$ iff for all $d \in D$, $\mathfrak{I},\mu\{d;v\} \models \alpha$
   where $\mu\{d;v\}$ is just like μ, except that $\mu(v)=d$.

For propositional subset:

$\mathfrak{I} \models p$ iff $I[p] \neq \{\}$ and the rest as above

Entailment defined

Semantic rules of interpretation tell us how to understand all wffs in terms of specification for non-logical symbols.

But some connections among sentences are independent of the non-logical symbols involved.

e.g. If α is true under $\Im$, then so is $\neg(\beta \wedge \neg\alpha)$,
no matter what $\Im$ is, why α is true, what β is, ...

$S \models \alpha$ iff for every $\Im$, if $\Im \models S$ then $\Im \models \alpha$.

Say that S <u>entails</u> α or α is a <u>logical consequence</u> of S:

In other words: for no $\Im$, $\Im \models S \cup \{\neg\alpha\}$. $S \cup \{\neg\alpha\}$ is <u>unsatisfiable</u>

Special case when S is empty: $\models \alpha$ iff for every $\Im$, $\Im \models \alpha$.

Say that α is <u>valid</u>.

Note: $\{\alpha_1, \alpha_2, ..., \alpha_n\} \models \alpha$ iff $\models (\alpha_1 \wedge \alpha_2 \wedge ... \wedge \alpha_n) \supset \alpha$

finite entailment reduces to validity

Why do we care?

We do not have access to user-intended interpretation of non-logical symbols

But, with <u>entailment</u>, we know that if *S* is true in the intended interpretation, then so is α.

If the user's view has the world satisfying *S,* then it must also satisfy α.

There may be other sentences true also; but α is logically guaranteed.

So what about ordinary reasoning?

Dog(fido) ➠ Mammal(fido) ??

Not entailment!

There are logical interpretations where $I[\mathrm{Dog}] \not\subset I[\mathrm{Mammal}]$

Key idea of KR:

include such connections <u>explicitly</u> in *S*
$\forall x[\mathrm{Dog}(x) \supset \mathrm{Mammal}(x)]$
Get: $S \cup \{\mathrm{Dog(fido)}\} \models \mathrm{Mammal(fido)}$

the rest is just details...

Knowledge bases

KB is set of sentences

explicit statement of sentences believed (including any assumed connections among non-logical symbols)

KB $\models \alpha$ α is a further consequence of what is believed

- explicit knowledge: KB
- implicit knowledge: $\{ \alpha \mid \text{KB} \models \alpha \}$

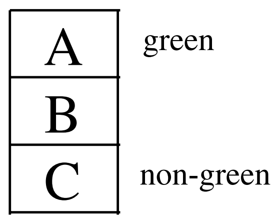
Often non trivial: explicit ➠ implicit

Example:

Three blocks stacked.

Top one is green.

Bottom one is not green.



Is there a green block directly on top of a non-green block?

A formalization

S = {On(a,b), On(b,c), Green(a), $\neg$Green(c)}

all that is required

$\alpha = \exists x \exists y[\mathrm{Green}(x) \wedge \neg\mathrm{Green}(y) \wedge \mathrm{On}(x,y)]$

Claim: $S \models \alpha$

Proof:

Let $\mathfrak{I}$ be any interpretation such that $\mathfrak{I} \models S$.

Case 1: $\mathfrak{I} \models$ Green(b).

$\therefore$ $\mathfrak{I} \models$ Green(b) $\wedge$ $\neg$Green(c) $\wedge$ On(b,c).

$\therefore$ $\mathfrak{I} \models \alpha$

Case 2: $\mathfrak{I} \not\models$ Green(b).

$\therefore$ $\mathfrak{I} \models$ $\neg$Green(b)

$\therefore$ $\mathfrak{I} \models$ Green(a) $\wedge$ $\neg$Green(b) $\wedge$ On(a,b).

$\therefore$ $\mathfrak{I} \models \alpha$

Either way, for any $\mathfrak{I}$, if $\mathfrak{I} \models S$ then $\mathfrak{I} \models \alpha$.

So $S \models \alpha$. QED

Knowledge-based system

Start with (large) KB representing what is explicitly known

e.g. what the system has been told or has learned

Want to influence behaviour based on what is implicit in the KB (or as close as possible)

Requires reasoning

deductive inference:

process of calculating entailments of KB

i.e given KB and any α, determine if KB $\models \alpha$

Process is sound if whenever it produces α, then KB $\models \alpha$

does not allow for plausible assumptions that may be true in the intended interpretation

Process is complete if whenever KB $\models \alpha$, it produces α

does not allow for process to miss some α or be unable to determine the status of α

3.

Expressing Knowledge

Knowledge engineering

KR is first and foremost about knowledge

meaning and entailment

find individuals and properties, then encode facts sufficient for entailments

Before implementing, need to understand clearly

- what is to be computed?
- why and where inference is necessary?

Example domain: soap-opera world

people, places, companies, marriages, divorces, hanky-panky, deaths, kidnappings, crimes, ...

Task: KB with appropriate entailments

- what vocabulary?
- what facts to represent?

Vocabulary

Domain-dependent predicates and functions

main question: what are the individuals?

here: people, places, companies, ...

named individuals

john, sleezyTown, faultyInsuranceCorp, fic, johnQsmith, ...

basic types

Person, Place, Man, Woman, ...

attributes

Rich, Beautiful, Unscrupulous, ...

relationships

LivesAt, MarriedTo, DaughterOf, HadAnAffairWith, Blackmails, ...

functions

fatherOf, ceoOf, bestFriendOf, ...

Basic facts

Usually atomic sentences and negations

- type facts
  - Man(john),
  - Woman(jane),
  - Company(faultyInsuranceCorp)
- property facts
  - Rich(john),
  - ¬HappilyMarried(jim),
  - WorksFor(jim,fic)
- equality facts
  - john = ceoOf(fic),
  - fic = faultyInsuranceCorp,
  - bestFriendOf(jim) = john

Like a simple database (can store in a table)

Complex facts

Universal abbreviations

$\forall y[\text{Woman}(y) \wedge y \neq \text{jane} \ \supset \ \text{Loves}(y,\text{john})]$

$\forall y[\text{Rich}(y) \wedge \text{Man}(y) \ \supset \ \text{Loves}(y,\text{jane})]$

$\forall x \forall y[\text{Loves}(x,y) \supset \neg\text{Blackmails}(x,y)]$

possible to express without quantifiers

Incomplete knowledge

$\text{Loves}(\text{jane},\text{john}) \vee \text{Loves}(\text{jane},\text{jim})$

which?

$\exists x[\text{Adult}(x) \wedge \text{Blackmails}(x,\text{john})]$

who?

cannot write down a more complete version

Closure axioms

$\forall x[\text{Person}(x) \ \supset \ x{=}\text{jane} \vee x{=}\text{john} \vee x{=}\text{jim} \ ...]$

$\forall x \forall y[\text{MarriedTo}(x,y) \ \supset \ ... \ ]$

$\forall x[\ x{=}\text{fic} \vee x{=}\text{jane} \vee x{=}\text{john} \vee x{=}\text{jim} \ ...]$

limit the domain of discourse

also useful to have $\text{jane} \neq \text{john}$ ...

Terminological facts

General relationships among predicates. For example:

disjoint $\forall x[\text{Man}(x) \supset \neg\text{Woman}(x)]$

subtype $\forall x[\text{Senator}(x) \supset \text{Legislator}(x)]$

exhaustive $\forall x[\text{Adult}(x) \supset \text{Man}(x) \vee \text{Woman}(x)]$

symmetry $\forall x \forall y\, [\text{MarriedTo}(x,y) \supset \text{MarriedTo}(y,x)]$

inverse $\forall x \forall y\, [\text{ChildOf}(x,y) \supset \text{ParentOf}(y,x)]$

type restriction $\forall x \forall y\, [\text{MarriedTo}(x,y) \supset$
$\text{Person}(x) \wedge \text{Person}(y) \wedge \text{OppSex}(x,y)]$

sometimes

Usually universally quantified conditionals or biconditionals

Entailments: 1

Is there a company whose CEO loves Jane?

$\exists x\ [\text{Company}(x) \wedge \text{Loves}(\text{ceoOf}(x),\text{jane})]$??

Suppose $\mathfrak{I} \models$ KB.

Then $\mathfrak{I} \models \text{Rich(john)},\ \text{Man(john)}$,

and $\mathfrak{I} \models \forall y[\text{Rich}(y) \wedge \text{Man}(y) \supset \text{Loves}(y,\text{jane})]$

so $\mathfrak{I} \models \text{Loves(john,jane)}$.

Also $\mathfrak{I} \models \text{john} = \text{ceoOf(fic)}$,

so $\mathfrak{I} \models \text{Loves(ceoOf(fic),jane)}$.

Finally $\mathfrak{I} \models \text{Company(faultyInsuranceCorp)}$,

and $\mathfrak{I} \models \text{fic} = \text{faultyInsuranceCorp}$,

so $\mathfrak{I} \models \text{Company(fic)}$.

Thus, $\mathfrak{I} \models \text{Company(fic)} \wedge \text{Loves(ceoOf(fic),jane)}$,

and so

$\mathfrak{I} \models \exists x\ [\text{Company}(x) \wedge \text{Loves}(\text{ceoOf}(x),\text{jane})]$.

Can extract identity of company from this proof

Entailments: 2

If no man is blackmailing John, then is he being blackmailed by somebody he loves?

$\forall x[\text{Man}(x) \supset \neg\text{Blackmails}(x,\text{john})] \supset$
$\exists y[\text{Loves}(\text{john},y) \wedge \text{Blackmails}(y,\text{john})]$??

Note: KB $\models (\alpha \supset \beta)$ iff KB $\cup \{\alpha\} \models \beta$

Let: $\mathfrak{I} \models$ KB $\cup \{\forall x[\text{Man}(x) \supset \neg\text{Blackmails}(x,\text{john})]\}$

Show: $\mathfrak{I} \models \exists y[\text{Loves}(\text{john},y) \wedge \text{Blackmails}(y,\text{john})$

Have: $\exists x[\text{Adult}(x) \wedge \text{Blackmails}(x,\text{john})]$ and $\forall x[\text{Adult}(x) \supset \text{Man}(x) \vee \text{Woman}(x)]$
so $\exists x[\text{Woman}(x) \wedge \text{Blackmails}(x,\text{john})]$.

Then: $\forall y[\text{Rich}(y) \wedge \text{Man}(y) \supset \text{Loves}(y,\text{jane})]$ and $\text{Rich}(\text{john}) \wedge \text{Man}(\text{john})$
so $\text{Loves}(\text{john},\text{jane})$!

But: $\forall y[\text{Woman}(y) \wedge y \neq \text{jane} \supset \text{Loves}(y,\text{john})]$
and $\forall x \forall y[\text{Loves}(x,y) \supset \neg\text{Blackmails}(x,y)]$
so $\forall y[\text{Woman}(y) \wedge y \neq \text{jane} \supset \neg\text{Blackmails}(y,\text{john})]$ and $\text{Blackmails}(\text{jane},\text{john})$!!

Finally: $\text{Loves}(\text{john},\text{jane}) \wedge \text{Blackmails}(\text{jane},\text{john})$
so: $\exists y[\text{Loves}(\text{john},y) \wedge \text{Blackmails}(y,\text{john})]$

What individuals?

Sometimes useful to reduce n-ary predicates to 1-place predicates and 1-place functions

- involves reifying properties: new individuals
- typical of description logics / frame languages (later)

Flexibility in terms of arity:

Purchases(john,sears,bike) or
Purchases(john,sears,bike,feb14) or
Purchases(john,sears,bike,feb14,$100)

Instead: introduce purchase objects

Purchase(p) $\wedge$ agent(p)=john $\wedge$ obj(p)=bike $\wedge$ source(p)=sears $\wedge$ ...

allows purchase to be described at various levels of detail

Complex relationships: MarriedTo(x,y) vs. ReMarriedTo(x,y) vs. ...

Instead define marital status in terms of existence of marriage and divorce events.

Marriage(m) $\wedge$ husband(m)=x $\wedge$ wife(m)=y $\wedge$ date(m)=... $\wedge$...

Abstract individuals

Also need individuals for numbers, dates, times, addresses, etc.

objects about which we ask wh-questions

Quantities as individuals

age(suzy) = 14

age-in-years(suzy) = 14
age-in-months(suzy) = 168

perhaps better to have an object for “the age of Suzy”, whose value in years is 14

years(age(suzy)) = 14

months(x) = 12*years(x)

centimeters(x) = 100*meters(x)

Similarly with locations and times

instead of

time(m)="Jan 5 2006 4:47:03EST"

can use

time(m)=t $\wedge$ year(t)=2006 $\wedge$ ...

Other sorts of facts

Statistical / probabilistic facts

- Half of the companies are located on the East Side.
- Most of the employees are restless.
- Almost none of the employees are completely trustworthy,

Default / prototypical facts

- Company presidents typically have secretaries intercepting their phone calls.
- Cars have four wheels.
- Companies generally do not allow employees that work together to be married.

Intentional facts

- John believes that Henry is trying to blackmail him.
- Jane does not want Jim to think that she loves John.

Others ...

4.

Resolution

Goal

Deductive reasoning in language as close as possible to full FOL

$\neg, \wedge, \vee, \exists, \forall$

Knowledge Level:

given KB, α, determine if KB $\models \alpha$.

or given an open $\alpha[x_1,x_2,...x_n]$, find $t_1,t_2,...t_n$ such that KB $\models \alpha[t_1,t_2,...t_n]$

When KB is finite $\{\alpha_1, \alpha_2, ..., \alpha_k\}$

KB $\models \alpha$

iff $\models [(\alpha_1 \wedge \alpha_2 \wedge ... \wedge \alpha_k) \supset \alpha]$

iff KB $\cup \{\neg\alpha\}$ is unsatisfiable

iff KB $\cup \{\neg\alpha\}$ $\models$ FALSE

where FALSE is something like $\exists x.(x \neq x)$

So want a procedure to test for validity, or satisfiability, or for entailing FALSE.

Will now consider such a procedure (first without quantifiers)

Clausal representation

Formula = set of clauses

Clause = set of literals

Literal = atomic sentence or its negation

positive literal and negative literal

Notation:

If ρ is a literal, then $\bar{\rho}$ is its complement

$\overline{p} \Rightarrow \neg p \qquad \overline{\neg p} \Rightarrow p$

To distinguish clauses from formulas:

[and] for clauses: $[p, \overline{r}, s]$ { and } for formulas: $\{\, [p, \overline{r}, s], [p, r, s], [\,\overline{p}\,] \,\}$

[] is the empty clause {} is the empty formula

So {} is different from {[]}!

Interpretation:

Formula understood as <u>conjunction</u> of clauses

Clause understood as <u>disjunction</u> of literals

Literals understood normally

$\{[p,\neg q], [r], [s]\}$ represents $((p \vee \neg q) \wedge r \wedge s)$

[ ] represents FALSE

CNF and DNF

Every propositional wff α can be converted into a formula α' in <u>C</u>onjunctive <u>N</u>ormal <u>F</u>orm (CNF) in such a way that $\models \alpha \equiv \alpha'$.

1. eliminate $\supset$ and $\equiv$ using $(\alpha \supset \beta) \Rrightarrow (\neg\alpha \vee \beta)$ etc.
2. push $\neg$ inward using $\neg(\alpha \wedge \beta) \Rrightarrow (\neg\alpha \vee \neg\beta)$ etc.
3. distribute $\vee$ over $\wedge$ using $((\alpha \wedge \beta) \vee \gamma) \Rrightarrow ((\alpha \vee \gamma) \wedge (\beta \vee \gamma))$
4. collect terms using $(\alpha \vee \alpha) \Rrightarrow \alpha$ etc.

Result is a conjunction of disjunction of literals.

an analogous procedure produces DNF,
a disjunction of conjunction of literals

We can identify CNF wffs with clausal formulas

$$(p \vee \neg q \vee r) \wedge (s \vee \neg r) \Rrightarrow \{ [p, \neg q, r], [s, \neg r] \}$$

So: given a finite KB, to find out if KB $\models \alpha$, it will be sufficient to

1. put (KB $\wedge \neg\alpha$) into CNF, as above
2. determine the satisfiability of the clauses

Resolution rule of inference

Given two clauses, infer a new clause:

From clause $\{ p \} \cup C_1$,

and $\{\neg p \} \cup C_2$,

infer clause $C_1 \cup C_2$.

$C_1 \cup C_2$ is called a <u>resolvent</u> of input clauses with respect to p.

Example:

clauses $[w, r, q]$ and $[w, s, \neg r]$ have $[w, q, s]$ as resolvent wrt r.

Special Case:

$[p]$ and $[\neg p]$ resolve to [] (the C_1 and C_2 are empty)

A <u>derivation</u> of a clause c from a set *S* of clauses is a sequence $c_1, c_2, ..., c_n$ of clauses, where $c_n = c$, and for each c_i, either

1. $c_i \in S$, or
2. c_i is a resolvent of two earlier clauses in the derivation

Write: $S \rightarrow c$ if there is a derivation

Rationale

Resolution is a symbol-level rule of inference, but has a connection to knowledge-level logical interpretations

Claim: Resolvent is <u>entailed</u> by input clauses.

Suppose $\Im \models (p \vee \alpha)$ and $\Im \models (\neg p \vee \beta)$

Case 1: $\Im \models p$

then $\Im \models \beta$, so $\Im \models (\alpha \vee \beta)$.

Case 2: $\Im \not\models p$

then $\Im \models \alpha$, so $\Im \models (\alpha \vee \beta)$.

Either way, $\Im \models (\alpha \vee \beta)$.

So: $\{(p \vee \alpha), (\neg p \vee \beta)\} \models (\alpha \vee \beta)$.

Special case:

$[p]$ and $[\neg p]$ resolve to [ ],

so $\{[p], [\neg p]\} \models$ FALSE

that is: $\{[p], [\neg p]\}$ is unsatisfiable

Derivations and entailment

Can extend the previous argument to derivations:

If $S \rightarrow c$ then $S \models c$

Proof: by induction on the length of the derivation.
Show (by looking at the two cases) that $S \models c_i$.

But the converse does not hold in general

Can have $S \models c$ without having $S \rightarrow c$.

Example: $\{[\neg p]\} \models [\neg p, \neg q]$ i.e. $\neg p \models (\neg p \vee \neg q)$
but no derivation

However.... Resolution *is* <u>refutation complete</u>!

Theorem: $S \rightarrow []$ iff $S \models []$

sound and complete
when restricted to [ ]

Result will carry over to quantified clauses (later)

So for any set *S* of clauses: *S* is unsatisfiable iff $S \rightarrow []$.

Provides method for determining satisfiability: search all derivations for [].

So provides a method for determining all entailments

A procedure for entailment

To determine if KB $|= \alpha$,

- put KB, $\neg\alpha$ into CNF to get *S*, as before
- check if $S \rightarrow$ [].

If KB = { }, then we are testing the validity of α

Non-deterministic procedure

1. Check if [] is in *S*.

   If yes, then return **UNSATISFIABLE**

2. Check if there are two clauses in *S* such that they resolve to produce a clause that is not already in *S*.

   If no, then return **SATISFIABLE**

3. Add the new clause to *S* and go to 1.

Note: need only convert KB to CNF once

- can handle multiple queries with same KB
- after addition of new fact α, can simply add new clauses α' to KB

So: good idea to keep KB in CNF

Example 1

KB

FirstGrade

FirstGrade $\supset$ Child

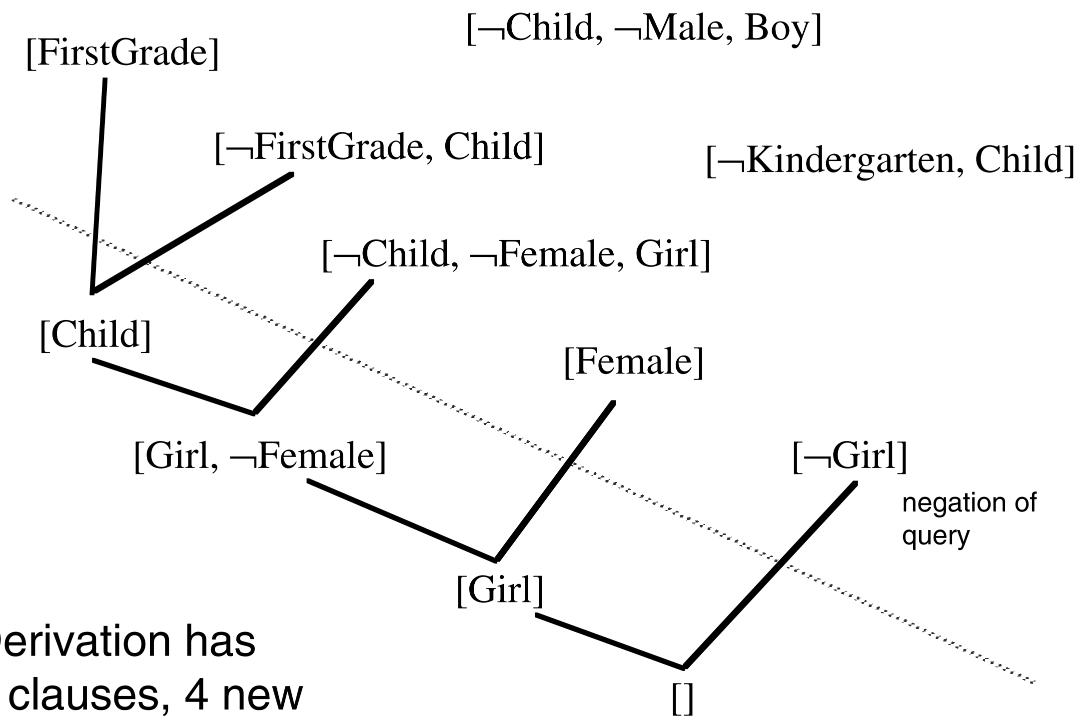
Child $\wedge$ Male $\supset$ Boy

Kindergarten $\supset$ Child

Child $\wedge$ Female $\supset$ Girl

Female

Show that KB $\models$ Girl



Derivation has
9 clauses, 4 new

Example 2

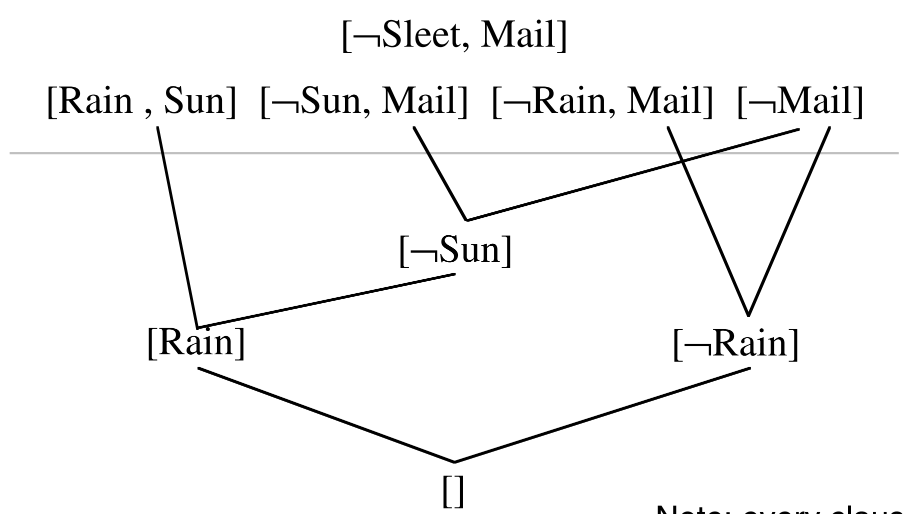
KB

(Rain ∨ Sun)

(Sun ⊃ Mail)

((Rain ∨ Sleet) ⊃ Mail)

Show KB |= Mail



Note: every clause not in *S* has 2 parents

Similarly KB |≠ Rain

Can enumerate all resolvents given ¬Rain, and [] will not be generated

Quantifiers

Clausal form as before, but atom is $P(t_1, t_2, ..., t_n)$, where t_i may contain variables

Interpretation as before, but variables are understood *universally*

Example: $\{ [P(x), \neg R(a,f(b,x))], [Q(x,y)] \}$

interpreted as

$\forall x \forall y \{[R(a,f(b,x)) \supset P(x)] \wedge Q(x,y)\}$

Substitutions: $\theta = \{v_1/t_1, v_2/t_2, ..., v_n/t_n\}$

Notation: If ρ is a literal and θ is a substitution, then $\rho\theta$ is the result of the substitution (and similarly, $c\theta$ where c is a clause)

Example: $\theta = \{x/a, y/g(x,b,z)\}$

$P(x,z,f(x,y))\,\theta = P(a,z,f(a,g(x,b,z)))$

A literal is <u>ground</u> if it contains no variables.

A literal ρ is an <u>instance</u> of ρ', if for some θ, $\rho = \rho'\theta$.

Generalizing CNF

Resolution will generalize to handling variables

Ignore = for now

But to convert wffs to CNF, we need three additional steps:

1. eliminate $\supset$ and $\equiv$
2. push $\neg$ inward using also $\neg\forall x.\alpha \Rightarrow \exists x.\neg\alpha$ etc.
3. standardize variables: each quantifier gets its own variable

   e.g. $\exists x[P(x)] \wedge Q(x) \Rightarrow \exists z[P(z)] \wedge Q(x)$ where z is a new variable
4. eliminate all existentials *(discussed later)*
5. move universals to the front using $(\forall x\alpha) \wedge \beta \Rightarrow \forall x(\alpha \wedge \beta)$

   where β does not use x
6. distribute $\vee$ over $\wedge$
7. collect terms

Get universally quantified conjunction of disjunction of literals

then drop all the quantifiers...

First-order resolution

Main idea: a literal (with variables) stands for all its instances; so allow all such inferences

So given $[P(x,a), \neg Q(x)]$ and $[\neg P(b,y), \neg R(b,f(y))]$,
want to infer $[\neg Q(b), \neg R(b,f(a))]$ among others

since $[P(x,a), \neg Q(x)]$ has $[P(b,a), \neg Q(b)]$ and
$[\neg P(b,y), \neg R(b,f(y))]$ has $[\neg P(b,a), \neg R(b,f(a))]$

Resolution:

Given clauses: $\{\rho_1\} \cup C_1$ and $\{\overline{\rho_2}\} \cup C_2$.

Rename variables, so that distinct in two clauses.

For any θ such that $\rho_1\theta = \rho_2\theta$, can infer $(C_1 \cup C_2)\theta$.

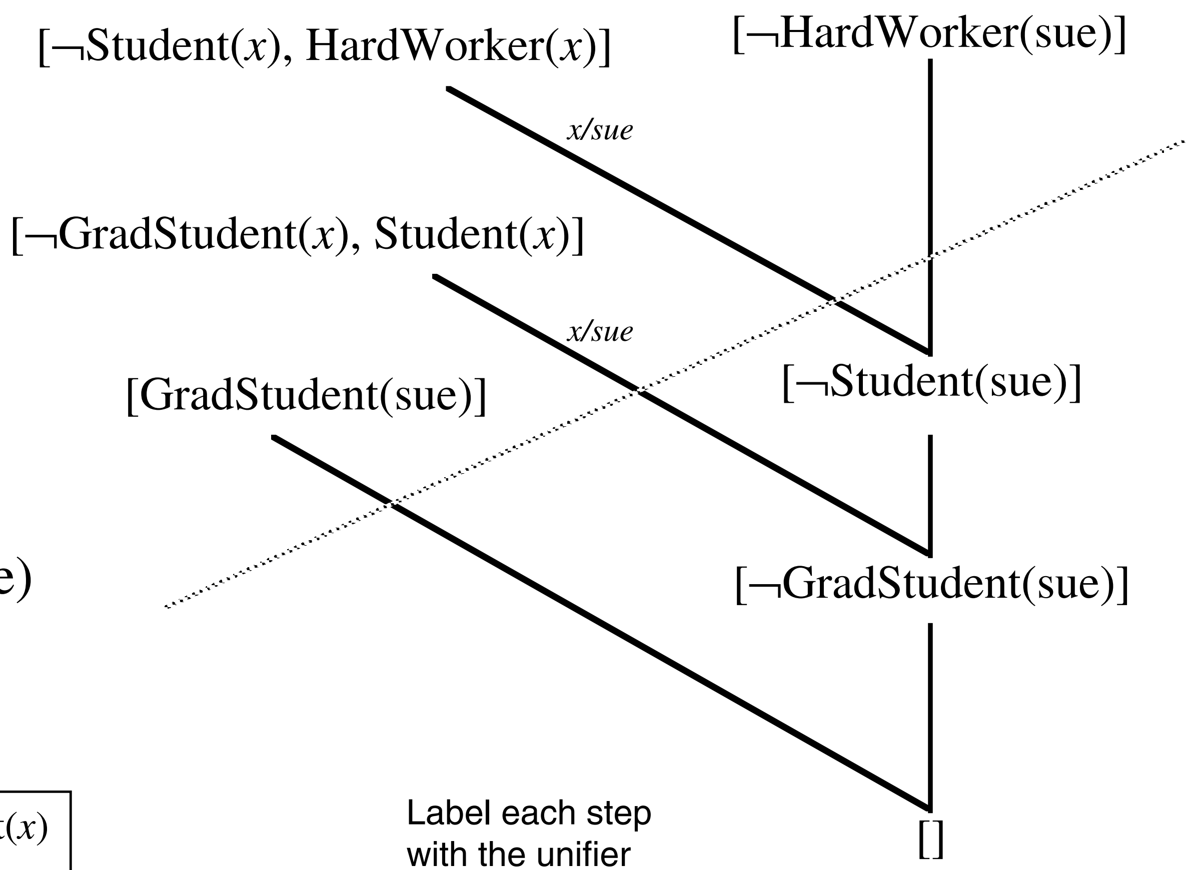
We say that ρ_1 unifies with ρ_2 and that θ is a unifier of the two literals

Resolution derivation: as before

Theorem: $S \rightarrow []$ iff $S \models []$ iff S is unsatisfiable

Note: There are pathological examples where a slightly more general definition of Resolution is required. We ignore them for now...

Example 3



$$\mathrm{KB} \overset{?}{\models} \mathrm{HardWorker(sue)}$$

KB

$\forall x$ GradStudent(x) $\supset$ Student(x)
$\forall x$ Student(x) $\supset$ HardWorker(x)
GradStudent(sue)

Label each step with the unifier

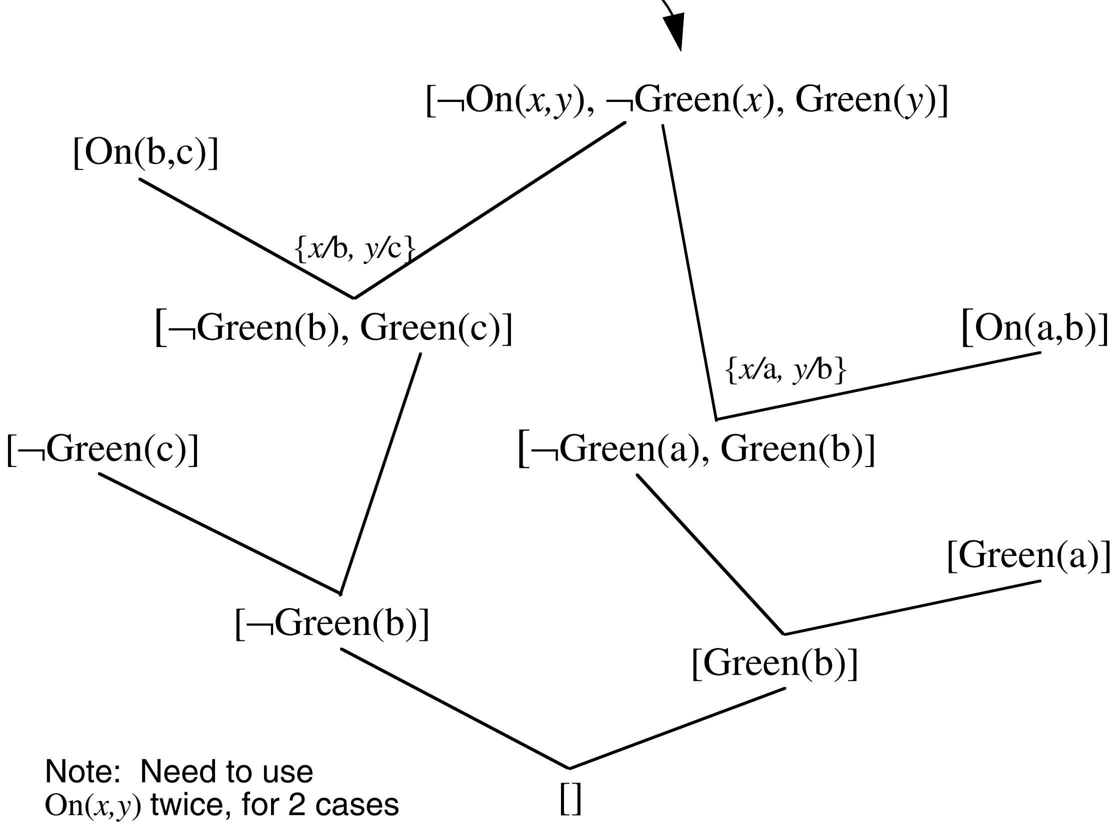
Point to relevant literals in clauses

The 3 block example

KB = {On(a,b), On(b,c), Green(a), ¬Green(c)}   already in CNF

Query = $\exists x \exists y[\mathrm{On}(x,y) \wedge \mathrm{Green}(x) \wedge \neg\mathrm{Green}(y)]$

Note: ¬Q has no existentials, so yields



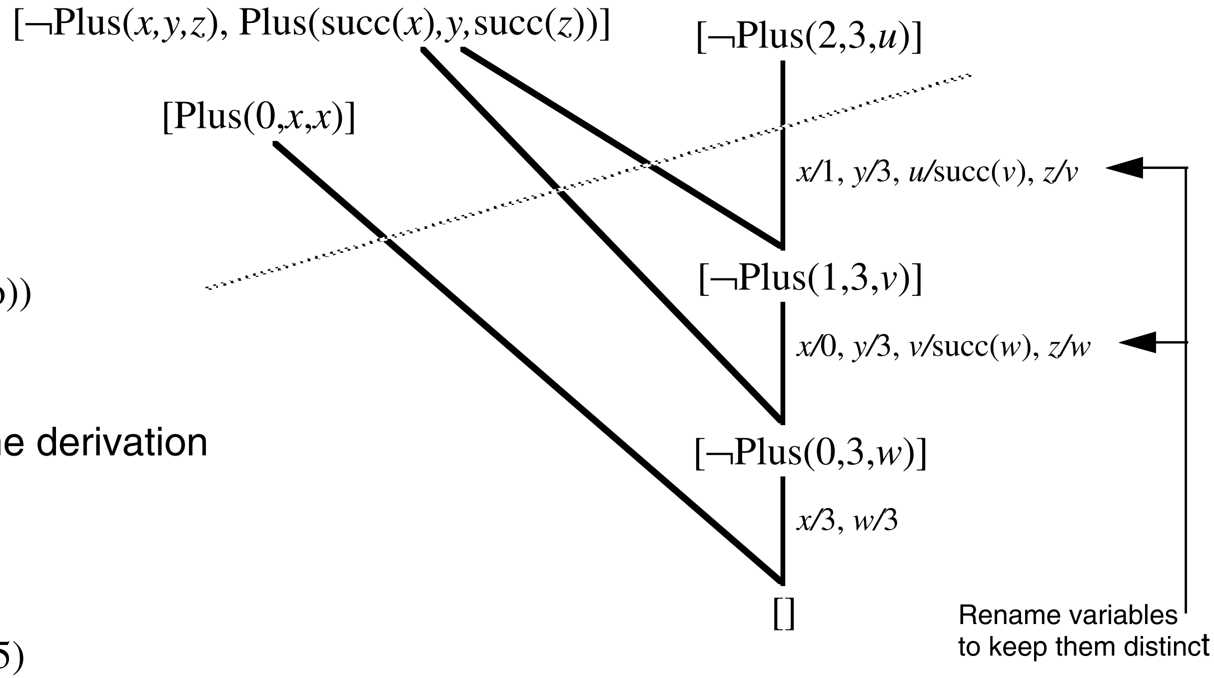
Note: Need to use
On(x,y) twice, for 2 cases

Arithmetic

KB: Plus(zero,x,x)
Plus(x,y,z) $\supset$ Plus(succ(x),y,succ(z))

Q: $\exists u$ Plus(2,3,u)

For readability,
we use
  0 for zero,
  1 for succ(zero),
  2 for succ(succ(zero))
etc.



Can find the answer in the derivation

  u/succ(succ(3))

that is: u/5

Can also derive Plus(2,3,5)

Answer predicates

In full FOL, we have the possibility of deriving $\exists x P(x)$ without being able to derive $P(t)$ for any t.

e.g. the three-blocks problem

$$\exists x \exists y [\mathrm{On}(x,y) \wedge \mathrm{Green}(x) \wedge \neg \mathrm{Green}(y)]$$

but cannot derive which block is which

Solution: answer-extraction process

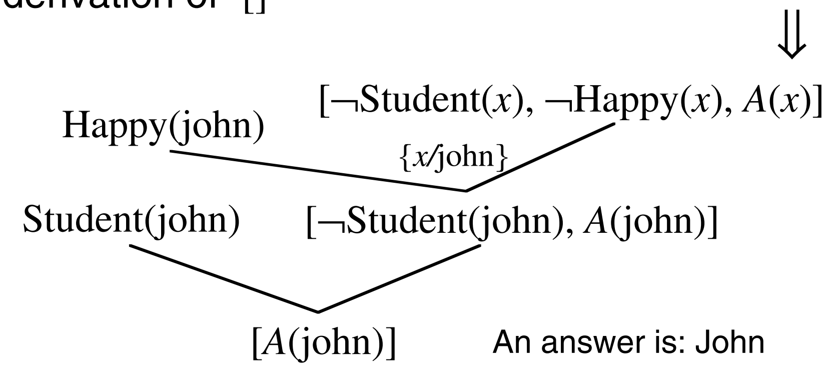
- replace query $\exists x P(x)$ by $\exists x[P(x) \wedge \neg A(x)]$

  where A is a new predicate symbol called the answer predicate

- instead of deriving [], derive any clause containing just the answer predicate
- can always convert to and from a derivation of []

KB: Student(john)
Student(jane)
Happy(john)

Q: $\exists x[\mathrm{Student}(x) \wedge \mathrm{Happy}(x)]$



An answer is: John

Disjunctive answers

KB:

Student(john)
Student(jane)
Happy(john) ∨ Happy(jane)

Query:

$\exists x[\text{Student}(x) \wedge \text{Happy}(x)]$

$\Downarrow$

[¬Student(x), ¬Happy(x), A(x)]

Student(jane) {x/jane}

Student(john) {x/john}

[¬Happy(jane), A(jane)]

[¬Happy(john), A(john)]

[Happy(john), Happy(jane)]

[Happy(john), A(jane)]

[A(jane), A(john)]

An answer is: either Jane or John

Note:

- can have variables in answer
- need to watch for Skolem symbols... (next)

Skolemization

So far, converting wff to CNF ignored existentials

e.g. $\exists x \forall y \exists z P(x,y,z)$

Idea: names for individuals claimed to exist, called Skolem constant and function symbols

there exists an x, call it a

for each y, there is a z, call it $f(y)$

get $\forall y P(a,y,f(y))$

So replace $\forall x_1(\ldots \forall x_2(\ldots \forall x_n(\ldots \exists y[\ldots\ y\ \ldots]\ \ldots)\ldots)\ldots)$

by $\forall x_1(\ldots \forall x_2(\ldots \forall x_n(\ \ldots\ [\ldots f(x_1,x_2,\ldots,x_n)\ \ldots]\ \ldots)\ldots)\ldots)$

f is a new function symbol that appears nowhere else

Skolemization does not preserve equivalence

e.g. $\not\models \exists x P(x) \equiv P(a)$

But it does preserve satisfiability

α is satisfiable iff α' is satisfiable (where α' is the result of Skolemization)

sufficient for resolution!

Variable dependence

Show that $\exists x \forall y R(x,y) \models \forall y \exists x R(x,y)$

show $\{\exists x \forall y R(x,y),\ \neg \forall y \exists x R(x,y)\}$ unsatisfiable

$\exists x \forall y R(x,y) \;\twoheadrightarrow\; \forall y R(a,y)$

$\neg \forall y \exists x R(x,y) \;\twoheadrightarrow\; \exists y \forall x \neg R(x,y) \;\twoheadrightarrow\; \forall x \neg R(x,b)$

then $\{\ [R(a,y)],\ [\neg R(x,b)]\ \} \to []$ with $\{x/a,\ y/b\}$.

Show that $\forall y \exists x R(x,y) \not\models \exists x \forall y R(x,y)$

show $\{\forall y \exists x R(x,y),\ \neg \exists x \forall y R(x,y)\}$ satisfiable

$\forall y \exists x R(x,y) \;\twoheadrightarrow\; \forall y R(f(y),y)$

$\neg \exists x \forall y R(x,y) \;\twoheadrightarrow\; \forall x \exists y \neg R(x,y) \;\twoheadrightarrow\; \forall x \neg R(x,\mathrm{g}(x))$

then get $\{\ [R(f(y),y)],\ [\neg R(x,g(x))]\ \}$

where the two literals do <u>not</u> unify

Note: important to get dependence of variables correct

$R(f(y),y)$ vs. $R(a,y)$ in the above

A problem

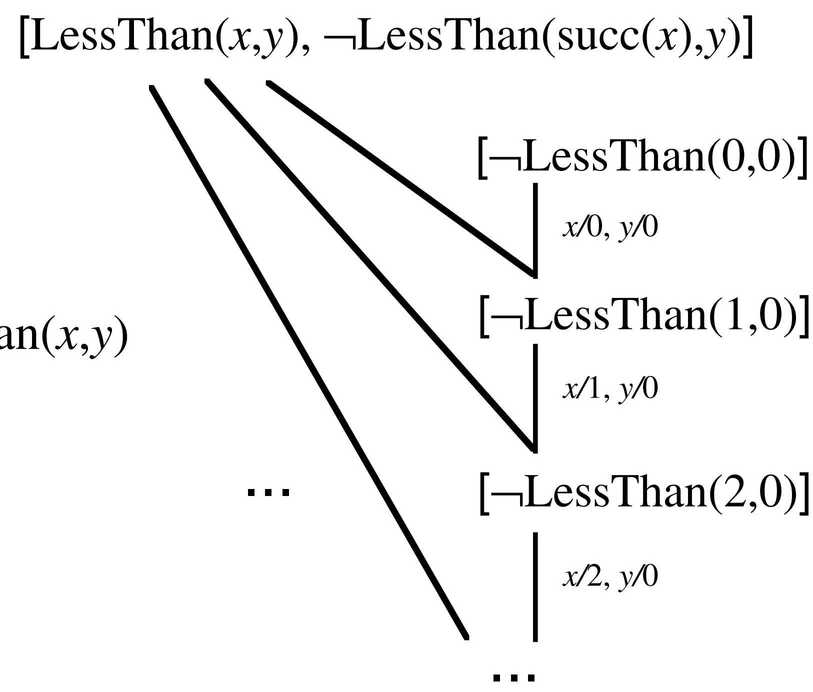
KB:

LessThan(succ(x),y) ⊃ LessThan(x,y)

Query:

LessThan(zero,zero)

Should fail since KB |≠ Q



Infinite branch of resolvents

cannot use a simple depth-first procedure to search for []

Undecidability

Is there a way to detect when this happens?

No! FOL is very powerful

can be used as a full programming language

just as there is no way to detect in general when a program is looping

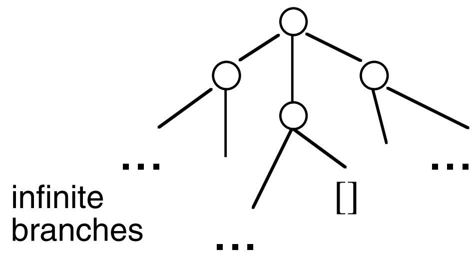
There can be no procedure that does this:

Proc[*Clauses*] =

If *Clauses* are unsatisfiable
then return YES
else return NO

However: Resolution is complete

some branch will contain [], for unsatisfiable clauses



So breadth-first search guaranteed to find []

search may not terminate on satisfiable clauses

Overly specific unifiers

In general, no way to guarantee efficiency, or even termination

later: put control into users' hands

One thing that can be done:

reduce redundancy in search, by keeping search as general as possible

Example

$$\ldots, P(g(x), f(x), z)] \quad [\neg P(y, f(w), a), \ldots$$

unified by

$\theta_1 = \{x/b,\ y/g(b),\ z/a,\ w/b\}$ gives $P(g(b), f(b), a)$

and by

$\theta_2 = \{x/f(z),\ y/g(f(z)),\ z/a,\ w/f(z)\}$ gives $P(g(f(z)), f(f(z)), a)$.

Might not be able to derive the empty clause from clauses having overly specific substitutions

wastes time in search!

Most general unifiers

θ is a <u>most general unifier</u> (MGU) of literals ρ_1 and ρ_2 iff

1. θ unifies ρ_1 and ρ_2
2. for any other unifier θ', there is a another substitution θ^* such that $\theta' = \theta\theta^*$

Note: composition $\theta\theta^*$ requires applying θ^* to terms in θ

for previous example, an MGU is

$$\theta = \{x/w,\ \ y/g(w),\ z/a\}$$

for which

$$\theta_1 \ = \ \theta\{w/b\}$$

$$\theta_2 \ = \ \theta\{w/f(z)\}$$

Theorem: Can limit search to most general unifiers only without loss of completeness (with certain caveats)

Computing MGUs

Computing an MGU, given a set of literals $\{\rho_i\}$

usually only have two literals

1. Start with $\theta := \{\}$.
2. If all the $\rho_i\theta$ are identical, then done;
   otherwise, get disagreement set, *DS*

   e.g $P(a,f(a,g(z),\ldots \quad P(a,f(a,u,\ldots$

   disagreement set, $DS = \{u,\ g(z)\}$
3. Find a variable $v \in$ *DS*, and a term $t \in$ *DS* not containing v.
   If not, fail.
4. $\theta := \theta\{v/t\}$
5. Go to 2

Note: there is a better *linear* algorithm

Herbrand Theorem

Some 1st-order cases can be handled by converting them to a propositional form

Given a set of clauses *S*

- the <u>Herbrand universe</u> of *S* is the set of all terms formed using only the function symbols in *S* (at least one)

  e.g., if *S* uses (unary) f, and c, d, $U = \{c, d, f(c), f(d), f(f(c)), f(f(d)), f(f(f(c))), \ldots\}$

- the <u>Herbrand base</u> of *S* is the set of all $c\theta$ such that $c \in S$ and θ replaces the variables in c by terms from the Herbrand universe

Theorem: *S* is satisfiable iff Herbrand base is

(applies to Horn clauses also)

Herbrand base has no variables, and so is essentially *propositional*, though usually infinite

- finite, when Herbrand universe is finite

  can use propositional methods (guaranteed to terminate)

- sometimes other "type" restrictions can be used to keep the Herbrand base finite

  include $f(t)$ only if t is the correct type

Resolution is difficult!

First-order resolution is not guaranteed to terminate.

What can be said about the propositional case?

Shown by Haken in 1985 that there are unsatisfiable clauses $\{c_1, c_2, ..., c_n\}$ such that the *shortest* derivation of [] contains on the order of 2^n clauses

Even if we could always find a derivation immediately, the most clever search procedure will still require *exponential* time on some problems

Problem just with resolution?

Probably not.

Determining if a set of clauses is satisfiable was shown by Cook in 1972 to be NP-complete

No easier than an extremely large variety of computational tasks

Roughly: any search task where what is searched for can be verified in polynomial time can be recast as a satisfiability problem

» satisfiability

» does graph of cities allow for a full tour of size ≤ *k* miles?

» can N queens be put on an N×N chessboard all safely? and many, many more....

Satisfiability is believed by most people to be unsolvable in polynomial time

SAT solvers

In the propositional case, procedures have been proposed for determining the satisfiability of a set of clauses that appear to work much better in practice than Resolution.

The most popular is called DP (or DPLL) based on ideas by Davis, Putnam, Loveland and Logemann. (See book for details.)

These procedures are called SAT solvers as they are mostly used to find a satisfying interpretation for clauses that are satisfiable.

related to constraint satisfaction programs (CSP)

Typically they have the property that if they *fail* to find a satisfying interpretation, a Resolution derivation of [ ] can be reconstructed from a trace of their execution.

so worst-case exponential behaviour, via Haken's theorem!

One interesting counter-example to this is the procedure GSAT, which has different limitations. (Again, see the book.)

Implications for KR

Problem: want to produce entailments of KB as needed for immediate action

full theorem-proving may be too difficult for KR!

need to consider other options ...

- giving control to user e.g. procedural representations (later)
- less expressive languages e.g. Horn clauses (and a major theme later)

In some applications, it is reasonable to wait

e.g. mathematical theorem proving, where we care about specific formulas

Best to hope for in general: reduce <u>redundancy</u>

main example: MGU, as before

but many other strategies (as we will see)

ATP: automated theorem proving

- area of AI that studies strategies for automatically proving difficult theorems
- main application: mathematics,but relevance also to KR

Strategies

1. Clause elimination

- pure clause

  contains literal ρ such that ρ does not appear in any other clause

  clause cannot lead to []

- tautology

  clause with a literal and its negation

  any path to [] can bypass tautology

- subsumed clause

  a clause such that one with a subset of its literals is already present

  path to [] need only pass through short clause

  can be generalized to allow substitutions

2. Ordering strategies

   many possible ways to order search, but best and simplest is

- unit preference

  prefer to resolve unit clauses first

  Why? Given unit clause and another clause, resolvent is a smaller one ➠ []

Strategies 2

3. Set of support

    KB is usually satisfiable, so not very useful to resolve among clauses with only ancestors in KB

    contradiction arises from interaction with $\neg Q$

    always resolve with at least one clause that has an ancestor in $\neg Q$

    preserves completeness (sometimes)

4. Connection graph

    pre-compute all possible unifications

    build a graph with edges between any two unifiable literals of opposite polarity

    label edge with MGU

    Resolution procedure:

    repeatedly: select link
    compute resolvent
    inherit links from parents after substitution

    Resolution as search: find sequence of links $L_1, L_2, \ldots$ producing []

Strategies 3

5. Special treatment for equality

instead of using axioms for =

relexitivity, symmetry, transitivity, substitution of equals for equals

use new inference rule: paramodulation

from $\{(t{=}s)\} \cup C_1$ and $\{P(\ldots\, t' \ldots)\} \cup C_2$

where $t\theta = t'\theta$

infer $\{P(\ldots\, s \,\ldots)\}\theta \;\cup\; C_1\theta \;\cup\; C_2\theta$.

collapses many resolution steps into one

see also: theory resolution (later)

6. Sorted logic

terms get sorts:

x: Male mother:[Person $\rightarrow$ Female]

keep taxonomy of sorts

only unify $P(s)$ with $P(t)$ when sorts are compatible

assumes only “meaningful” paths will lead to []

Finally...

7. Directional connectives

given $[\neg p, q]$, can interpret as either

from *p*, infer *q* (forward)

to prove *q*, prove *p* (backward)

procedural reading of $\supset$

In 1st case: would only resolve $[\neg p, q]$ with $[p, ...]$ producing $[q, ...]$

In 2nd case: would only resolve $[\neg p, q]$ with $[\neg q, ...]$ producing $[\neg p, ...]$

Intended application:

forward: $\text{Battleship}(x) \supset \text{Gray}(x)$

do not want to try to prove something is gray
by trying to prove that it is a battleship

backward: $\text{Person}(x) \supset \text{Has}(x, \text{spleen})$

do not want to conclude the spleen property for
each individual inferred to be a person

This is the starting point for the procedural representations (later)

5.

Reasoning with Horn Clauses

Horn clauses

Clauses are used two ways:

- as disjunctions: (rain $\vee$ sleet)
- as implications: ($\neg$child $\vee$ $\neg$male $\vee$ boy)

Here focus on 2nd use

Horn clause = at most one +ve literal in clause

- positive / definite clause = exactly one +ve literal

  e.g. $[\neg p_1, \neg p_2, ..., \neg p_n, q]$
- negative clause = no +ve literals

  e.g. $[\neg p_1, \neg p_2, ..., \neg p_n]$ and also [ ]

Note: $[\neg p_1, \neg p_2, ..., \neg p_n, q]$ is a representation for
$(\neg p_1 \vee \neg p_2 \vee ... \vee \neg p_n \vee q)$ or $[(p_1 \wedge p_2 \wedge ... \wedge p_n) \supset q]$

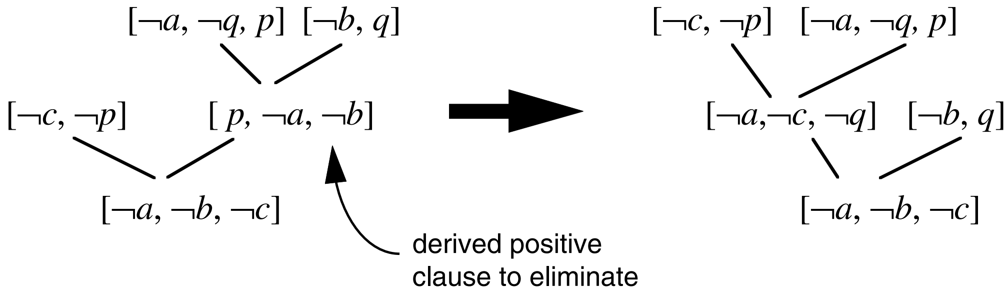
so can read as: If p_1 and p_2 and ... and p_n then q

and write as: $p_1 \wedge p_2 \wedge ... \wedge p_n \Rightarrow q$ or $q \Leftarrow p_1 \wedge p_2 \wedge ... \wedge p_n$

Resolution with Horn clauses

Only two possibilities:

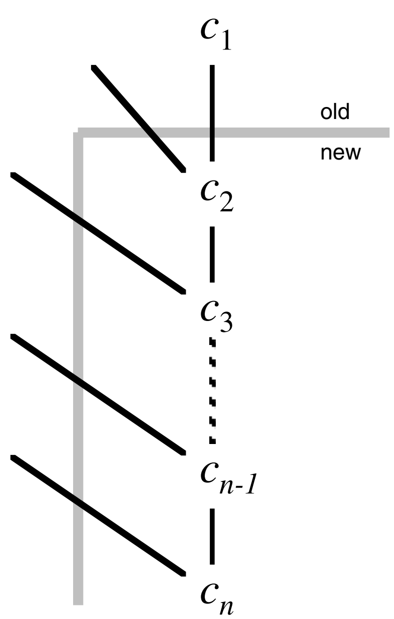
It is possible to rearrange derivations of negative clauses so that all new derived clauses are negative



Further restricting resolution

Can also change derivations such that each derived clause is a resolvent of the previous derived one (negative) and some positive clause in the original set of clauses

- Since each derived clause is negative, one parent must be positive (and so from original set) and one parent must be negative.
- Chain backwards from the final negative clause until both parents are from the original set of clauses
- Eliminate all other clauses not on this direct path



This is a recurring pattern in derivations

- See previously:
  - example 1, example 3, arithmetic example
- But not:
  - example 2, the 3 block example

SLD Resolution

An <u>SLD-derivation</u> of a clause c from a set of clauses *S* is a sequence of clause c_1, c_2, ... c_n such that $c_n = c$, and

1. $c_1 \in S$
2. c_{i+1} is a resolvent of c_i and a clause in *S*

Write: $S \overset{\text{SLD}}{\rightarrow} c$

SLD means S(elected) literals
L(inear) form
D(efinite) clauses

Note: SLD derivation is just a special form of derivation and where we leave out the elements of *S* (except c_1)

In general, cannot restrict ourselves to just using SLD-Resolution

Proof: $S = \{[p, q], [p, \neg q], [\neg p, q]\ [\neg p, \neg q]\}$. Then $S \rightarrow []$.

Need to resolve some $[\ \rho\ ]$ and $[\ \overline{\rho}\ ]$ to get [].
But *S* does not contain any unit clauses.

So will need to derive both $[\ \rho\ ]$ and $[\ \overline{\rho}\ ]$ and then resolve them together.

Completeness of SLD

However, for Horn clauses, we can restrict ourselves to SLD-Resolution

Theorem: SLD-Resolution is refutation complete for Horn clauses: $H \rightarrow []$ iff $H \overset{\text{SLD}}{\rightarrow} []$

So: H is unsatisfiable iff $H \overset{\text{SLD}}{\rightarrow} []$

This will considerably simplify the search for derivations

Note: in Horn version of SLD-Resolution, each clause in the $c_1, c_2, ..., c_n$, will be negative

So clauses H must contain at least one negative clause, c_1 and this will be the only negative clause of H used.

Typical case:

– KB is a collection of positive Horn clauses

– Negation of query is the negative clause

Example 1 (again)

KB

FirstGrade

FirstGrade $\supset$ Child

Child $\wedge$ Male $\supset$ Boy

Kindergarten $\supset$ Child

Child $\wedge$ Female $\supset$ Girl

Female

SLD derivation

[$\neg$Girl]

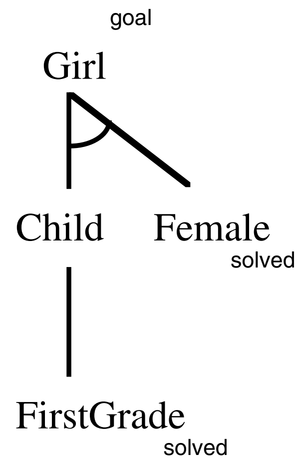
[$\neg$Child, $\neg$Female]

[$\neg$Child]

[$\neg$FirstGrade]

[]

alternate representation



A goal tree whose nodes are atoms, whose root is the atom to prove, and whose leaves are in the KB

Show KB $\cup$ $\{\neg$Girl$\}$ unsatisfiable

Prolog

Horn clauses form the basis of Prolog

$$\text{Append}(\text{nil},y,y)$$

$$\text{Append}(x,y,z) \;\Rightarrow\; \text{Append}(\text{cons}(w,x),y,\text{cons}(w,z))$$

What is the result of appending [c] to the list [a,b] ?

$\text{Append}(\text{cons}(\text{a},\text{cons}(\text{b},\text{nil})),\ \text{cons}(\text{c},\text{nil}),\ u)$ goal

$u\,/\,\text{cons}(\text{a},u')$

$\text{Append}(\text{cons}(\text{b},\text{nil}),\ \text{cons}(\text{c},\text{nil}),\ u')$

$u'\,/\,\text{cons}(\text{b},u'')$

$\text{Append}(\text{nil},\ \text{cons}(\text{c},\text{nil}),\ u'')$

solved: $u''\,/\ \text{cons}(\text{c},\text{nil})$

So goal succeeds with $u = \text{cons}(\text{a},\text{cons}(\text{b},\text{cons}(\text{c},\text{nil})))$
that is: $\text{Append}([\text{a b}],[\text{c}],[\text{a b c}])$

With SLD derivation, can always extract answer from proof

$H \models \exists x\, \alpha(x)$

iff

for some term t, $H \models \alpha(t)$

Different answers can be found by finding other derivations

Back-chaining procedure

Solve[q_1, q_2, ..., q_n] = /* to establish conjunction of q_i */

If n=0 then return **YES**; /* empty clause detected */

For each $d \in$ KB do

If $d = [q_1, \neg p_1, \neg p_2, \ldots, \neg p_m]$ /* match first q */

and /* replace q by -ve lits */

Solve[p_1, p_2, ..., p_m, q_2, ..., q_n] /* recursively */

then return **YES**

end for; /* can't find a clause to eliminate q */

Return **NO**

Depth-first, left-right, back-chaining

- depth-first because attempt p_i before trying q_i
- left-right because try q_i in order, 1,2, 3, ...
- back-chaining because search from goal q to facts in KB p

This is the execution strategy of Prolog

First-order case requires unification *etc.*

Problems with back-chaining

Can go into infinite loop

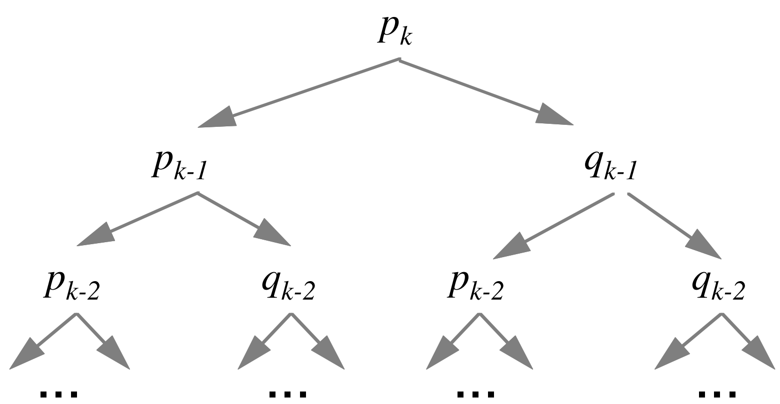
tautologous clause: $[p\ ,\ \neg p]$ (corresponds to Prolog program with `p :- p`).

Previous back-chaining algorithm is inefficient

Example: Consider $2n$ atoms, $p_0, \ldots, p_{n-1}, q_0, \ldots, q_{n-1}$ and $4n\text{-}4$ clauses

$$(p_{i-1} \Rightarrow p_i),\ (q_{i-1} \Rightarrow p_i),\ (p_{i-1} \Rightarrow q_i),\ (q_{i-1} \Rightarrow q_i).$$

With goal p_k the execution tree is like this



Solve[p_k] eventually fails after 2^k steps!

Is this problem inherent in Horn clauses?

Forward-chaining

Simple procedure to determine if Horn KB $|= q$.

main idea: mark atoms as solved

1. If q is marked as solved, then return **YES**
2. Is there a $\{p_1, \neg p_2, ..., \neg p_n\} \in$ KB such that $p_2, ..., p_n$ are marked as solved, but the positive lit p_1 is not marked as solved?

    no: return **NO**

    yes: mark p_1 as solved, and go to 1.

FirstGrade example:

Marks: FirstGrade, Child, Female, Girl then done!

Note: FirstGrade gets marked since all the negative atoms in the clause (none) are marked

Observe:

- only letters in KB can be marked, so at most a linear number of iterations
- not goal-directed, so not always desirable
- a similar procedure with better data structures will run in *linear* time overall

First-order undecidability

Even with just Horn clauses, in the first-order case we still have the possibility of generating an infinite branch of resolvents.

KB:

LessThan(succ(x),y) $\Rightarrow$ LessThan(x,y)

Query:

LessThan(zero,zero)

As with full Resolution, there is no way to detect when this will happen

There is no procedure that will test for the satisfiability of first-order Horn clauses

the question is undecidable

$$[\neg\text{LessThan}(0,0)] \xrightarrow{x/0,\ y/0} [\neg\text{LessThan}(1,0)] \xrightarrow{x/1,\ y/0} [\neg\text{LessThan}(2,0)] \xrightarrow{x/2,\ y/0} \ldots$$

As with non-Horn clauses, the best that we can do is to give control of the deduction to the *user*

to some extent this is what is done in Prolog, but we will see more in “Procedural Control”

6.

Procedural Control of Reasoning

Declarative / procedural

Theorem proving (like resolution) is a general domain-independent method of reasoning

Does not require the user to know how knowledge will be used

will try all logically permissible uses

Sometimes we have ideas about how to use knowledge, how to search for derivations

do not want to use arbitrary or stupid order

Want to communicate to theorem-proving procedure some *guidance* based on properties of the domain

- perhaps specific method to use
- perhaps merely method to avoid

Example: directional connectives

In general: control of reasoning

DB + rules

Can often separate (Horn) clauses into two components:

Example:

MotherOf(jane,billy)
FatherOf(john,billy)
FatherOf(sam, john)
...

a database of facts
- basic facts of the domain
- usually ground atomic wffs

ParentOf(x,y) $\Leftarrow$ MotherOf(x,y)
ParentOf(x,y) $\Leftarrow$ FatherOf(x,y)
ChildOf(x,y) $\Leftarrow$ ParentOf(y,x)
AncestorOf(x,y) $\Leftarrow$ ...
...

collection of rules
- extends the predicate vocabulary
- usually universally quantified conditionals

Both retrieved by unification matching

Control issue: how to use the rules

Rule formulation

Consider AncestorOf in terms of ParentOf

Three logically equivalent versions:

1. $\text{AncestorOf}(x,y) \Leftarrow \text{ParentOf}(x,y)$
   $\text{AncestorOf}(x,y) \Leftarrow \text{ParentOf}(x,z) \wedge \text{AncestorOf}(z,y)$
2. $\text{AncestorOf}(x,y) \Leftarrow \text{ParentOf}(x,y)$
   $\text{AncestorOf}(x,y) \Leftarrow \text{ParentOf}(z,y) \wedge \text{AncestorOf}(x,z)$
3. $\text{AncestorOf}(x,y) \Leftarrow \text{ParentOf}(x,y)$
   $\text{AncestorOf}(x,y) \Leftarrow \text{AncestorOf}(x,z) \wedge \text{AncestorOf}(z,y)$

Back-chaining goal of AncestorOf(sam,sue) will ultimately reduce to set of ParentOf(–,–) goals

1. get ParentOf(sam,z): find child of Sam searching *downwards*
2. get ParentOf(z,sue): find parent of Sue searching *upwards*
3. get ParentOf(–,–): find parent relations searching *in both directions*

Search strategies are not equivalent

if more than 2 children per parent, (2) is best

Algorithm design

Example: Fibonacci numbers

1, 1, 2, 3, 5, 8, 13, 21, ...

Version 1:

$\mathrm{Fibo}(0, 1)$

$\mathrm{Fibo}(1, 1)$

$\mathrm{Fibo}(\mathrm{s}(\mathrm{s}(n)), x) \Leftarrow \mathrm{Fibo}(n, y) \wedge \mathrm{Fibo}(\mathrm{s}(n), z) \wedge \mathrm{Plus}(y, z, x)$

Requires *exponential* number of Plus subgoals

Version 2:

$\mathrm{Fibo}(n, x) \Leftarrow \mathrm{F}(n, 1, 0, x)$

$\mathrm{F}(0, c, p, c)$

$\mathrm{F}(\mathrm{s}(n), c, p, x) \Leftarrow \mathrm{Plus}(p, c, s) \wedge \mathrm{F}(n, s, c, x)$

Requires only *linear* number of Plus subgoals

Ordering goals

Example:

AmericanCousinOf(x,y) $\Leftarrow$ American(x) $\wedge$ CousinOf(x,y)

In back-chaining, can try to solve either subgoal first

Not much difference for AmericanCousinOf(fred, sally), but big difference for AmericanCousinOf(x, sally)

1. find an American and then check to see if she is a cousin of Sally
2. find a cousin of Sally and then check to see if she is an American

So want to be able to order goals

better to generate cousins and test for American

In Prolog: order clauses, and literals in them

Notation: G :- $G_1, G_2, ..., G_n$ stands for

$G \Leftarrow G_1 \wedge G_2 \wedge ... \wedge G_n$

but goals are attempted in presented order

Commit

Need to allow for backtracking in goals

AmericanCousinOf(x,y) :- CousinOf(x,y), American(x)

for goal AmericanCousinOf(x,sally), may need to try to solve the goal American(x) for many values of x

But sometimes, given clause of the form

G :- *T*, *S*

goal *T* is needed only as a *test* for the applicability of subgoal *S*

- if *T* succeeds, commit to *S* as the *only* way of achieving goal *G*.
- if *S* fails, then *G* is considered to have failed
  - do not look for other ways of solving *T*
  - do not look for other clauses with *G* as head

In Prolog: use of cut symbol

Notation: G :- T_1, T_2, ..., T_m, !, G_1, G_2, ..., G_n

attempt goals in order, but if all T_i succeed, then commit to G_i

If-then-else

Sometimes inconvenient to separate clauses in terms of unification:

$\mathrm{G}(\mathrm{zero}, -)$:- *method 1*
$\mathrm{G}(\mathrm{succ}(n), -)$:- *method 2*

For example, may split based on computed property:

$\mathrm{Expt}(a, n, x)$:- $\mathrm{Even}(n)$, ... (*what to do when* n *is even*)
$\mathrm{Expt}(a, n, x)$:- $\mathrm{Even}(\mathrm{s}(n))$, ... (*what to do when* n *is odd*)

want: check for even numbers only once

Solution: use ! to do if-then-else

G :- P, !, Q.
G :- R.

To achieve G: if P then use Q else use R

Example:

$\mathrm{Expt}(a, n, x)$:- $n = 0$, !, $x = 1$.
$\mathrm{Expt}(a, n, x)$:- $\mathrm{Even}(n)$, !, (*for even* n)
$\mathrm{Expt}(a, n, x)$:- (*for odd* n)

Note: it would be correct to write

$\mathrm{Expt}(a, 0, x)$:- !, $x = 1$.

but not

$\mathrm{Expt}(a, 0, 1)$:- !.

Controlling backtracking

Consider solving a goal like

1 AncestorOf(jane,billy), Male(jane)

2 ParentOf(jane,billy), Male(jane)

3 Male(jane)

FAILS

4 ParentOf(z, billy), AncestorOf(jane, z), Male(jane)

Eventually FAILS

So goal should really be: AncestorOf(jane,billy), !, Male(jane)

Similarly:

$$\text{Member}(x,l) \Leftarrow \text{FirstElement}(x,l)$$
$$\text{Member}(x,l) \Leftarrow \text{Rest}(l,l') \wedge \text{Member}(x,l')$$

If only to be used for testing, want

$$\text{Member}(x,l) \text{ :- } \text{FirstElement}(x,l), !, .$$

On failure, do not try to find another x later in the rest of the list

Negation as failure

Procedurally: we can distinguish between the following:

can solve goal $\neg G$ *vs.* cannot solve goal G

Use **not**(G) to mean the goal that succeeds if G fails, and fails if G succeeds

Roughly: **not**(G) :- G, !, fail. /* fail if G succeeds */
not(G). /* otherwise succeed */

Only terminates when failure is *finite* (no more resolvents)

Useful when DB + rules is complete

NoChildren(x) :- **not**(ParentOf(x,y))

or when method already exists for complement

Composite(n) :- n > 1, **not**(PrimeNum(n))

Declaratively: same reading as $\neg$, but not when *new* variables in G

[**not**(ParentOf(x,y)) $\supset$ NoChildren(x)] ✓

vs. [$\neg$ParentOf(x,y) $\supset$ NoChildren(x)] ✗

Dynamic DB

Sometimes useful to think of DB as a snapshot of the world that can be changed dynamically

assertions and deletions to the DB

then useful to consider 3 procedural interpretations for rules like

$$\mathrm{ParentOf}(x,y) \Leftarrow \mathrm{MotherOf}(x,y)$$

1. If-needed: Whenever have a goal matching $\mathrm{ParentOf}(x,y)$, can solve it by solving $\mathrm{MotherOf}(x,y)$

   ordinary back-chaining, as in Prolog

2. If-added: Whenever something matching $\mathrm{MotherOf}(x,y)$ is added to the DB, also add $\mathrm{ParentOf}(x,y)$

   forward-chaining

3. If-removed: Whenever something matching $\mathrm{ParentOf}(x,y)$ is removed from the DB, also remove $\mathrm{MotherOf}(x,y)$, if this was the reason

   keeping track of dependencies in DB

Interpretations (2) and (3) suggest demons

procedures that monitor DB and fire when certain conditions are met

The Planner language

Main ideas:

1. DB of facts

   (Mother susan john)   (Person john)

2. If-needed, if-added, if-removed procedures consisting of
   - body: program to execute
   - pattern for invocation   (Mother *x* *y*)

3. Each program statement can succeed or fail
   - **`(goal`** *p***`)`**, **`(assert`** *p***`)`**, **`(erase`** *p***`)`**,
   - **`(and`** *s* ... *s***`)`**, statements with backtracking
   - **`(not`** *s***`)`**, negation as failure
   - **`(for`** *p* *s***`)`**, do *s* for every way *p* succeeds
   - **`(finalize`** *s***`)`**, like cut
   - a lot more, including all of Lisp

examples:

```
(proc if-needed (cleartable)
   (for (on x table)
      (and (erase (on x table)) (goal (putaway x)))))

(proc if-removed (on x y) (print x " is no longer on " y))
```

Shift from proving conditions
to making conditions hold!

7.

Rules in Production Systems

Direction of reasoning

A conditional like $P \Rightarrow Q$ can be understood as transforming

- assertions of P to assertions of Q
- goals of Q to goals of P

Can represent the two cases explicitly:

(**assert** P) $\Rightarrow$ (**assert** Q)

(**goal** Q) $\Rightarrow$ (**goal** P)

and then distinguish between

1. goal vs. data directed reasoning
   - goal: from Q towards P
   - data: from P towards Q
2. forward vs. backward-chaining
   - forward: along the $\Rightarrow$
   - backward: against the $\Rightarrow$

Possible to have

- **(proc if-added (mygoal Q) ... (mygoal P))**
- **(proc if-needed (myassert P)... (myassert Q))**

How to do data-directed reasoning in Prolog

Now: a formalism with forward-chaining

Production systems

Idea: working memory + production rule set

Working memory: like DB, but volatile

Production rule: **`IF`** *conditions* **`THEN`** *actions*

condition: tests on WM

action: changes to WM

Basic operation: cycle of

1. recognize

   find <u>conflict set</u>: rules whose conditions are satisfied by current WM

2. resolve

   determine which of the rules will fire

3. act

   perform required changes to WM

Stop when no rules fire

Working memory

Set of working memory elements (WME)

Each WME is of the form $(type\ \ attr_1\ val_1\ \ attr_2\ val_2\ \ \ldots\ \ attr_n\ \ val_n)$

where $type$, $attr_i$, val_i are all atoms

Examples: (person age 27 home Toronto)
(goal task openDoor importance 5)
(student name JohnSmith dept CS)

Understood as $\exists x[type(x) \wedge attr_1(x)=val_1 \wedge \ldots \wedge attr_n(x)=val_n]$

- individual is not explicitly named
- order of attributes is not significant

Can handle n-ary relations as usual

(myAssertion relation OlderThan firstArg John secondArg Mary)

Rule conditions

Conditions: tested conjunctively

a condition is p or $-p$, where p is a pattern of the form

$(type\ \ attr_1\ spec_1\ \dots\ attr_k\ \ spec_k)$

where each specification must be one of

- an atom
- an expression within [ ]
- a variable
- a test, within {}
- the $\wedge$, $\vee$, $\neg$ of a specification

Examples:

(person age $[n+4]$ occupation x)

$-$ (person age $\{< 23 \wedge > 6\}$)

A rule is applicable if there are values of the variables to satisfy all the conditions

- for a pattern, need WME of the correct type and for each *attr* in pattern, *val* must match *spec*
- for $-p$, there must be no WME that matches p $\quad\therefore$ negation as failure

Rule actions

Actions: performed sequentially

An action is of the form

- **`ADD`** *pattern*
- **`REMOVE`** *index*
- **`MODIFY`** *index* (*attr spec*)

where

- index i refers to the WME that matched i-th pattern (inapplicable to **-**p)
- variables and expressions refer to values obtained in the matching

Examples:

`IF` (Student name x)
`THEN` **`ADD`** (Person name x)
ordinary forward chaining

`IF` (Person age x) (Birthday)
`THEN` **`REMOVE 2`**
`MODIFY 1` (age [x+1])
database update

`IF` (starting)
`THEN` **`REMOVE 1`**
`ADD` (phase val 1) control information

Example 1

Placing bricks in order of size

largest in place 1, next in place 2, etc.

Initial working memory

```
(counter  index 1)
(brick  name A  size 10  place heap)
(brick  name B  size 30  place heap)
(brick  name C  size 20  place heap)
```

Production rules:

IF (brick place heap name n size s)
-(brick place heap size {> s})
-(brick place hand)
THEN **MODIFY 1** (place hand)

put the largest brick in your hand

IF (brick place hand) (counter index i)
THEN **MODIFY 1** (place i)
MODIFY 2 (index [i+1])

put a brick in your hand at the next spot

Trace

Only one rule can fire at a time, so no conflict resolution is required

The following modifications to WM

1. (brick name B size 30 place hand)
2. (brick name B size 30 place 1)
   (counter index 2)
3. (brick name C size 20 place hand)
4. (brick name C size 20 place 2)
   (counter index 3)
5. (brick name A size 10 place hand)
6. (brick name A size 10 place 3)
   (counter index 4)

So the final working memory is

(counter index 4) (brick name A size 10 place 3) (brick name B size 30 place 1) (brick name C size 20 place 2)

Example 2

How many days are there in a year?

Start with: (want-days year *n*)

End with: (has-days days *m*)

1. **IF** (want-days year *n*)
   THEN **REMOVE 1**
   ADD (year mod4 [*n mod* 4]
   mod100 [*n mod* 100]
   mod400 [*n mod* 400])

2. **IF** (year mod400 0)
   THEN **REMOVE 1** **ADD** (has-days days 366)

3. **IF** (year mod100 0 mod400 {≠ 0})
   THEN **REMOVE 1** **ADD** (has-days days 365)

4. **IF** (year mod4 0 mod100 {≠ 0})
   THEN **REMOVE 1** **ADD** (has-days days 366)

5. **IF** (year mod4 {≠ 0})
   THEN **REMOVE 1** **ADD** (has-days days 365)

Applications

1. Psychological modeling

```
IF   (goal is get-unit-digit)
     (minuend unit d)
     (subtrahend unit {> d})
THEN REMOVE 1
     ADD (goal is borrow-from-tens)
```

fine-grained modeling of symbol manipulation performed by people during problem solving

2. Expert systems

   rules used by experts in a problem area to perform complex tasks

   (examples later)

Claimed advantages:

- modularity: each rule acts independently of the others
- fine-grained control: no complex goal or control stack
- transparency: can recast rules in English to provide explanation of behaviour

MYCIN

System developed at Stanford to aid physicians in treating bacterial infections

Approximately 500 rules for recognizing about 100 causes of infection

```
IF
```
the type of *x* is primary bacteremia

the suspected entry point of *x* is the gastrointestinal tract

the site of the culture of *x* is one of the sterile sites

```
THEN
```
there is evidence that *x* is bacteroides

\+

other more static data structures (not in WM)

- lists of organisms
- clinical parameters

Certainty factors

numbers from 0 to 1 attached to conclusions to rank order alternatives

AND – take min OR – take max

XCON

System developed at CMU (as R1) and used extensively at DEC (now owned by Compaq) to configure early Vax computers

Nearly 10,000 rules for several hundred component types

- Major stimulus for commercial interest in rule-based expert systems ☆

```
IF
    the context is doing layout and assigning a power supply
    an sbi module of any type has been put in a cabinet
    the position of the sbi module is known
    there is space available for the power supply
    there is no available power supply
    the voltage and the frequency of the components are known
THEN
    add an appropriate power supply
```

Context switching

XCON and others use rules of the form

IF the current context is *x*
THEN deactivate *x*
activate context *y*

organized to fire when no other rules apply

Useful for grouping rules

IF (control phase 1) **AND** ...
THEN ...
...
IF (control phase 1) **AND** ...
THEN ... **MODIFY** **1** (phase 2) ...

IF (control phase 2) **AND** ...
THEN ...
...
IF (control phase 2) **AND** ...
THEN ... **MODIFY** **1** (phase 3) ...

Allows emulation of control structures.

But still difficult for *complex* control

Conflict resolution

Sometimes with data-directed reasoning, we want to fire *all* applicable rules

With goal-directed reasoning, we may want a *single* rule to fire

- arbitrary
- first rule in order of presentation (as in Prolog)
- specificity, as in

  IF (bird) **THEN ADD** (can-fly)

  IF (bird weight {> 100}) **THEN ADD** (cannot-fly)

  IF (bird) (penguin) **THEN ADD** (cannot-fly)

- recency
  - fire on rule that uses most recent WME
  - fire on least recently used rule
- refractoriness
  - never use same rule for same value of variables (called <u>rule instance</u>)
  - only use a rule/WME pair once (will need a "refresh" otherwise)

Conflict combinations

OPS5:

1. discard rule instances that have already been used
2. order remaining instances in terms of recency of WME matching 1st condition (and then of 2nd condition, etc.)
3. if still no single rule, order rules by number of conditions
4. select arbitrarily among those remaining

SOAR:

system that attempts to find a way to move from a start state to a goal state by applying productions

selecting what rule to fire

$\equiv$

deciding what to do next

if unable to decide, SOAR sets up the selection as a new (meta-)goal to solve, and the process iterates

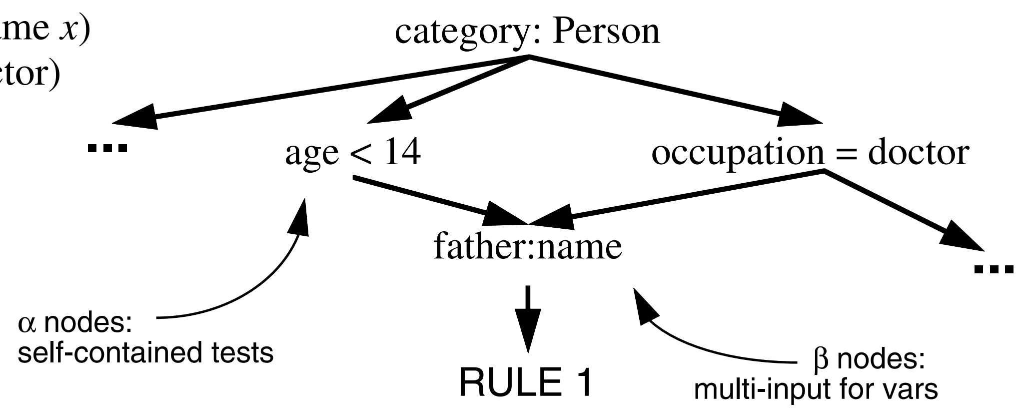
Rete procedure

Early systems spent 90% of their time matching, even with indexing and hashing.

But:
- WM is modified only slightly on each cycle
- many rules share conditions

So:
- incrementally pass WME through network of tests
- tokens that make it through satisfy all conditions and produce conflict set
- can calculate new conflict set in terms of old one and change to WM

IF (Person father *y* age {< 14} name *x*)
(Person name *y* occupation doctor)
THEN ...



8.

Object-Oriented Representation

Organizing procedures

With the move to put control of inference into the user's hands, we're focusing on more procedural representations

knowing facts by executing code

Even production systems are essentially programming languages.

Note also that everything so far is *flat*, i.e., sentence-like representations

- information about an object is scattered in axioms
- procedure fragments and rules have a similar problem

With enough procedures / sentences in a KB, it could be critical to *organize* them

- production systems might have rule sets, organized by context of application
- but this is not a natural, *representational* motivation for grouping

Object-centered representation

Most obvious organizational technique depends on our ability to see the world in terms of <u>objects</u>

- physical objects:
  - a desk has a surface-material, # of drawers, width, length, height, color, procedure for unlocking, etc.
  - some variations: no drawers, multi-level surface, built-in walls (carrel)
- also, *situations* can be object-like:
  - a class: room, participants, teacher, day, time, seating arrangement, lighting, procedures for registering, grading, etc.
  - leg of a trip: destination, origin, conveyance, procedures for buying ticket, getting through customs, reserving hotel room, locating a car rental etc.

Suggests clustering procedures for determining properties, identifying parts, interacting with parts, as well as constraints between parts, all of *objects*

- legs of desk connect to and support surface
- beginning of a travel leg and destination of prior one

object-centered constraints

Situation recognition

Focus on objects as an organizational / chunking mechanism to make some things easier to find

Suggests a different kind of reasoning than that covered so far

basic idea originally proposed by Marvin Minsky

- recognize (guess) situation; activate relevant object representations
- use those object representations to set up expectations

  some for verification; some make it easier to interpret new details

- flesh out situation once you've recognized

Wide applicability, but typical applications include

- relationship recognition e.g., story understanding
- data monitoring
- propagation and enforcement of constraints for planning tasks

  this latter is most doable and understandable,
  so we will concentrate on it

Basic frame language

Let's call our object structures frames

note wide variety of interpretations in literature

Two types:

- individual frames

  represent a single object like a person, part of a trip

- generic frames

  represent categories of objects, like students

An individual frame is a named list of buckets called slots. What goes in the bucket is called a filler of the slot. It looks like this:

(*frame-name*
  <*slot-name1* *filler1*>
  <*slot-name2* *filler2* > …)

where frame names and slot names are atomic, and fillers are either numbers, strings or the names of other individual frames.

Notation:	individual frames:	toronto
	slot names:	:Population (note ":" at start)
	generic frames:	CanadianCity

Instances and specializations

Individual frames have a special slot called <u>:INSTANCE-OF</u> whose filler is the name of a generic frame:

(toronto
<:**INSTANCE-OF** CanadianCity>
<:Province ontario>
<:Population 4.5M>…)

(tripLeg123-1
<:**INSTANCE-OF** TripLeg>
<:Destination toronto>…)

Generic frames have a syntax that is similar to that of individual frames, except that they have a slot called <u>:IS-A</u> whose filler is the name of another generic frame

(CanadianCity
<:**IS-A** City>
<:Province CanadianProvince>
<:Country canada>…)

We say that the frame toronto is an <u>instance</u> of the frame CanadianCity and that the frame CanadianCity is a <u>specialization</u> of the frame City

Procedures and defaults

Slots in generic frames can have associated <u>procedures</u>

1. computing a filler (when no slot filler is given)

   (Table
   <:Clearance [**IF-NEEDED** `computeClearanceFromLegs`]> …)

2. propagating constraints (when a slot filler is given)

   (Lecture
   <:DayOfWeek WeekDay>
   <:Date [**IF-ADDED** `computeDayOfWeek`]> …)

If we create an instance of Table, the :Clearance will be calculated as needed. Similarly, the filler for :DayOfWeek will be calculated when :Date is filled.

For instances of CanadianCity, the :Country slot will be filled automatically. But we can also have

(city135
<:**INSTANCE-OF** CanadianCity>
<:Country holland>)

The filler canada in CanadianCity is considered a <u>default</u> value.

IS-A and inheritance

Specialization relationships imply that procedures and fillers of more general frame are applicable to more specific frame: inheritance.

For example, instances of MahoganyCoffeeTable will inherit the procedure from Table (via CoffeeTable)

(CoffeeTable
<:**IS-A** Table> ...)

(MahoganyCoffeeTable
<:**IS-A** CoffeeTable> ...)

Similarly, default values are inheritable, so that Clyde inherits a colour from RoyalElephant, not Elephant

(Elephant
<:**IS-A** Mammal>
<:Colour gray> ...)

(RoyalElephant
<:**IS-A** Elephant>
<:Colour white>)

(clyde
<:**INSTANCE-OF** RoyalElephant>)

Reasoning with frames

Basic (local) reasoning goes like this:

1. user instantiates a frame, i.e., declares that an object or situation exists
2. slot fillers are inherited where possible
3. inherited **IF-ADDED** procedures are run, causing more frames to be instantiated and slots to be filled.

If the user or any procedure requires the filler of a slot then:

1. if there is a filler, it is used
2. otherwise, an inherited **IF-NEEDED** procedure is run, potentially causing additional actions

Globally:

- make frames be major situations or object-types you need to flesh out
- express constraints between slots as **IF-NEEDED** and **IF-ADDED** procedures
- fill in default values when known

$\Rightarrow$ like a fancy, semi-symbolic spreadsheet

Planning a trip

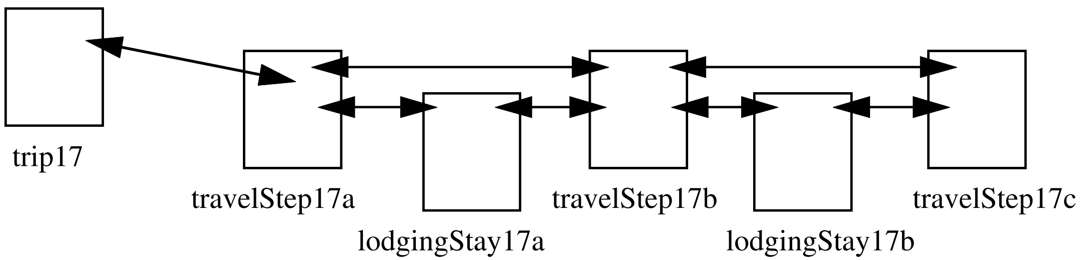
A simple example: a frame system to assist in travel planning (and possibly documentation – automatically generate forms)

Basic structure (main frame types):

- a Trip will be a sequence of TravelSteps

  these will be linked together by slots

- a TravelStep will usually terminate in a LodgingStay (except the last, or one with two travels on one day)
  - a LodgingStay will point to its arriving TravelStep and departing TravelStep
  - TravelSteps will indicate the LodgingStays of their origin and destination



(trip17
<:**INSTANCE-OF** Trip>
<:FirstStep travelStep17a>
<:Traveler ronB> ...)

Parts of a trip

TravelSteps and LodgingStays share some properties (e.g., :BeginDate, :EndDate, :Cost, :PaymentMethod), so we might create a more general category as the parent frame for both of them:

```
(Trip
    <:FirstStep TravelStep>
    <:Traveler  Person>
    <:BeginDate  Date>
    <:TotalCost Price> ...)
```

```
(TripPart
    <:BeginDate>
    <:EndDate>
    <:Cost>
    <:PaymentMethod> …)
```

```
(TravelStep
    <:IS-A  TripPart>
    <:Means>
    <:Origin>   <:Destination>
    <:NextStep>   <:PreviousStep>
    <:DepartureTime>   <:ArrivalTime>
    <:OriginLodgingStay>
    <:DestinationLodgingStay> …)
```

```
(LodgingStay
    <:IS-A  TripPart>
    <:ArrivingTravelStep>
    <:DepartingTravelStep>
    <:City>
    <:LodgingPlace> …)
```

Travel defaults and procedures

Embellish frames with defaults and procedures

```
(TravelStep
    <:Means  airplane> ...)

(TripPart
    <:PaymentMethod  visaCard> ...)

(TravelStep
    <:Origin [IF-NEEDED {if no SELF:PreviousStep then newark}]>)
(Trip
    <:TotalCost
       [IF-NEEDED
         { x←SELF:FirstStep;
           result←0;
           repeat
            {  if  exists x:NextStep
               then
                    { result←result + x:Cost +
                              x:DestinationLodgingStay:Cost;
                      x←x:NextStep }
               else  return  result+x:Cost }}]>)
```

assume this is 0 if there is no LodgingStay

Program notation (for an imaginary language):

- SELF is the current frame being processed
- if x refers to an individual frame, and y to a slot, then xy refers to the filler of the slot

More attached procedures

```
(TravelStep
      <:NextStep
         [IF-ADDED
           {if  SELF:EndDate ≠ SELF:NextStep:BeginDate
            then
                 SELF:DestinationLodgingStay ←
                      SELF:NextStep:OriginLodgingStay ←
                           create new LodgingStay
                                with :BeginDate = SELF:EndDate
                                and with :EndDate = SELF:NextStep:BeginDate
                                and with :ArrivingTravelStep = SELF
                                and with :DepartingTravelStep = SELF:NextStep
                …}]>
  …)
```

Note: default :City of LodgingStay, etc. can also be calculated:

```
(LodgingStay
      <:City [IF-NEEDED  {SELF:ArrivingTravelStep:Destination}]…>   ...)
```

Frames in action

Propose a trip to Toronto on Dec. 21, returning Dec. 22

(trip18
<:**INSTANCE-OF** Trip>
<:FirstStep travelStep18a>)

the first thing to do is to create the trip and the first step

(travelStep18a
<:**INSTANCE-OF** TravelStep>
<:BeginDate 12/21/98>
<:EndDate 12/21/98>
<:Means>
<:Origin>
<:Destination toronto>
<:NextStep> <:PreviousStep>
<:DepartureTime> <:ArrivalTime>)

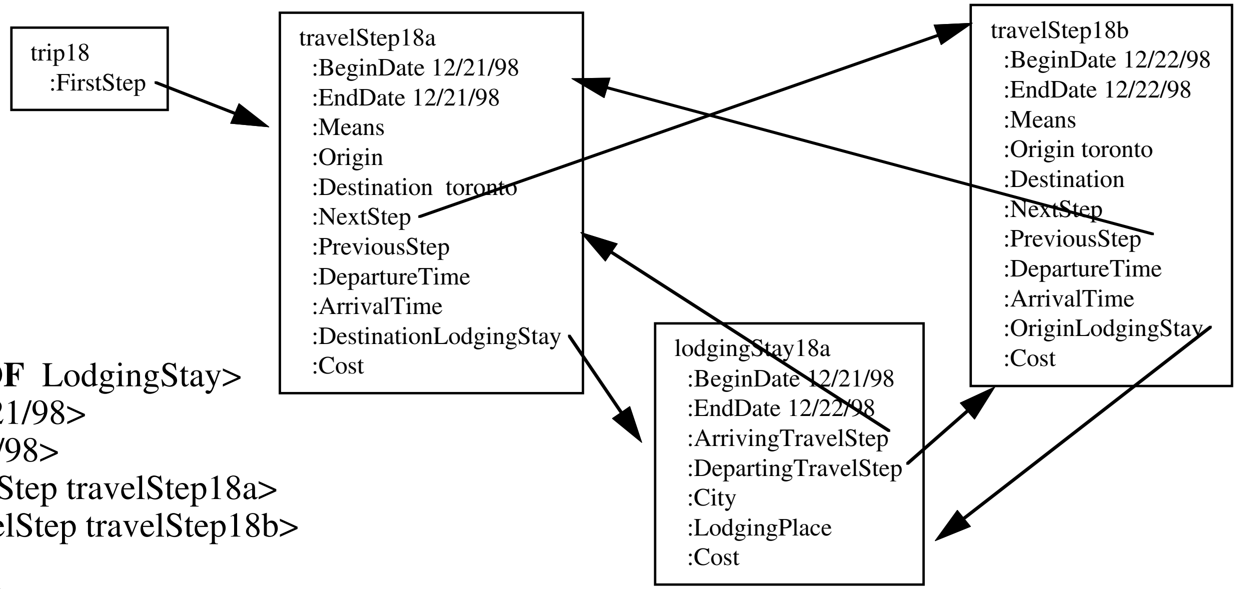
(travelStep18b
<:**INSTANCE-OF** TravelStep>
<:BeginDate 12/22/98>
<:EndDate 12/22/98>
<:Means>
<:Origin toronto>
<:Destination>
<:NextStep>
<:PreviousStep travelStep18a>
<:DepartureTime> <:ArrivalTime>)

the next thing to do is to create the second step and link it to the first by changing the :NextStep

(travelStep18a
<:NextStep travelStep18b>)

Triggering procedures

IF-ADDED on :NextStep then creates a LodgingStay:



(lodgingStay18a
<:**INSTANCE-OF** LodgingStay>
<:BeginDate 12/21/98>
<:EndDate 12/22/98>
<:ArrivingTravelStep travelStep18a>
<:DepartingTravelStep travelStep18b>
<:City>
<:LodgingPlace>)

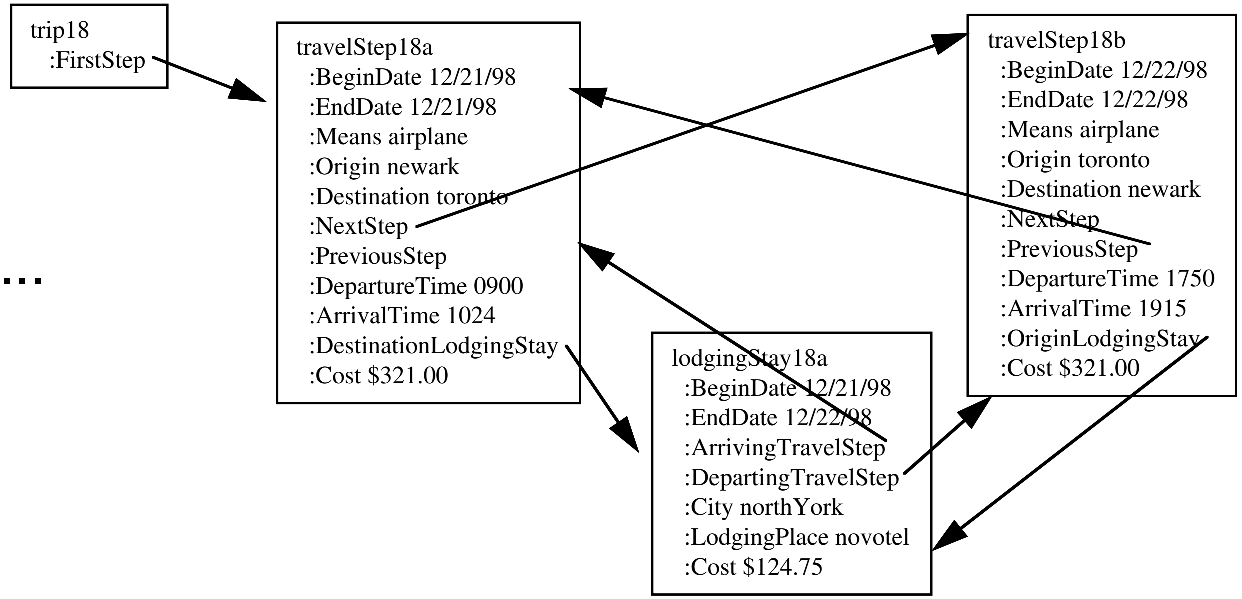
If requested, **IF-NEEDED** can provide :City for lodgingStay18a (toronto)

which could then be overridden by hand, if necessary
(e.g. usually stay in North York, not Toronto)

Similarly, apply default for :Means and default calc for :Origin

Finding the cost of the trip

So far...



Finally, we can use :TotalCost **IF-NEEDED** procedure (see above) to calculate the total cost of the trip:

- result← 0, x←travelStep18a, x:NextStep=travelStep18b
- result←0+$321.00+$124.75; x← travelStep18b, x:NextStep=NIL
- return: result=$445.75+$321.00 = $766.75

Using the formalism

Main purpose of the above: embellish a sketchy description with defaults, implied values

- maintain consistency
- use computed values to
  1. allow derived properties to look explicit
  2. avoid up front, potentially unneeded computation

Monitoring

- hook to a DB, watch for changes in values
- like an ES somewhat, but monitors are more object-centered, inherited

Scripts for story understanding

generate expectations (e.g., restaurant)

Real, Minsky-like commonsense reasoning

- local cues $\Rightarrow$ potentially relevant frames $\Rightarrow$ further expectations
- look to match expectations ; mismatch $\Rightarrow$ "differential diagnosis"

Extensions

1. Types of procedures
   - **IF-REMOVED**

     e.g., remove TravelStep ⇒ remove LodgingStay
   - “servants” and “demons”

     flexible “pushing” and “pulling” of data
2. Slots
   - multiple fillers
   - “facets” – more than just defaults and fillers
     - [**REQUIRE** <class>] (or procedure)
     - **PREFER** – useful if conflicting fillers
3. Metaframes

   (CanadianCity <:**INSTANCE-OF** GeographicalCityType> …)

   (GeographicalCityType <:**IS-A** CityType>
   <:AveragePopulation NonNegativeNumber> …)
4. Frames as actions (“scripts”)

Object-oriented programming

Somewhat in the manner of production systems, specifying problems with frames can easily slide into a style of *programming*, rather than a declarative object-oriented modeling of the world

- note that direction of procedures (pushing/pulling) is explicitly specified

  not declarative

This drifts close to conventional object-oriented programming (developed concurrently).

- same advantages:
  - definition by specialization
  - localization of control
  - encapsulation
  - etc.
- main difference:
  - frames: centralized, conventional control regime (instantiate/ inherit/trigger)
  - object-oriented programming: objects acting as small, independent agents sending each other messages

9.

Structured Descriptions

From sentences to objects

As we saw with frames, it useful to shift the focus away from the true *sentences* of an application towards the categories of *objects* in the application and their properties.

In frame systems, this was done *procedurally,* and we concentrated on hierarchies of frames as a way of organizing collections of procedures.

In this section, we look at the categories of objects themselves:

- objects are members of multiple categories

  e.g. a doctor, a wife, a mother of two

- categories of objects can be more or less specific than others

  e.g. a doctor, a professional, a surgeon

- categories of objects can have parts, sometimes in multiples

  e.g. books have titles, tables have legs

- the relation among the parts of an object can be critical in its being a member of a category

  e.g. a stack vs. a pile of bricks

Noun phrases

In FOL, all categories and properties of objects are represented by atomic predicates.

- In some cases, these correspond to simple *nouns* in English such as Person or City.
- In other cases, the predicates seem to be more like *noun phrases* such as MarriedPerson or CanadianCity or AnimalWithFourLegs.

Intuitively, these predicates have an internal structure and connections to other predicates.

e.g. A married person must be a person.

These connections hold by *definition* (by virtue of what the predicates themselves mean), not by virtue of the facts we believe about the world.

In FOL, there is no way to break apart a predicate to see how it is formed from other predicates.

Here we will examine a logic that allows us to have both atomic and non-atomic predicates: a <u>description logic</u>

Concepts, roles, constants

In a description logic, there are sentences that will be true or false (as in FOL).

In addition, there are three sorts of expressions that act like nouns and noun phrases in English:

- concepts are like category nouns — Dog, Teenager, GraduateStudent
- roles are like relational nouns — :Age, :Parent, :AreaOfStudy
- constants are like proper nouns — johnSmith, chair128

These correspond to unary predicates, binary predicates and constants (respectively) in FOL.

See also: generic frames, slots, and individual frames.
However: roles can have multiple fillers.

However, unlike in FOL, concepts need not be atomic and can have semantic relationships to each other.

roles will remain atomic (for now)

The symbols of DL

Three types of non-logical symbols:

- atomic concepts:

  Dog, Teenager, GraduateStudent

  We include a distinguished concept: Thing

- roles: (all are atomic)

  :Age, :Parent, :AreaOfStudy

- constants:

  johnSmith, chair128

Four types of logical symbols:

- punctuation: [,], (,)
- positive integers: 1, 2, 3, ...
- concept-forming operators: `ALL`, `EXISTS`, `FILLS`, `AND`
- connectives: $\sqsubseteq$, $\doteq$, and $\rightarrow$

The syntax of DL

The set of concepts is the least set satisfying:

- Every atomic concept is a concept.
- If r is a role and d is a concept, then [ALL r d] is a concept.
- If r is a role and n is an integer, then [EXISTS n r] is a concept.
- If r is a role and c is a constant, then [FILLS r c] is a concept.
- If d_1, ..., d_k are concepts, then so is [AND d_1, ..., d_k].

Three types of sentences in DL:

- If d and e are concepts, then ($d \sqsubseteq e$) is a sentence.
- if d and e are concepts, then ($d \doteq e$) is a sentence.
- If d is a concept and c is a constant, then ($c \rightarrow d$) is a sentence.

The meaning of concepts

Constants stand for individuals, concepts for sets of individuals, and roles for binary relations.

The meaning of a complex concept is derived from the meaning of its parts the same way a noun phrases is:

- [`EXISTS` *n r*] describes those individuals that stand in relation *r* to at least *n* other individuals
- [`FILLS` *r c*] describes those individuals that stand in the relation *r* to the individual denoted by *c*
- [`ALL` *r d*] describes those individuals that stand in relation *r* only to individuals that are described by *d*
- [`AND` d_1 ... d_k] describes those individuals that are described by all of the d_i.

For example:

"a company with at least 7 directors, whose managers are all women with PhDs, and whose min salary is $24/hr"

```
[AND Company
     [EXISTS 7 :Director]
     [ALL :Manager [AND Woman
                        [FILLS :Degree phD]]]
     [FILLS :MinSalary $24.00/hour]]
```

A DL knowledge base

A DL knowledge base is a set of DL sentences serving mainly to

- give names to definitions

  e.g. (FatherOfDaughters $\doteq$
  [AND Male [EXISTS 1 :Child]
  [ALL :Child Female]])

  "A FatherOfDaughters is precisely a male with at least one child and all of whose children are female"

- give names to partial definitions

  e.g. (Dog $\sqsubseteq$ [AND Mammal Pet
  CarnivorousAnimal
  [FILLS :VoiceCall barking]])

  gives necessary but not sufficient conditions

  "A dog is among other things a mammal that is a pet and a carnivorous animal whose voice call includes barking"

- assert properties of individuals

  e.g. (joe $\rightarrow$
  [AND FatherOfDaughters Surgeon]])

  "Joe is a FatherOfDaughters and a Surgeon"

Other types of DL sentences are typically not used in a KB.

e.g. ([AND Rational Animal] $\doteq$ [AND Featherless Biped])

Formal semantics

Interpretation $\mathfrak{I} = \langle D, I \rangle$ as in FOL, where

- for every constant c, $I[c] \in D$
- for every atomic concept a, $I[a] \subseteq D$
- for every role r, $I[r] \subseteq D \times D$

We then extend the interpretation to all concepts as subsets of the domain as follows:

- $I[\text{Thing}] = D$
- $I[[\texttt{ALL}\ r\ d]] = \{x \in D \mid$ for any y, if $\langle x,y \rangle \in I[r]$ then $y \in I[d]\}$
- $I[[\texttt{EXISTS}\ n\ r]] = \{x \in D \mid$ there are at least n y such that $\langle x,y \rangle \in I[r]\}$
- $I[[\texttt{FILLS}\ r\ c]] = \{x \in D \mid \langle x, I[c] \rangle \in I[r]\}$
- $I[[\texttt{AND}\ d_1 \ldots d_k]] = I[d_1] \cap \ldots \cap I[d_k]$

A sentence of DL will then be <u>true</u> or false as follows:

- $\mathfrak{I} \models (d \sqsubseteq e)$ iff $I[d] \subseteq I[e]$
- $\mathfrak{I} \models (d \doteq e)$ iff $I[d] = I[e]$
- $\mathfrak{I} \models (c \rightarrow e)$ iff $I[c] \in I[e]$

Entailment and reasoning

Entailment in DL is defined as in FOL:

A set of DL sentences S <u>entails</u> a sentence α (which we write $S \models \alpha$) iff

for every $\mathfrak{I}$, if $\mathfrak{I} \models S$ then $\mathfrak{I} \models \alpha$

A sentence is <u>valid</u> iff it is entailed by the empty set.

Given a KB consisting of DL sentences, there are two basic sorts of reasoning we consider:

1. determining if KB $\models (c \rightarrow e)$

   whether a named individual <u>satisfies</u> a certain description

2. determining if KB $\models (d \sqsubseteq e)$

   whether one description is <u>subsumed</u> by another

the other case, KB $\models (d \doteq e)$ reduces to

KB $\models (d \sqsubseteq e)$ and KB $\models (d \sqsubseteq e)$

Entailment vs. validity

In some cases, an entailment will hold because the sentence in question is valid.

- ([AND Doctor Female] $\sqsubseteq$ Doctor)
- ([FILLS :Child sue] $\sqsubseteq$ [EXISTS 1 :Child])
- (john $\rightarrow$ [ALL :Hobby Thing])

But in most other cases, the entailment depends on the sentences in the KB.

For example,

([AND Surgeon Female] $\sqsubseteq$ Doctor)

is not valid.

But it is entailed by a KB that contains

(Surgeon $\doteq$ [AND Specialist [FILLS :Specialty surgery]])

(Specialist $\sqsubseteq$ Doctor)

Computing subsumption

We begin with computing subsumption, that is, determining whether or not KB $\models (d \sqsubseteq e)$.

and therefore whether $d \doteq e$

Some simplifications to the KB:

- we can remove the $(c \rightarrow d)$ assertions from the KB
- we can replace $(d \sqsubseteq e)$ in KB by $(d \doteq [\texttt{AND}\ e\ a])$, where a is a new atomic concept
- we assume that in the KB for each $(d \doteq e)$, the d is atomic and appears only once on the LHS
- we assume that the definitions in the KB are acyclic

  vs. cyclic $(d \doteq [\texttt{AND}\ e\ f])$, $(e \doteq [\texttt{AND}\ d\ g])$

Under these assumptions, it is sufficient to do the following:

- <u>normalization</u>: using the definitions in the KB, put d and e into a special normal form, d' and e'
- <u>structure matching</u>: determine if each part of e' is matched by a part of d'.

Normalization

Repeatedly apply the following operations to the two concepts:

- expand a definition: replace an atomic concept by its KB definition
- flatten an AND concept:

  [AND ... [AND $d\ e\ f$] ...] $\Rightarrow$ [AND ... $d\ e\ f$ ...]

- combine the ALL operations with the same role:

  [AND ... [ALL $r\ d$] ... [ALL $r\ e$] ...] $\Rightarrow$ [AND ... [ALL r [AND $d\ e$]] ...]

- combine the EXISTS operations with the same role:

  [AND ... [EXISTS $n_1\ r$] ... [EXISTS $n_2\ r$] ...] $\Rightarrow$
  [AND ... [EXISTS $n\ r$] ...] (where n =Max(n_1,n_2))

- remove a vacuous concept: Thing, [ALL r Thing], [AND]
- remove a duplicate expression

In the end, we end up with a normalized concept of the following form

atomic

[AND a_1 ... a_i
[FILLS $r_1\ c_1$] ... [FILLS $r_j\ c_j$]
[EXISTS $n_1\ s_1$] ... [EXISTS $n_k\ s_k$]
[ALL $t_1\ e_1$] ... [ALL $t_m\ e_m$]]

unique roles

normalized

Normalization example

```
[AND  Person
      [ALL :Friend Doctor]
      [EXISTS 1 :Accountant]
      [ALL :Accountant [EXISTS 1 :Degree]]
      [ALL :Friend Rich]
      [ALL :Accountant [AND Lawyer [EXISTS 2 :Degree]]]]
```

⇓

```
[AND  Person
      [EXISTS 1 :Accountant]
      [ALL :Friend [AND Rich Doctor]]
      [ALL :Accountant [AND Lawyer [EXISTS 1 :Degree] [EXISTS 2 :Degree]]]]
```

⇓

```
[AND  Person
      [EXISTS 1 :Accountant]
      [ALL :Friend [AND Rich Doctor]]
      [ALL :Accountant [AND Lawyer [EXISTS 2 :Degree]]]]
```

Structure matching

Once we have replaced atomic concepts by their definitions, we no longer need to use the KB.

To see if a normalized concept [AND e_1 .... e_m] subsumes a normalized concept [AND d_1 ... d_n], we do the following:

For each component e_j, check that there is a matching component d_i, where

- if e_j is atomic or [FILLS r c], then d_i must be identical to it;
- if e_j = [EXISTS 1 r], then d_i must be [EXISTS n r] or [FILLS r c];
- if e_j = [EXISTS n r] where n >1, then d_i must be of the form [EXISTS m r] where $m \geq n$;
- if e_j = [ALL r e'], then d_i must be [ALL r d'], where recursively e' subsumes d'.

In other words, for every part of the more general concept, there must be a corresponding part in the more specific one.

It can be shown that this procedure is sound and complete: it returns YES iff KB $\models (d \sqsubseteq e)$.

Structure matching example

[AND Person
[FILLS :Age 27]
[EXISTS 2 :Child]
[EXISTS 1 :Spouse]
[ALL :Friend [AND Doctor Rich]]]

[AND Person
Female
[FILLS :Age 27]
[EXISTS 3 :Child]
[FILLS :Spouse jack]
[FILLS :Mother sue]
[ALL :Friend [AND Doctor
[FILLS :Specialty surgery]
Rich]]]

not needed

Computing satisfaction

To determine if KB $\models (c \rightarrow e)$, we use the following procedure:

1. find the most specific concept d such that KB $\models (c \rightarrow d)$
2. determine whether or not KB $\models (d \sqsubseteq e)$, as before.

To a first approximation, the d we need is the AND of every d_i such that $(c \rightarrow d_i) \in$ KB.

However, this can miss some inferences!

Suppose the KB contains

(joe → Person)
(canCorp → [AND Company
[ALL :Manager Canadian]
[FILLS :Manager joe]]

then the KB $\models$ (joe → Canadian).

To find the d, a more complex procedure is used that *propagates* constraints from one individual (canCorp) to another (joe).

The individuals we need to consider need not be named by constants; they can be individuals that arise from EXISTS (like Skolem constants).

Taxonomies

Two common sorts of queries in a DL system:

- given a query concept q, find all constants c such that KB $\models (c \rightarrow q)$

  e.g. q is [AND Stock FallingPrice MyHolding]   might want to trigger a procedure for each such c

- given a query constant c, find all *atomic* concepts a such that KB $\models (c \rightarrow a)$

We can exploit the fact that concepts tend to be structured hierarchically to answer queries like these more efficiently.

Taxonomies arise naturally out of a DL KB:

- the nodes are the atomic concepts that appear on the LHS of a sentence ($a \sqsubseteq d$) or ($a \doteq d$) in the KB
- there is an edge from a_i to a_j if ($a_i \sqsubseteq a_j$) is entailed and there is no distinct a_k such that ($a_i \sqsubseteq a_k$) and ($a_k \sqsubseteq a_j$).

  can link every constant c to the most specific atomic concepts a in the taxonomy such that KB $\models (c \rightarrow a)$

Positioning a new atom in a taxonomy is called classification

Computing classification

Consider adding ($a_{new} \doteq d$) to the KB.

- find S, the <u>most specific subsumers</u> of d: the atoms a such that KB $\models (d \sqsubseteq a)$, but nothing below a see below
- find G, the <u>most general subsumees</u> of d: the atoms a such that KB $\models (a \sqsubseteq d)$, but nothing above a

  if $S \cap G$ is not empty, then a_{new} is not new
- remove any links from atoms in G to atoms in S
- add links from all the atoms in G to a_{new} and from a_{new} to all the atoms in S
- reorganize the constants:

  for each constant c such that KB $\models (c \rightarrow a)$ for all $a \in S$, but KB $\models (c \rightarrow a)$ for no $a \in G$, and where KB $\models (c \rightarrow d)$, remove links from c to S and put a single link from c to a_{new}.

Adding ($a_{new} \sqsubseteq d$) is similar, but with no subsumees.

Subsumers and subsumees

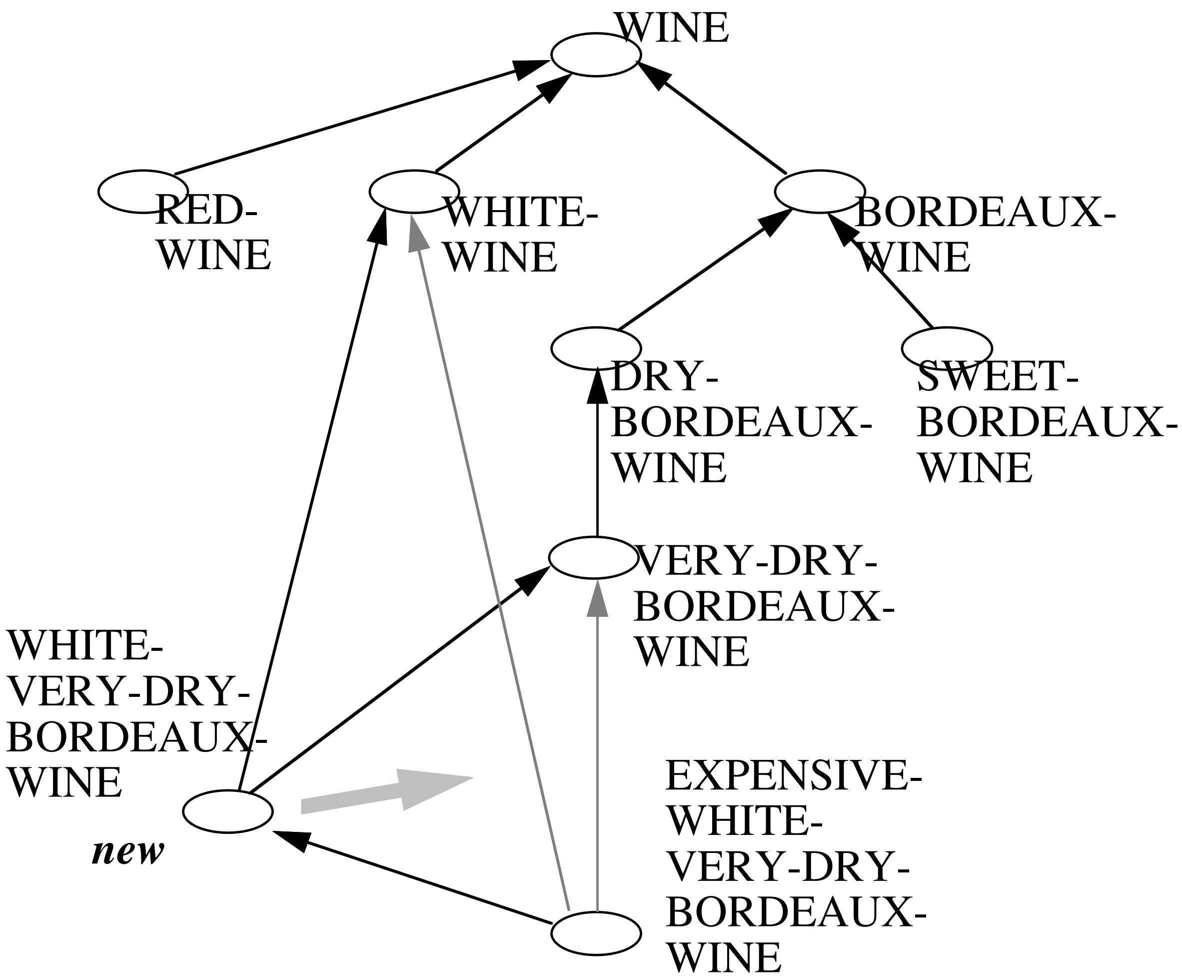
Calculating the most specific subsumers of a concept d:

- Start with $S = \{\text{Thing}\}$.
- Repeatedly do the following:
  - Suppose that some $a \in S$ has at least one child a' just below it in the taxonomy such that KB $\models (d \sqsubseteq a')$.
  - Then remove a from S and replace it by all such children a'.

Calculating the most general subsumees of a concept d:

- Start with G = the most specific subsumers.
- Repeatedly do the following:
  - Suppose that for some $a \in G$, KB $\not\models (a \sqsubseteq d)$.
  - Then remove a from G and replace it by all of its children (or delete it, if there are none).
- Repeatedly delete any element of G that has a parent subsumed by d.

An example of classification



Using the taxonomic structure

Note that classification uses the structure of the taxonomy:

If there is an a' just below a in the taxonomy such that KB $\not\models (d \sqsubseteq a')$, we never look below this a'. If this concept is sufficiently high in the taxonomy (e.g. just below Thing), an entire subtree will be ignored.

Queries can also exploit the structure:

For example, to find the constants described by a concept q, we simply classify q and then look for constants in the part of the taxonomy subtended by q. The rest of the taxonomy not below q is ignored.

This natural structure allows us to build and use very large knowledge bases.

- the time taken will grow linearly with the *depth* of the taxonomy
- we would expect the depth of the taxonomy to grow *logarithmically* with the size of the KB
- under these assumptions, we can handle a KB with thousands or even millions of concepts and constants.

Taxonomies vs. frame hierarchies

The taxonomies in DL look like the **IS-A** hierarchies with frames.

There is a big difference, however:

- in frame systems, the KB designer gets to decide what the fillers of the :IS-A slot will be; the :IS-A hierarchy is constructed manually
- in DL, the taxonomy is completely determined by the meaning of the concepts and the subsumption relation over concepts

For example, a concept such as

[AND Fish [FILLS :Size large]]

must appear in the taxonomy below Fish even if it was first constructed to be given the name Whale. It cannot simply be positioned below Mammal.

To correct our mistake, we need to associate the name with a different concept:

[AND Mammal [FILLS :Size large] ...]

Inheritance and propagation

As in frame hierarchies, atomic concepts in DL inherit properties from concepts higher up in the taxonomy.

For example, if a Doctor has a medical degree, and Surgeon is below Doctor, then a Surgeon must have a medical degree.

This follows from the logic of concepts:

If KB $\models$ (Doctor $\sqsubseteq$ [EXISTS 1 :MedicalDegree])
and KB $\models$ (Surgeon $\sqsubseteq$ Doctor)
then KB $\models$ (Surgeon $\sqsubseteq$ [EXISTS 1 :MedicalDegree])

This is a simple form of *strict* inheritance (*cf*. next chapter)

Also, as noted in computing satisfaction (e.g. with joe and canCorp), adding an assertion like $(c \rightarrow e)$ to a KB can cause other assertions $(c' \rightarrow e')$ to be entailed for other individuals.

This type of propagation is most interesting in applications where membership in classes is monitored and changes are significant.

Extensions to the language

A number of extensions to the DL language have been considered in the literature:

- upper bounds on the number of fillers

  [AND [EXISTS 2 :Child] [AT-MOST 3 :Child]]

  opens the possibility of inconsistent concepts

- sets of individuals: [ALL :Child [ONE-OF wally theodore]]

- relating the role fillers: [SAME-AS :President :CEO]

- qualified number restriction: [EXISTS 2 :Child Female] vs.
  [AND [EXISTS 2 :Child] [ALL :Child Female]]

- complex (non-atomic) roles: [EXISTS 2 [RESTR :Child Female]]

  [ALL [RESTR :Child Female] Married] vs.
  [ALL :Child [AND Female Married]]

Each of these extensions adds extra complexity to the problem of calculating subsumption.

This topic will be explored for RESTR in Chapter 16.

Some applications

Like production systems, description logics have been used in a number of sorts of applications:

- interface to a DB

  relational DB, but DL can provide a nice higher level view of the data based on objects

- working memory for a production system

  instead of a having rules to reason about a taxonomy and inheritance of properties, this part of the reasoning can come from a DL system

- assertion and classification for monitoring

  incremental change to KB can be monitored with certain atomic concepts declared "critical"

- contradiction detection in configuration

  for a DL that allows contradictory concepts, can alert the user when these are detected. This works well for incremental construction of a concept representing e.g. a configuration of a computer.

10.

Inheritance

Hierarchy and inheritance

As we noticed with both frames and description logics, hierarchy or taxonomy is a natural way to view the world

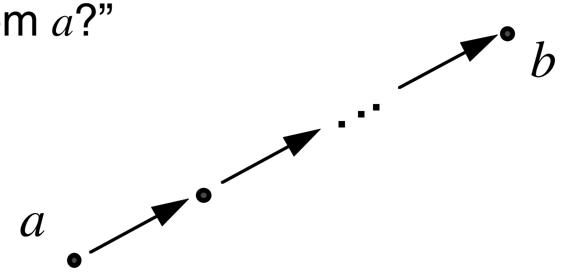
importance of *abstraction* in remembering and reasoning

- groups of things share properties in the world
- do not have to repeat representations

  e.g. sufficient to say that "elephants are mammals" to know a lot about them

Inheritance is the result of transitivity reasoning over paths in a network

- for strict networks, *modus ponens* (if-then reasoning) in graphical form
- "does a inherit from b?" is the same as "is b in the transitive closure of :**IS-A** (or subsumption) from a?"

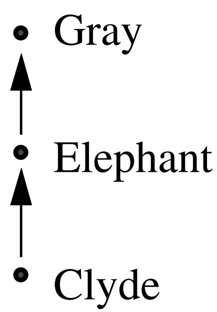


graphically, is there a path of :**IS-A** connections from a to b?

Path-based reasoning

Focus just on inheritance and transitivity

- many interesting considerations in looking just at where information comes from in a network representation
- abstract frames/descriptions, and properties into <u>nodes</u> in graphs, and just look at reasoning with paths and the conclusions they lead us to



note the translation of property, Gray, and the constant Clyde into a node

- <u>edges</u> in the network: Clyde·Elephant, Elephant·Gray
- <u>paths</u> included in this network: edges plus {Clyde·Elephant·Gray}

    in general, a path is a sequence of 1 or more edges

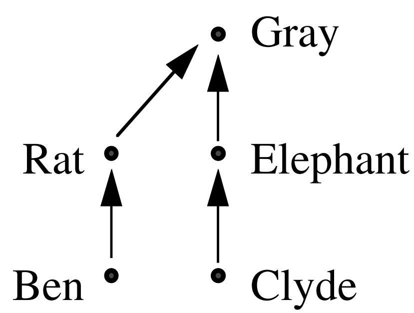
- <u>conclusions</u> supported by the paths:

$$\text{Clyde} \rightarrow \text{Elephant}; \quad \text{Elephant} \rightarrow \text{Gray}; \quad \text{Clyde} \rightarrow \text{Gray}$$

Inheritance networks

(1) Strict inheritance in trees

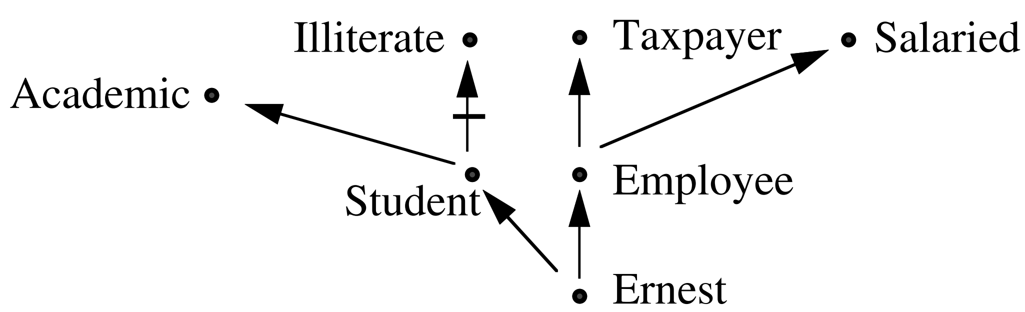
- as in description logics
- conclusions produced by complete transitive closure on all paths (any traversal procedure will do); all reachable nodes are implied



(2) Strict inheritance in DAGs

- as in DL's with multiple `AND` parents (= multiple inheritance)
- same as above: all conclusions you can reach by any paths are supported

Note: negative edge from Student: "is not a"

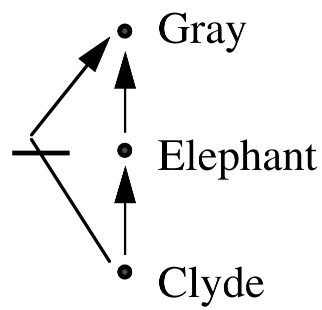


Inheritance with defeasibility

(3) Defeasible inheritance

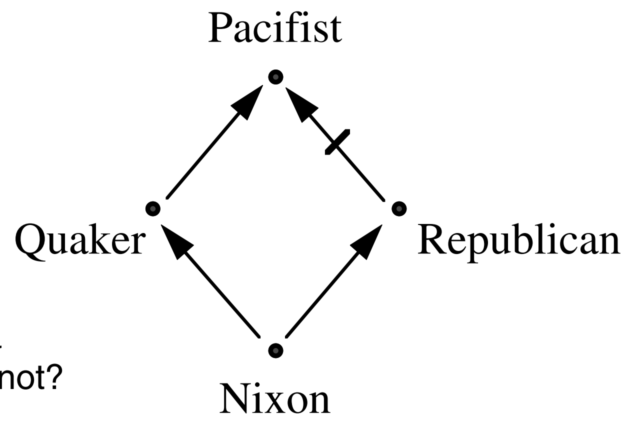
- as in frame systems
- inherited properties do not always hold, and can be *overridden* (defeated)
- conclusions determined by searching upward from "focus node" and selecting first version of property you want

while elephants in general are gray, Clyde is not



A key problem: *ambiguity*

- *credulous* accounts choose arbitrarily
- *skeptical* accounts are more conservative



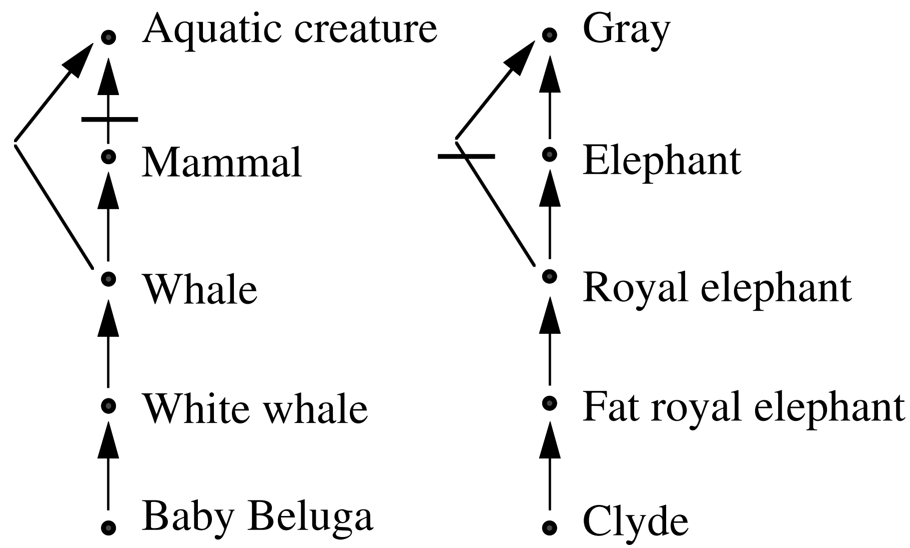
Is Nixon a pacifist or not?

Shortest path heuristic

Defeasible inheritance in DAGs

- links have *polarity* (positive or negative)
- use <u>shortest path heuristic</u> to determine which polarity counts

Intuition: inherit from the most specific subsuming class



- as a result, not all paths count in generating conclusions
  - some are “preempted”
  - but some are “admissible”

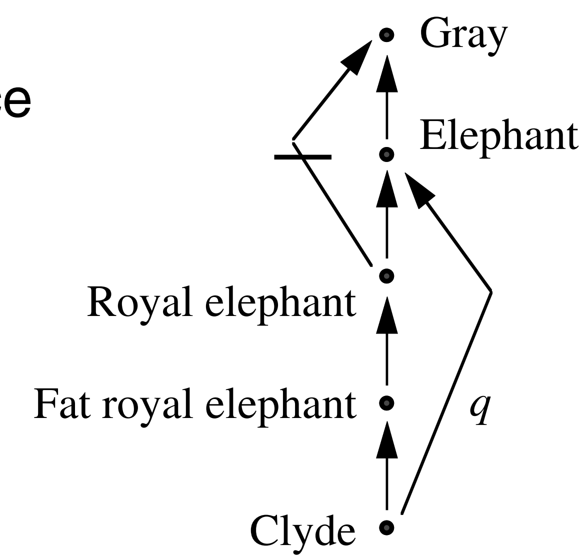
think of paths as *arguments* in support of conclusions

⇒ the inheritance problem = what are the admissible conclusions?

Problems with shortest path

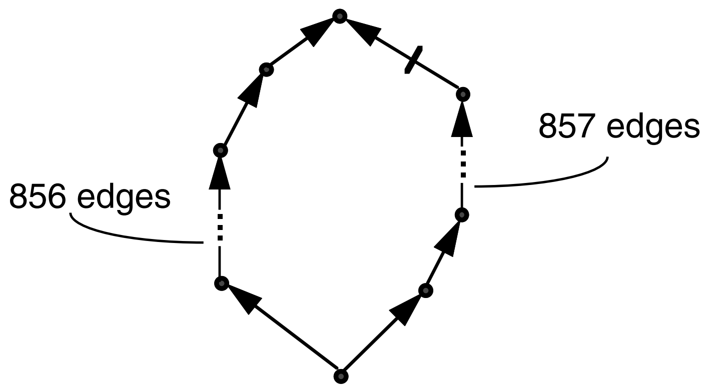
1. Shortest path heuristic produces incorrect answers in the presence of redundant edges (which are already implied!)

the redundant edge *q*, expressing that Clyde is an Elephant changes polarity of conclusion about color



2. Anomalous behavior with ambiguity

adding 2 edges to the left side changes the conclusion!



Why should length be a factor? This network should be ambiguous…

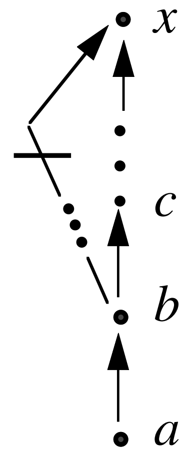
Specificity criteria

Shortest path is a <u>specificity criterion</u> (sometimes called a <u>preemption strategy</u>) which allows us to make admissibility choices among competing paths

- It's not the only possible one
- Consider "*inferential distance*": not linear distance, but topologically based
  - a node a is nearer to node b than to node c if there is a path from a to c through b
  - idea: conclusions from b preempt those from c

  This handles $\text{Clyde} \rightarrow \neg\text{Gray}$ just fine, as well as redundant links

- But what if path from b to c has some of its edges preempted? what if some are redundant?



A formalization (Stein)

An <u>inheritance hierarchy</u> $\Gamma = <V,E>$ is a directed, acyclic graph (DAG) with positive and negative edges, intended to denote "(normally) is-a" and "(normally) is-not-a", respectively.

- positive edges are written $a \cdot x$
- negative edges are written $a \cdot \neg x$

A sequence of edges is a <u>path</u>:

- a <u>positive path</u> is a sequence of one or more positive edges $a \cdot \ldots \cdot x$
- a <u>negative path</u> is a sequence of positive edges followed by a single negative edge $a \cdot \ldots \cdot v \cdot \neg x$

  Note: there are no paths with more than 1 negative edge.

  Also: there might be 0 positive edges.

A path (or <u>argument</u>) supports a <u>conclusion</u>:

- $a \cdot \ldots \cdot x$ supports the conclusion $a \rightarrow x$ (a is an x)
- $a \cdot \ldots \cdot \neg x$ supports $a \not\rightarrow x$ (a is not an x)

  Note: a conclusion may be supported by many arguments

However: not all arguments are equally believable...

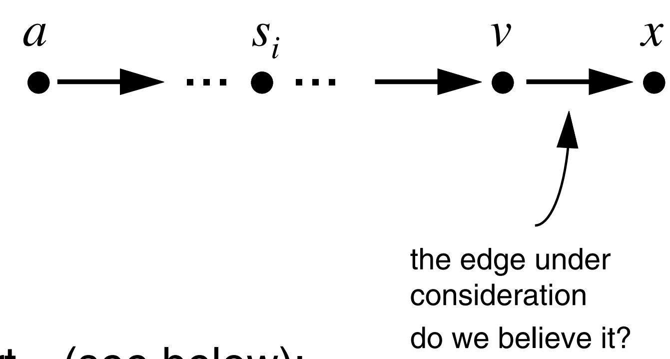
Support and admissibility

Γ supports a path $a \cdot s_1 \cdot \ldots \cdot s_n \cdot (\neg)x$ if the corresponding set of edges $\{a \cdot s_1, \ldots, s_n \cdot (\neg)x\}$ is in E, and the path is admissible according to specificity (see below).

the hierarchy supports a conclusion $a \to x$ (or $a \not\to x$) if it supports some corresponding path

A path is admissible if every edge in it is admissible.

An edge $v \cdot x$ is admissible in Γ wrt a if there is a positive path $a \cdot s_1 \ldots s_n \cdot v$ ($n \geq 0$) in E and



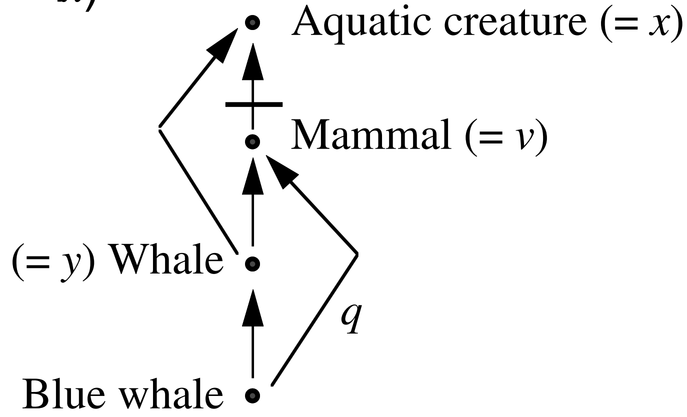
1. each edge in $a \cdot s_1 \ldots s_n \cdot v$ is admissible in Γ wrt a (recursively);
2. no edge in $a \cdot s_1 \ldots s_n \cdot v$ is redundant in Γ wrt a (see below);
3. no intermediate node $a, s_1, \ldots, s_n$ is a preemptor of $v \cdot x$ wrt a (see below).

A negative edge $v \cdot \neg x$ is handled analogously.

Preemption and redundancy

A node y along path $a \cdot ... y ... \cdot v$ is a <u>preemptor</u> of the edge $v \cdot x$ wrt a if $y \cdot \neg x \in E$ (and analogously for $v \cdot \neg x$)

for example, in this figure the node Whale preempts the negative edge from Mammal to Aquatic creature wrt both Whale and Blue whale



A positive edge $b \cdot w$ is <u>redundant</u> in Γ wrt node a if there is some positive path $b \cdot t_1 \ldots t_m \cdot w \in E$ $(m \geq 1)$, for which

1. each edge in $b \cdot t_1 \ldots \cdot t_m$ is admissible in Γ wrt a;
2. there are no c and i such that $c \cdot \neg t_i$ is admissible in Γ wrt a;
3. there is no c such that $c \cdot \neg w$ is admissible in Γ wrt a.

The edge labelled q above is redundant

The definition for a negative edge $b \cdot \neg w$ is analogous

Credulous extensions

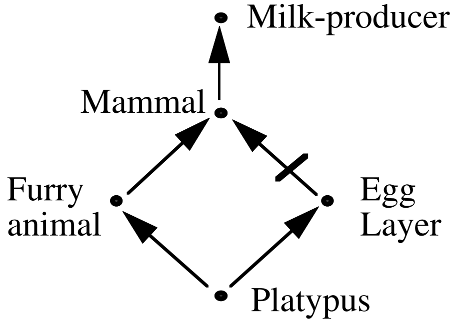
Γ is <u>a-connected</u> iff for every node x in Γ, there is a path from a to x, and for every edge $v \cdot (\neg) x$ in Γ, there is a *positive* path from a to v.

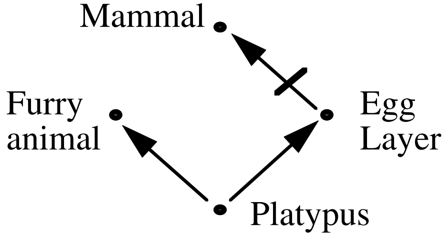
In other words, every node and edge is reachable from a

Γ is (potentially) <u>ambiguous</u> wrt a node a if there is some node $x \in V$ such that both $a \cdot s_1 \ldots s_n \cdot x$ and $a \cdot t_1 \ldots t_m \cdot \neg x$ are paths in Γ

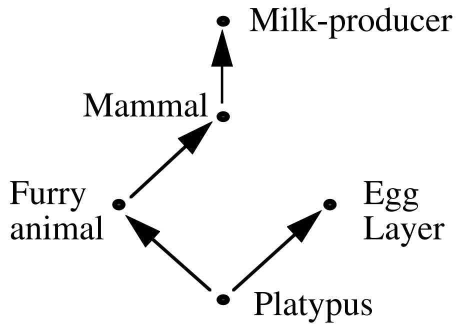
A <u>credulous extension</u> of Γ wrt node a is a maximal unambiguous a-connected subhierarchy of Γ wrt a

If X is a credulous extension of Γ, then adding an edge of Γ to X makes X either ambiguous or not a-connected





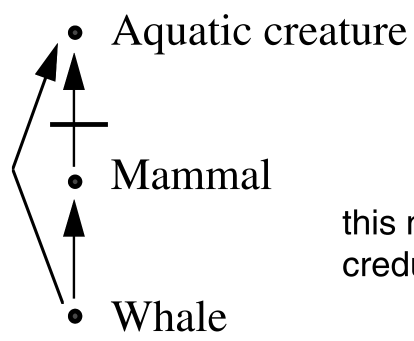
Extension 1



Extension 2

Preferred extensions

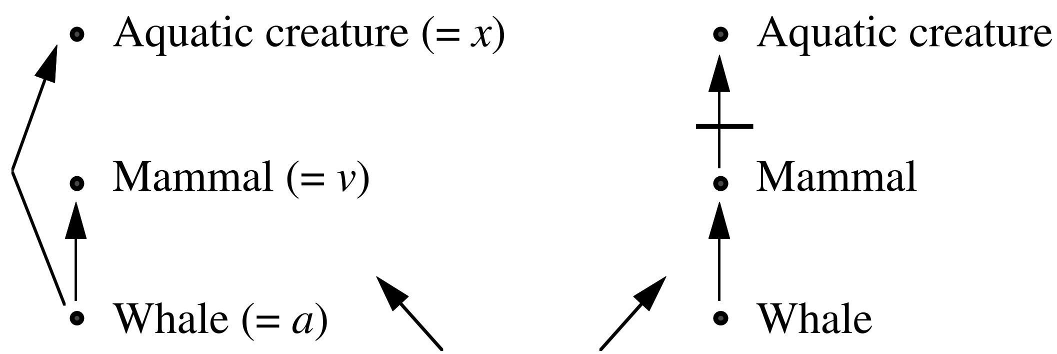
Credulous extensions do not incorporate any notion of admissibility or preemption.



this network has two credulous extensions

Let X and Y be credulous extensions of Γ wrt node a. X is <u>preferred</u> to Y iff there are nodes v and x such that:

- X and Y agree on all edges whose endpoints precede v topologically,
- there is an edge $v \cdot x$ (or $v \cdot \neg x$) that is *inadmissible* in Γ,
- this edge is in Y, but not in X.



is preferred to

A credulous extension is a <u>preferred extension</u> if there is no other extension that is preferred to it.

Subtleties

What to believe?

- “credulous” reasoning: choose a preferred extension and believe all the conclusions supported
- “skeptical” reasoning: believe the conclusions from any path that is supported by all preferred extensions
- “ideally skeptical” reasoning: believe the conclusions that are supported by all preferred extensions

  note: ideally skeptical reasoning cannot be computed in a path-based way (conclusions may be supported by different paths in each extension)

We’ve been doing “upwards” reasoning

- start at a node and see what can be inherited from its ancestor nodes
- there are many variations on this definition; none has emerged as the agreed upon, or “correct” one
- an alternative looks from the top and sees what propagates down

  upwards is more efficient

11.

Defaults

Strictness of FOL

To reason from $P(a)$ to $Q(a)$, need either

- facts about a itself
- universals, e.g. $\forall x(P(x) \supset Q(x))$
  - something that applies to all instances
  - all or nothing!

But most of what we learn about the world is in terms of <u>generics</u>

e.g., encyclopedia entries for ferris wheels, violins, turtles, wildflowers

Properties are not strict for all instances, because

- genetic / manufacturing varieties
  - early ferris wheels
- cases in exceptional circumstances
  - dried wildflowers
- borderline cases
  - toy violins
- imagined cases
  - flying turtles

etc.

Generics vs. universals

✓ Violins have four strings.

vs.

× All violins have four strings.

vs.

? All violins that are not E_1 or E_2 or ... have four strings.

(exceptions usually cannot be enumerated)

Similarly, for general properties of individuals

- Alexander the great: ruthlessness
- Ecuador: exports
- pneumonia: treatment

Goal: be able to say a P is a Q in general, but not necessarily

It is reasonable to conclude $Q(a)$ given $P(a)$,
unless there is a good reason not to

Here: qualitative version (no numbers)

Varieties of defaults (I)

General statements

- prototypical: The prototypical P is a Q.

  Owls hunt at night.

- normal: Under typical circumstances, P's are Q's.

  People work close to where they live.

- statistical: Most P's are Q's.

  The people in the waiting room are growing impatient.

Lack of information to the contrary

- group confidence: All known P's are Q's.

  Natural languages are easy for children to learn.

- familiarity: If a P was not a Q, you would know it.
  - an older brother
  - very unusual individual, situation or event

Varieties of defaults (II)

Conventional

- conversational: Unless I tell you otherwise, a P is a Q

  "There is a gas station two blocks east."
  the default: the gas station is open.

- representational: Unless otherwise indicated, a P is a Q

  the speed limit in a city

Persistence

- inertia: A P is a Q if it used to be a Q.
  - colours of objects
  - locations of parked cars (for a while!)

Here: we will use "Birds fly" as a typical default.

Closed-world assumption

Reiter's observation:

There are usually many more -ve facts than +ve facts!

Example: airline flight guide provides

DirectConnect(cleveland,toronto) DirectConnect(toronto,northBay)
DirectConnect(toronto,winnipeg) ...

but not: ¬DirectConnect(cleveland,northBay)

Conversational default, called CWA:

only +ve facts will be given, relative to some vocabulary

But note: KB $\not\models$ -ve facts (would have to answer: "I don't know")

Proposal: a new version of entailment: KB $\models_c \alpha$ iff KB $\cup$ *Negs* $\models \alpha$

where *Negs* $= \{\neg p \mid p$ atomic and KB $\not\models p\}$

Note: relation to negation as failure

a common pattern:
KB´ = KB $\cup$ Δ

Gives: KB $\models_c$ +ve facts and -ve facts

Properties of CWA

For every α (without quantifiers), KB $\models_c \alpha$ or KB $\models_c \neg\alpha$

Why? Inductive argument:

- immediately true for atomic sentences
- push $\neg$ in, e.g. KB $\models \neg\neg\alpha$ iff KB $\models \alpha$
- KB $\models (\alpha \wedge \beta)$ iff KB $\models \alpha$ and KB $\models \beta$
- Say KB $\not\models_c (\alpha \vee \beta)$. Then KB $\not\models_c \alpha$ and KB $\not\models_c \beta$.
  So by induction, KB $\models_c \neg\alpha$ and KB $\models_c \neg\beta$. Thus, KB $\models_c \neg(\alpha \vee \beta)$.

CWA is an assumption about <u>complete</u> knowledge

never any unknowns, relative to vocabulary

In general, a KB has <u>incomplete</u> knowledge,

e.g. Let KB be $(p \vee q)$. Then KB $\models (p \vee q)$,
but KB $\not\models p$, KB $\not\models \neg p$, KB $\not\models q$, KB $\not\models \neg q$

With CWA, have: If KB $\models_c (\alpha \vee \beta)$, then KB $\models_c \alpha$ or KB $\models_c \beta$.

similar argument to above

Query evaluation

With CWA can reduce queries (without quantifiers) to the atomic case:

$\text{KB} \models_c (\alpha \wedge \beta)$ iff $\text{KB} \models_c \alpha$ and $\text{KB} \models_c \beta$

$\text{KB} \models_c (\alpha \vee \beta)$ iff $\text{KB} \models_c \alpha$ or $\text{KB} \models_c \beta$

$\text{KB} \models_c \neg(\alpha \wedge \beta)$ iff $\text{KB} \models_c \neg\alpha$ or $\text{KB} \models_c \neg\beta$

$\text{KB} \models_c \neg(\alpha \vee \beta)$ iff $\text{KB} \models_c \neg\alpha$ and $\text{KB} \models_c \neg\beta$

$\text{KB} \models_c \neg\neg\alpha$ iff $\text{KB} \models_c \alpha$

reduces to: $\text{KB} \models_c \rho$, where ρ is a literal

If $\text{KB} \cup \textit{Negs}$ is consistent, get $\text{KB} \models_c \neg\alpha$ iff $\text{KB} \not\models_c \alpha$

reduces to: $\text{KB} \models_c p$, where p is atomic

If atoms stored as a table, deciding if $\text{KB} \models_c \alpha$ is like DB-retrieval:

- reduce query to set of atomic queries
- solve atomic queries by table lookup

Different from ordinary logic reasoning (e.g. no reasoning by cases)

Consistency of CWA

If KB is a set of atoms, then KB $\cup$ *Negs* is always consistent

Also works if KB has conjunctions and if KB has only negative disjunctions

If KB contains $(\neg p \vee \neg q)$. Add both $\neg p, \neg q$.

Problem when KB $\models (\alpha \vee \beta)$, but KB $\not\models \alpha$ and KB $\not\models \beta$

e.g. KB = $(p \vee q)$ *Negs* = $\{\neg p, \neg q\}$

KB $\cup$ *Negs* is inconsistent and so for every α, KB $\models_c \alpha$!

Solution: only apply CWA to atoms that are "uncontroversial"

One approach: <u>GCWA</u>

Negs $= \{\neg p \mid$ If KB $\models (p \vee q_1 \vee \ldots \vee q_n)$ then KB $\models (q_1 \vee \ldots \vee q_n)\ \}$

When KB is consistent, get:

– KB $\cup$ *Negs* consistent

– everything derivable is also derivable by CWA

Quantifiers and equality

So far, results do not extend to wffs with quantifiers

can have KB $\not\models_c \forall x.\alpha$ and KB $\not\models_c \neg\forall x.\alpha$

e.g. just because for every t, we have KB $\models_c \neg\mathrm{DirectConnect}(\mathrm{myHome}, t)$
does not mean that KB $\models_c \forall x[\neg\mathrm{DirectConnect}(\mathrm{myHome}, x)]$

But may want to treat KB as providing complete information about what individuals exist

Define: KB $\models_{cd}$ α iff KB $\cup$ *Negs* $\cup$ *Dc* $\models \alpha$ — where the c_i are all the constants appearing in KB (assumed finite)

where *Dc* is <u>domain closure</u>: $\forall x[x{=}c_1 \vee \ldots \vee x{=}c_n]$,

Get: KB $\models_{cd} \exists x.\alpha$ iff KB $\models_{cd}$ $\alpha[x/c]$, for some c appearing in the KB
KB $\models_{cd} \forall x.\alpha$ iff KB $\models_{cd}$ $\alpha[x/c]$, for all c appearing in the KB

Then add: *Un* is <u>unique names</u>: $(c_i \neq c_j)$, for $i \neq j$

Get: KB $\models_{cdu}$ $(c = d)$ iff c and d are the same constant

$\longrightarrow$ full recursive query evaluation

Non-monotonicity

Ordinary entailment is monotonic

If KB $\models \alpha$, then KB* $\models \alpha$, for any KB $\subseteq$ KB*

But CWA entailment is *not* monotonic

Can have KB $\models_c \alpha$, KB $\subseteq$ KB', but KB' $\not\models_c \alpha$

e.g. $\{p\} \models_c \neg q$, but $\{p, q\} \not\models_c \neg q$

Suggests study of <u>non-monotonic reasoning</u>

- start with explicit beliefs
- generate implicit beliefs non-monotonically, taking *defaults* into account
- implicit beliefs may not be uniquely determined (vs. monotonic case)

Will consider three approaches:

- minimal entailment: interpretations that minimize abnormality
- default logic: KB as facts + default rules of inference
- autoepistemic logic: facts that refer to what is/is not believed

Minimizing abnormality

CWA makes the extension of all predicates as small as possible

by adding negated literals

Generalize: do this only for selected predicates

Ab predicates used to talk about typical cases

Example KB: $\text{Bird(chilly)}, \ \neg\text{Flies(chilly)},$
$\text{Bird(tweety)}, \ (\text{chilly} \neq \text{tweety}),$
$\forall x[\text{Bird}(x) \wedge \neg\text{Ab}(x) \supset \text{Flies}(x)]$

All birds that are normal fly

Would like to conclude by default Flies(tweety), but KB $|\neq$ Flies(tweety)

because there is an interpretation $\mathfrak{I}$ where $I[\text{tweety}] \in I[\text{Ab}]$

Solution: consider only interpretations where $I[\text{Ab}]$ is as small as possible, relative to KB

for example: KB requires that $I[\text{chilly}] \in I[\text{Ab}]$

this is sometimes called “circumscription” since we circumscribe the Ab predicate

Generalizes to many Ab_i predicates

Minimal entailment

Given two interps over the same domain, $\mathfrak{I}_1$ and $\mathfrak{I}_2$

$\mathfrak{I}_1 \leq \mathfrak{I}_2$ iff $I_1[\mathrm{Ab}] \subseteq I_2[\mathrm{Ab}]$ for every Ab predicate

$\mathfrak{I}_1 < \mathfrak{I}_2$ iff $\mathfrak{I}_1 \leq \mathfrak{I}_2$ but not $\mathfrak{I}_2 \leq \mathfrak{I}_1$ read: $\mathfrak{I}_1$ is more normal than $\mathfrak{I}_2$

Define a new version of entailment, $\models_{\leq}$ by

KB $\models_{\leq} \alpha$ iff for every $\mathfrak{I}$, if $\mathfrak{I} \models$ KB and no $\mathfrak{I}^* < \mathfrak{I}$ s.t. $\mathfrak{I}^* \models$ KB then $\mathfrak{I} \models \alpha$.

So α must be true in all interps satisfying KB that are *minimal* in abnormalities

Get: KB $\models_{\leq}$ Flies(tweety)

because if interp satisfies KB and is minimal, only *I[chilly]* will be in *I[Ab]*

Minimization need not produce a *unique* interpretation:

Bird(a), Bird(b), $[\neg\mathrm{Flies}(a) \vee \neg\mathrm{Flies}(b)]$ yields two minimal interpretations

KB $\not\models_{\leq}$ Flies(a), KB $\not\models_{\leq}$ Flies(b), KB $\models_{\leq}$ $\mathrm{Flies}(a) \vee \mathrm{Flies}(b)$

Different from the CWA: no inconsistency!

But stronger than GCWA: conclude a or b flies

Fixed and variable predicates

Imagine KB as before + $\forall x[\mathrm{Penguin}(x) \supset \mathrm{Bird}(x) \wedge \neg\mathrm{Flies}(x)]$

Get: KB $\models \forall x[\mathrm{Penguin}(x) \supset \mathrm{Ab}(x)]$

So minimizing Ab also minimizes penguins: KB $\models_{\leq} \forall x \neg \mathrm{Penguin}(x)$

McCarthy's definition: Let **P** and **Q** be sets of predicates

$\mathfrak{I}_1 \leq \mathfrak{I}_2$ iff same domain and

1. $I_1[P] \subseteq I_2[P]$, for every $P \in$ **P** — Ab predicates
2. $I_1[Q] = I_2[Q]$, for every $Q \notin$ **Q** — fixed predicates

so only predicates in **Q** are allowed to vary

Get definition of $\models_{\leq}$ that is parameterized by what is minimized *and* what is allowed to vary

Previous example: minimize Ab, but allow only Flies to vary.

Problems:
- need to decide what to allow to vary
- cannot conclude $\neg\mathrm{Penguin}(\mathrm{tweety})$ by default!

only get default $(\neg\mathrm{Penguin}(\mathrm{tweety}) \supset \mathrm{Flies}(\mathrm{tweety}))$

Default logic

Beliefs as deductive theory

explicit beliefs = axioms

implicit beliefs = theorems = least set closed under inference rules

e.g. If we can prove α and $(\alpha \supset \beta)$, then infer β

Would like to generalize to default rules:

If can prove $\mathrm{Bird}(x)$, but *cannot* prove $\neg\mathrm{Flies}(x)$, then infer $\mathrm{Flies}(x)$.

Problem: how to characterize theorems

cannot write a derivation, since do not know when to apply default rules

no guarantee of unique set of theorems

If cannot infer p, infer q + If cannot infer q, infer p ??

Solution: default logic

no notion of theorem

instead, have <u>extensions</u>: sets of sentences that are "reasonable" beliefs, given explicit facts and default rules

Extensions

Default logic KB uses two components: KB = $\langle F,D \rangle$

- F is a set of sentences (facts)
- D is a set of <u>default rules</u>: triples $\langle \alpha : \beta / \gamma \rangle$ read as

If you can infer α, and β is *consistent*, then infer γ

α: the prerequisite, β: the justification, γ: the conclusion

e.g. $\langle \mathrm{Bird}(\mathrm{tweety}) : \mathrm{Flies}(\mathrm{tweety}) \ / \ \mathrm{Flies}(\mathrm{tweety}) \rangle$

treat $\langle \mathrm{Bird}(x) : \mathrm{Flies}(x) / \mathrm{Flies}(x) \rangle$ as <u>set</u> of rules

Default rules where $\beta = \gamma$ are called <u>normal</u> and write as $\langle \alpha \Rightarrow \beta \rangle$

will see later a reason for wanting non-normal ones

A set of sentences E is an <u>extension</u> of $\langle F,D \rangle$ iff for every sentence π, E satisfies the following:

$\pi \in E$ iff $F \cup \Delta \models \pi$, where $\Delta = \{\gamma \mid \langle \alpha : \beta / \gamma \rangle \in D, \ \alpha \in E, \neg\beta \notin E\}$

So, an extension E is the set of entailments of $F \cup \{\gamma\}$, where the γ are assumptions from D.

to check if E is an extension, guess at Δ and show that it satisfies the above constraint

Example

Suppose KB has

F = Bird(chilly), $\neg$Flies(chilly), Bird(tweety)
D = $\langle$Bird(x) $\Rightarrow$ Flies(x)$\rangle$

then there is a unique extension, where Δ = Flies(tweety)

- This is an extension since tweety is the only t for this Δ such that Bird(t) $\in E$ and $\neg$Flies(t) $\notin E$.
- No other extension, since this applies no matter what Flies(t) assumptions are in Δ.

But in general can have multiple extensions:

F = {Republican(dick), Quaker(dick)} D = { $\langle$Republican(x) $\Rightarrow$ $\neg$Pacifist(x)$\rangle$, $\langle$Quaker(x) $\Rightarrow$ Pacifist(x)$\rangle$ }

Two extensions: E_1 has Δ = $\neg$Pacifist(dick); E_2 has Δ = Pacifist(dick)

Which to believe?

<u>credulous</u>: choose an extension arbitrarily
<u>skeptical</u>: believe what is common to all extensions

Can sometimes use non-normal defaults to avoid conflicts in defaults

$\langle$ Quaker(x) : Pacifist(x) $\wedge\neg$Republican(x) / Pacifist(x) $\rangle$

but then need to consider all possible interactions in defaults!

Unsupported conclusions

Extension tries to eliminate facts that do not result from either F or D.

e.g., we do not want Yellow(tweety) and its entailments in the extension

But the definition has a problem:

Suppose $F = \{\}$ and $D = \langle\, p : \mathrm{True} \,/\, p \,\rangle$.

Then E = entailments of $\{p\}$ is an extension

since $p \in E$ and $\neg\mathrm{True} \notin E$, for above default

However, no good reason to believe p!

Only support for p is default rule, which requires p itself as a prerequisite

So default should have no effect. Want one extension: E = entailments of $\{\}$

Reiter's definition:

For any set S, let $\Gamma(S)$ be the least set containing F, closed under entailment, and satisfying

if $\langle\alpha : \beta \,/\, \gamma\rangle \in D$, $\alpha \in \Gamma(S)$, and $\neg\beta \notin S$, then $\gamma \in \Gamma(S)$.

note: not $\Gamma(S)$

A set E is an <u>extension</u> of $\langle F, D\rangle$ iff $E = \Gamma(E)$.

called a <u>fixed point</u> of the Γ operator

Autoepistemic logic

One disadvantage of default logic is that rules cannot be combined or reasoned about

$\langle \alpha : \beta \,/\, \gamma \rangle \;\twoheadrightarrow\; \langle \alpha : \beta \,/\, (\gamma \vee \delta) \rangle$

Solution: express defaults as *sentences* in an extended language that talks about <u>belief</u> explicitly

for any sentence α, we have another sentence $\mathbf{B}\alpha$

$\mathbf{B}\alpha$ says "I believe α": autoepistemic logic

e.g. $\forall x[\mathrm{Bird}(x) \wedge \neg\mathbf{B}\neg\mathrm{Flies}(x) \supset \mathrm{Flies}(x)]$

All birds fly except those that I believe to not fly =

Any bird not believed to be flightless flies.

No longer expressing defaults using formulas of FOL.

Semantics of belief

These are not sentences of FOL, so what semantics and entailment?

- modal logic of belief provide semantics
- for here: treat $\mathbf{B}\alpha$ as if it were an new atomic wff
- still get entailment: $\forall x[\text{Bird}(x) \wedge \neg\mathbf{B}\neg\text{Flies}(x) \supset \text{Flies}(x) \vee \text{Run}(x)]$

Main property for set of implicit beliefs, E:

1. If $E \models \alpha$ then $\alpha \in E$. (closed under entailment)
2. If $\alpha \in E$ then $\mathbf{B}\alpha \in E$. (positive introspection)
3. If $\alpha \notin E$ then $\neg\mathbf{B}\alpha \in E$. (negative introspection)

Any such set of sentences is called stable

Note: if E contains p but does not contain q, it will contain $\mathbf{B}p$, $\mathbf{BB}p$, $\mathbf{BBB}p$, $\neg\mathbf{B}q$, $\mathbf{B}\neg\mathbf{B}q$, $\mathbf{B}(\mathbf{B}p \wedge \neg\mathbf{B}q)$, *etc*.

Stable expansions

Given KB, possibly containing **B** operators, our implicit beliefs should be a stable set that is minimal.

Moore's definition: A set of sentences E is called a <u>stable expansion</u> of KB iff it satisfies the following:

$\pi \in E$ iff KB $\cup\, \Delta \models \pi$, where $\Delta = \{\mathbf{B}\alpha \mid \alpha \in E\} \cup \{\neg\mathbf{B}\alpha \mid \alpha \notin E\}$

fixed point of another operator

analogous to the extensions of default logic

Example: for KB = $\{$ Bird(chilly), $\neg$Flies(chilly), Bird(tweety),
$\forall x[\text{Bird}(x) \wedge \neg\mathbf{B}\neg\text{Flies}(x) \supset \text{Flies}(x)]$ $\}$

get a unique stable expansion containing Flies(tweety)

As in default logic, stable expansions are not uniquely determined

KB = $\{(\neg\mathbf{B}p \supset q), (\neg\mathbf{B}q \supset p)\}$

2 stable expansions
(one with p, one with q)

KB = $\{(\neg\mathbf{B}p \supset p)\}$ (self-defeating default)

no stable expansions!
so what to believe?

Enumerating stable expansions

Define: A wff is <u>objective</u> if it has no **B** operators

When a KB is propositional, and **B** operators only dominate objective wffs, we can enumerate all stable expansions using the following:

1. Suppose **B**α_1, **B**α_2, ... **B**α_n are all the **B** wffs in KB.
2. Replace some of these by True and the rest by $\neg\mathrm{True}$ in KB and simplify. Call the result KB° (it's objective).

   at most 2^n possible replacements

3. Check that for each α_i,
   - if **B**α_i was replaced by True, then KB° $\models \alpha_i$
   - if **B**α_i was replaced by $\neg\mathrm{True}$, then KB° $\not\models \alpha_i$
4. If yes, then KB° determines a stable expansion.

   entailments of KB° are the objective part

Example enumeration

For KB = { $\text{Bird(chilly)}, \ \neg\text{Flies(chilly)}, \ \text{Bird(tweety)},$
$[\text{Bird(tweety)} \wedge \neg\mathbf{B}\neg\text{Flies(tweety)} \supset \text{Flies(tweety)}],$
$[\text{Bird(chilly)} \wedge \neg\mathbf{B}\neg\text{Flies(chilly)} \supset \text{Flies(chilly)}]$ }

Two **B** wffs: $\mathbf{B}\neg\text{Flies(tweety)}$ and $\mathbf{B}\neg\text{Flies(chilly)}$,
so four replacements to try.

Only one satisfies the required constraint:

$\mathbf{B}\neg\text{Flies(tweety)} \rightarrow \neg\text{True},$
$\mathbf{B}\neg\text{Flies(chilly)} \rightarrow \text{True}$

Resulting KB° has

$(\text{Bird(tweety)} \supset \text{Flies(tweety)})$

and so entails

Flies(tweety)

as desired.

More ungroundedness

Definition of stable expansion may not be strong enough

KB = $\{(\mathbf{B}p \supset p)\}$ has 2 stable expansions:

- one without p and with $\neg\mathbf{B}p$

  corresponds to KB° = $\{\}$

- one with p and $\mathbf{B}p$.

  corresponds to KB° = $\{p\}$

But why should p be believed?

only justification for having p is having $\mathbf{B}p$!

similar to problem with default logic extension

Konolige's definition:

A <u>grounded</u> stable expansion is a stable expansion that is minimal wrt to the set of sentences without **B** operators.

rules out second stable expansion

Other examples suggest that an even stronger definition is required!

can get an equivalence with Reiter's definition of extension in default logic

12.

Vagueness, Uncertainty and Degrees of Belief

Noncategorical statements

Ordinary commonsense knowledge quickly moves away from categorical statements like "a P is *always* (*unequivocably*) a Q"

There are many ways in which we can come to less than categorical information

- things are *usually* (*almost never*, *occasionally*, *seldomly*, *rarely, almost always*) a certain way
- judgments about how good an example something is

  e.g., barely rich, a poor example of a chair, not very tall
- imprecision of sensors

  e.g., the best you can do is to get within +/-10%
- reliability of sources of information

  e.g., "most of the time he's right on the money"
- strength/confidence/trust in generic information or deductive rules

Conclusions will not "follow" in the usual sense

Weakening a universal

There are at least 3 ways a universal like $\forall x\, P(x)$ can be made ro be less categorical:

$\forall x\, P(x)$

strength of quantifier
("95% of birds fly")

statistical interpretation/
probabilistic sentences

applicability of predicate/
degree of membership
("fairly tall")

flexible membership
vague predicates

degree of belief in whole sentence ("80% confidence in this fact")

uncertain knowledge/
subjective probability

Objective probability

Statistical (frequency) view of sentences

<u>objective</u>: does not depend on who is assessing the probability

Always applied to collections

can not assign probabilities to (random) events that are not members of any obvious repeatable sequence:

- ok for "the probability that I will pick a red face card from the deck"
- not ok for "the probability that the Blue Jays will win the World Series this Fall"
- "the probability that Tweety flies is between .9 and .95" is always false (either Tweety flies or not)

Can use probabilities to correspond to English words like "rarely," "likely," "usually"

generalized quantifiers: "most," "many," "few"

For most x, $Q(x)$ vs. For all x, $Q(x)$

The basic postulates

Numbers between 0 and 1 representing frequency of an event in a (large enough) random sample

extremes: 0 = never happens; 1 = always happens

Start with set U of all possible occurrences. An event a is any subset of U.
A probability measure is any function $\boldsymbol{Pr}$ from events to [0,1] satisfying:

- $\boldsymbol{Pr}(U) = 1$.
- If $a_1, \ldots, a_n$ are disjoint events, then $\boldsymbol{Pr}(\cup a_i) = \Sigma\, \boldsymbol{Pr}(a_i)$

Conditioning: the probability of one event may depend on its interaction with others

$\boldsymbol{Pr}(a|b)$ = probability of a, given b = $\boldsymbol{Pr}(a \cap b) \,/\, \boldsymbol{Pr}(b)$

Conditional independence:

event a is judged independent of event b conditional on background knowledge s if knowing that b happened does not affect the probability of a

$\boldsymbol{Pr}(a|s) = \boldsymbol{Pr}(a|b,s)$ (note: CI is symmetric)

Note: without independence, $\boldsymbol{Pr}(a|s)$ and $\boldsymbol{Pr}(a|b,s)$ can be very different.

Some useful consequences

Conjunction:

$$\mathbf{Pr}(ab) \;=\; \mathbf{Pr}(a|b) \cdot \mathbf{Pr}(b)$$

conditionally independent: $\mathbf{Pr}(ab) \;=\; \mathbf{Pr}(a) \cdot \mathbf{Pr}(b)$

Negation:

$$\mathbf{Pr}(\neg s) = 1 - \mathbf{Pr}(s)$$

$$\mathbf{Pr}(\neg s|d) = 1 - \mathbf{Pr}(s|d)$$

If $b_1, b_2, \ldots, b_n$ are pairwise disjoint and exhaust all possibilities, then

$$\mathbf{Pr}(a) \;=\; \sum \mathbf{Pr}(ab_i) \;=\; \sum \mathbf{Pr}(a \mid b_i) \cdot \mathbf{Pr}(b_i)$$

$$\mathbf{Pr}(a \mid c) \;=\; \sum \mathbf{Pr}(ab_i \mid c)$$

Bayes' rule:

$$\mathbf{Pr}(a|b) \;=\; \mathbf{Pr}(a) \cdot \mathbf{Pr}(b|a) \,/\, \mathbf{Pr}(b)$$

if a is a disease and b is a symptom, it is usually easier to estimate numbers on RHS of equation (see below, for subjective probabilities)

Subjective probability

It is reasonable to have non-categorical beliefs even in categorical sentences

- confidence/certainty in a sentence
- "your" probability = *subjective*

Similar to defaults

- move from statistical/group observations to belief about individuals
- but not categorical: how certain am I that Tweety flies?

"Prior probability" $Pr(x|s)$ (s = prior state of information or background knowledge)

"Posterior probability" $Pr(x|E,s)$ (E = new evidence)

Need to *combine evidence* from various sources

how to derive new beliefs from prior beliefs and new evidence?

want explanations; probability is just a summary

From statistics to belief

Would like to go from statistical information (e.g., the probability that a bird chosen at random will fly) to a degree of belief (e.g., how certain are we that this particular bird, Tweety, flies)

Traditional approach is to find a reference class for which we have statistical information and use the statistics for that class to compute an appropriate degree of belief for an individual

Imagine trying to assign a degree of belief to the proposition "Eric (an American male) is tall" given facts like these

A) 20% of American males are tall
B) 25% of Californian males are tall
C) Eric is from California

This is called direct inference

Problem: individuals belong to many classes

- with just A → .2
- A,B,C - prefer more specific → .25
- A,C - no statistics for more specific class → .2?
- B - are Californians a representative sample?

Basic Bayesian approach

Would like a more principled way of calculating subjective probabilities

Assume we have n atomic propositions $p_1,\ldots, p_n$ we care about. A logical interpretation I can be thought of as a specification of which p_i are true and which are false.

Notation: for $n=4$, we use $\langle\neg p_1, p_2, p_3, \neg p_4\rangle$ to mean the interpretation where only p_2 and p_3 are true.

A <u>joint probability distribution</u> J, is a function from interpretations to $[0,1]$ satisfying $\Sigma\, J(I) = 1$ (where $J(I)$ is the degree of belief in the world being as per I).

The <u>degree of belief</u> in any sentence α: $\boldsymbol{Pr}(\alpha) = \sum_{I \models \alpha} J(I)$

Example: $\boldsymbol{Pr}(p_2 \wedge \neg p_4) = J(\langle\neg p_1, p_2, p_3, \neg p_4\rangle) + J(\langle\neg p_1, p_2, \neg p_3, \neg p_4\rangle) + J(\langle p_1, p_2, p_3, \neg p_4\rangle) + J(\langle p_1, p_2, \neg p_3, \neg p_4\rangle).$

Problem with the approach

To calculate the probabilities of arbitrary sentences involving the p_i, we would need to know the full joint distribution function.

For n atomic sentences, this requires knowing 2^n numbers

impractical for all but very small problems

Would like to make plausible assumptions to cut down on what needs to be known.

In the simplest case, all the atomic sentences are independent. This gives us that

$$J(\langle P_1,\ldots,P_n\rangle) = \boldsymbol{Pr}(P_1 \wedge \ldots \wedge P_n) = \prod \boldsymbol{Pr}(P_i) \quad \text{(where } P_i \text{ is either } p_i \text{ or } \neg p_i\text{)}$$

and so only n numbers are needed.

Bu this assumption is too strong. A better assumption:

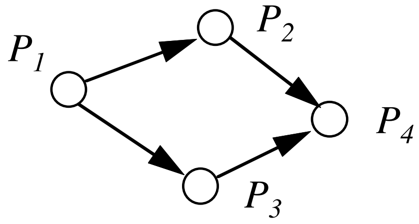
the probability of each P_i only depends on a small number of P_j, and the dependence is acyclic.

Belief networks

Represent all the atoms in a belief network (or Bayes' network).

Assume: $\boxed{J(\langle P_1,\ldots,P_n\rangle) = \prod \mathbf{Pr}(P_i \mid c(P_i)) \quad \text{where } \mathbf{Pr}(c(P_i)) > 0}$ $c(P)$ = parents of node P

Example:



$$J(\langle P_1,P_2,P_3,P_4\rangle) = \mathbf{Pr}(P_1) \cdot \mathbf{Pr}(P_2 \mid P_1) \cdot \mathbf{Pr}(P_3 \mid P_1) \cdot \mathbf{Pr}(P_4 \mid P_2,P_3).$$

So: $J(p_1,\overline{p}_2,p_3,\overline{p}_4) = \mathbf{Pr}(p_1) \cdot \mathbf{Pr}(\overline{p_2} \mid p_1) \cdot \mathbf{Pr}(p_3 \mid p_1) \cdot \mathbf{Pr}(\overline{p_4} \mid \overline{p}_2,p_3)$

$= \mathbf{Pr}(p_1) \cdot [1 - \mathbf{Pr}(p_2 \mid p_1)] \cdot \mathbf{Pr}(p_3 \mid p_1) \cdot [\, 1 - \mathbf{Pr}(p_4 \mid \overline{p}_2,p_3)]$

To fully specify the joint distribution (and therefore probabilities over any subset of the variables), we only need $\mathbf{Pr}(P \mid c(P))$ for every node P.

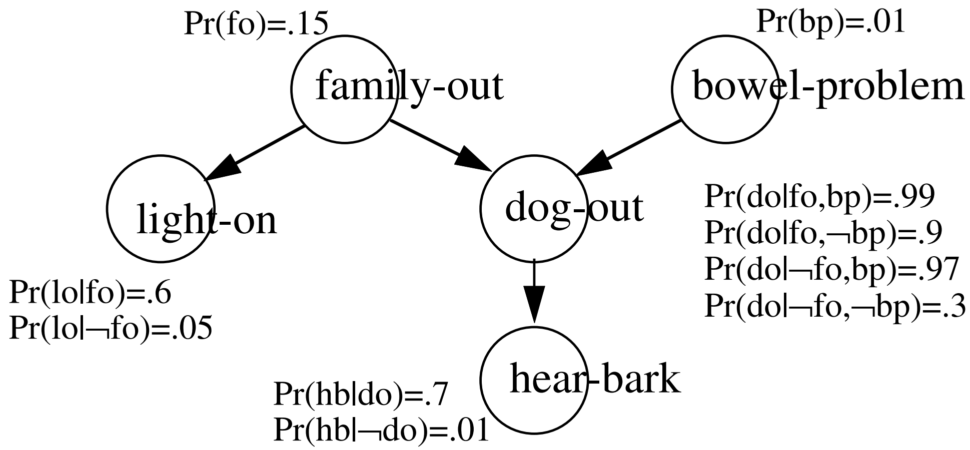
If node P has parents $Q_1, \ldots, Q_m$, then we need to know the values of $\mathbf{Pr}(p \mid q_1,q_2,\ldots q_m)$, $\mathbf{Pr}(p \mid \overline{q}_1,q_2 \ldots q_m)$, $\mathbf{Pr}(p \mid q_1,\overline{q}_2,\ldots q_m)$, ..., $\mathbf{Pr}(p \mid \overline{q}_1,\overline{q}_2,\ldots \overline{q}_m)$.

$n \cdot 2^m$ numbers << 2^n numbers !

Using belief networks

Assign a node to each variable in the domain and draw arrows toward each node P from a select set $c(P)$ of nodes perceived to be "direct causes" of P.

arcs can often be interpreted as causal connections



From the DAG, we get that

$$J(\langle \text{FO, LO, BP, DO, HB} \rangle) =$$
$$\boldsymbol{Pr}(\text{FO}) \times \boldsymbol{Pr}(\text{LO} \mid \text{FO}) \times \boldsymbol{Pr}(\text{BP}) \times \boldsymbol{Pr}(\text{DO} \mid \text{FO, BP}) \times \boldsymbol{Pr}(\text{HB} \mid \text{DO})$$

Using this formula and the 10 numbers above, we can calculate the full joint distribution

Example calculation

Suppose we want to calculate $\boldsymbol{Pr}(\text{fo} \mid \text{lo}, \neg\text{hb})$

$\boldsymbol{Pr}(\text{fo} \mid \text{lo}, \neg\text{hb}) = \boldsymbol{Pr}(\text{fo}, \text{lo}, \neg\text{hb}) \,/\, \boldsymbol{Pr}(\text{lo}, \neg\text{hb})$ where

$\boldsymbol{Pr}(\text{fo}, \text{lo}, \neg\text{hb}) = \Sigma\, J(\langle \text{fo}, \text{lo}, \text{BP}, \text{DO}, \neg\text{hb}\rangle)$ first 4 values below

$\boldsymbol{Pr}(\text{lo}, \neg\text{hb}) = \Sigma\, J(\langle \text{FO}, \text{lo}, \text{BP}, \text{DO}, \neg\text{hb}\rangle)$ all 8 values below

$J(\langle \text{fo},\text{lo},\text{bp},\text{do},\neg\text{hb}\rangle) = .15 \cdot .6 \cdot .01 \cdot .99 \cdot .3 = .0002673 +$
$J(\langle \text{fo},\text{lo},\text{bp},\neg\text{do},\neg\text{hb}\rangle) = .15 \cdot .6 \cdot .01 \cdot .01 \cdot .99 = .00000891 +$
$J(\langle \text{fo},\text{lo},\neg\text{bp},\text{do},\neg\text{hb}\rangle) = .15 \cdot .6 \cdot .99 \cdot .9 \cdot .3 = .024057 +$
$J(\langle \text{fo},\text{lo},\neg\text{bp},\neg\text{do},\neg\text{hb}\rangle) = .15 \cdot .6 \cdot .99 \cdot .1 \cdot .99 = .0088209 +$
$J(\langle \neg\text{fo},\text{lo},\text{bp},\text{do},\neg\text{hb}\rangle) = .85 \cdot .05 \cdot .01 \cdot .97 \cdot .3 = .000123675$
$J(\langle \neg\text{fo},\text{lo},\text{bp},\neg\text{do},\neg\text{hb}\rangle) = .85 \cdot .05 \cdot .01 \cdot .03 \cdot .99 = .0000126225 +$
$J(\langle \neg\text{fo},\text{lo},\neg\text{bp},\text{do},\neg\text{hb}\rangle) = .85 \cdot .05 \cdot .99 \cdot .3 \cdot .3 = .00378675$
$J(\langle \neg\text{fo},\text{lo},\neg\text{bp},\neg\text{do},\neg\text{hb}\rangle) = .85 \cdot .05 \cdot .99 \cdot .7 \cdot .99 = .029157975$

$\boldsymbol{Pr}(\text{fo} \mid \text{lo}, \neg\text{hb}) = .03316 \,/\, .06624 = .5$

Bypassing the full calculation

Often it is possible to calculate some probability values without first calculating the full joint distribution

Example: what is $\mathbf{Pr}(\mathrm{fo} \mid \mathrm{lo})$?

by Bayes rule: $\mathbf{Pr}(\mathrm{fo} \mid \mathrm{lo}) = \overset{\checkmark}{\mathbf{Pr}(\mathrm{lo} \mid \mathrm{fo})} \cdot \overset{\checkmark}{\mathbf{Pr}(\mathrm{fo})} \,/\, \overset{?}{\mathbf{Pr}(\mathrm{lo})}$

but: $\mathbf{Pr}(\mathrm{lo}) = \overset{\checkmark}{\mathbf{Pr}(\mathrm{lo} \mid \mathrm{fo})} \cdot \overset{\checkmark}{\mathbf{Pr}(\mathrm{fo})} + \overset{\checkmark}{\mathbf{Pr}(\mathrm{lo} \mid \overline{\mathrm{fo}})} \cdot \overset{\checkmark}{\mathbf{Pr}(\overline{\mathrm{fo}})}$

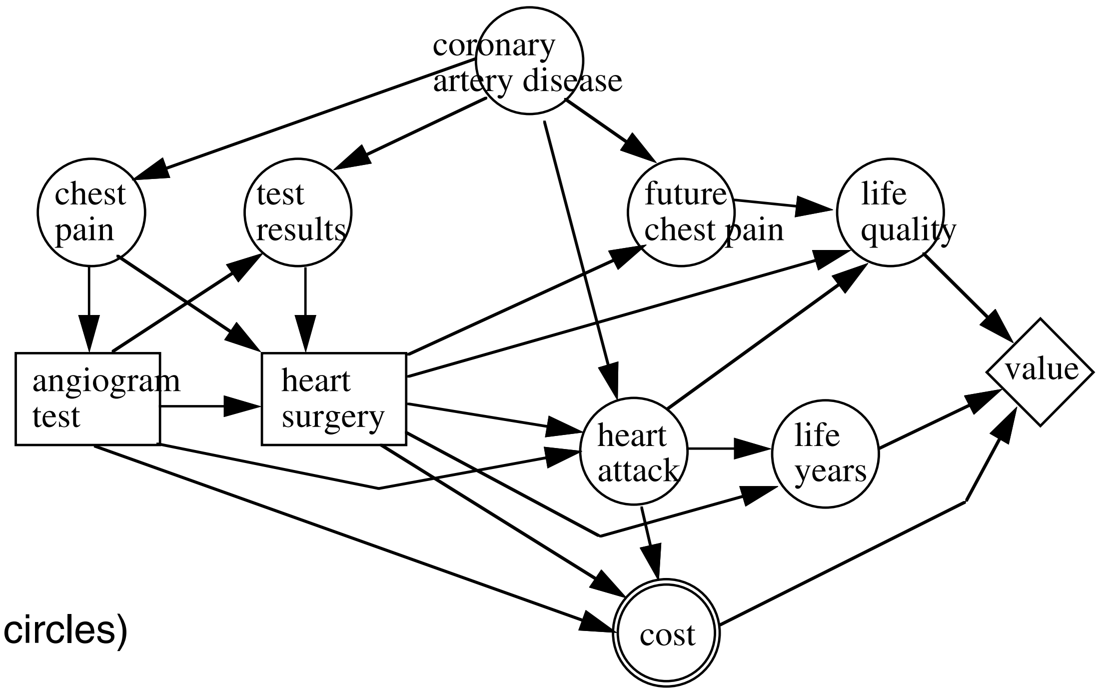
But in general, the problem is NP-hard

- the problem is even hard to approximate in general
- much of the attention on belief networks involves special-purpose procedures that work well for restricted topologies

Influence diagrams

Graphical knowledge representation for decision problems

- nodes represent propositions or quantities of interest, including decision variables, states of the world, and preference values
- arcs represent influence or relevance (probabilistic or deterministic relationships between the variables)



Node types

chance nodes (circles)
value nodes (diamonds)
decision nodes (rectangles)
deterministic nodes (double circles)

Dempster-Shafer theory

Another attempt at evidence-pooling

for cases where there is uncertainty about probability

Uses two-part measure: *belief* and *plausibility*

these are lower and upper bounds on probabilities of a proposition

Relational DB example

Name	Age
a	[22,26]
b	[20,22]
c	[30,35]
d	[20,22]
e	[28,30]

$\{20,21,22\}$ is the set of possibilities of Age(d), or the *possibility distribution* of Age(d)

Set membership questions like $\mathrm{Age}(x) \in Q$ cease to be applicable; more natural to ask about the possibility of Q given the table above of $\mathrm{Age}(x)$

if Q=[20,25], it is *possible* that $Age(a) \in Q$, *not possible* that $Age(c) \in Q$, *certain* that $Age(d) \in Q$

What is the probability that the age of someone is in the range [20,25]?

belief=2/5; plausibility=3/5. So answer is [.4,.6].

DS combination rule $\rightarrow$ multiple sources

Vague predicates

Not every predicate fits every object exactly (nor fails completely)

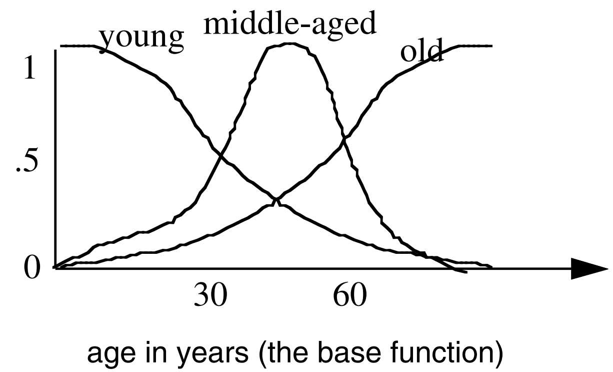
- Categories with degrees of membership
  e.g., fast, old, distant
- Problem: reference sets
  - big fly vs. big elephant

We call predicates that are thought of a holding to a degree vague predicates (or fuzzy predicates).

For each vague predicate, there is a precise base function in terms of which it is understood.

- tall: height
- rich: net worth
- bald: percent hair cover

A degree curve maps the base function to [0,1].



Conjunction and disjunction

As with probabilities, we need boolean combinations of properties

Negation is as with probability:

degree of membership in $\neg P = 1 -$ degree of membership in P

But handle conjunction with MIN and disjunction with MAX!

Example:

Suppose an individual has very high (.95) degree of membership in predicates Tall, Coordinated, Strong, ... for 20 predicates.

Then want to say very high (.95) degree of membership in $(\mathrm{Tall} \wedge \mathrm{Coordinated} \wedge \mathrm{Strong} \wedge \ldots)$

as opposed to

Suppose there is a very high (.95) probability of being Tall, of being Coordinated, of being Strong, ... for 20 predicates.

The probability of being all of them at the same time $(\mathrm{Tall} \wedge \mathrm{Coordinated} \wedge \mathrm{Strong} \wedge \ldots)$ can be low.

Other operators: “very” = square; “somewhat” = square root

Rules with vague predicates

Imagine degrees of fraud = {high, somewhat high, medium, somewhat low, low}, based on a numeric universe of discourse (to some maximum amount)

Construct a set of rules that indicate degrees of fraud based on authorizations and difference in amount of recorded accountability and actual stock:

1) If number of authorizations is often
   then fraud is somewhat high

2) If amount is larger than usual
   then high fraud

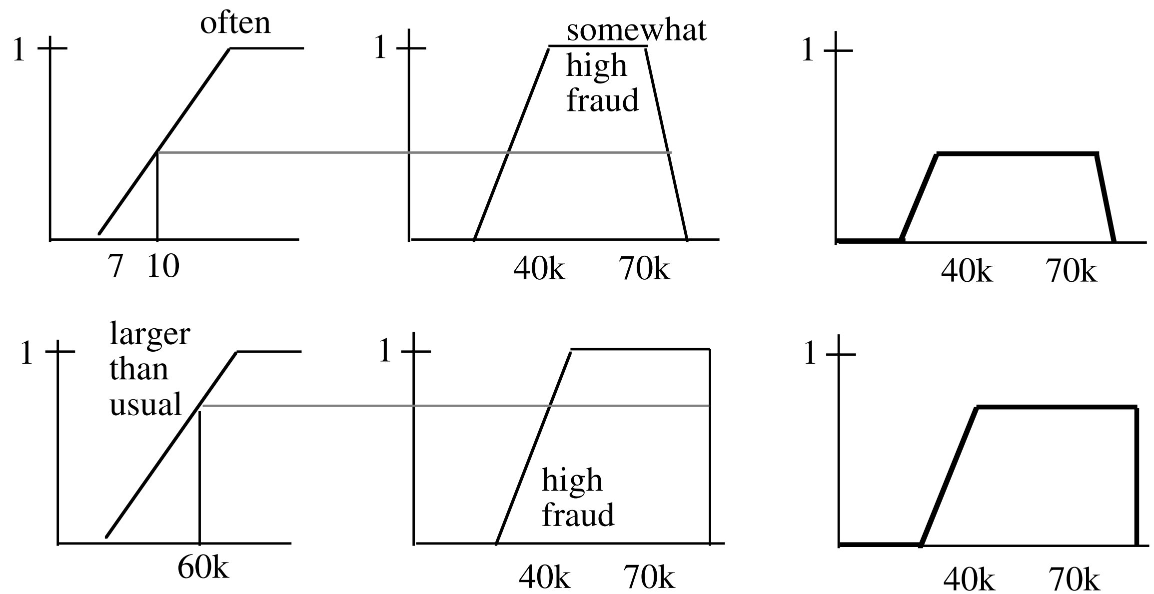
Want to estimate the amount of fraud given inputs

10 authorizations,
amount of $60K

Applying rules

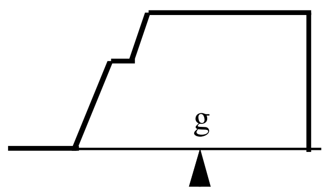
Use degree curves for “somewhat high”, “larger than usual” *etc.*

Can combine with rules in a way that allows conclusion of rule to apply to the degree that the condition of the rule applied.



Given: 10 authorizations
amount of 60k

conclusion:



g =center
of gravity

13.

Explanation and Diagnosis

Abductive reasoning

So far: reasoning has been primarily *deductive*:

- given KB, is α an implicit belief?
- given KB, for what x is $\alpha[x]$ an implicit belief?

Even default / probabilistic reasoning has a similar form

Now consider a new type of question:

Given KB, and an α that I do *not* believe,

what would be sufficient to make me believe that α was true?

or what else would I have to believe for α to become an implicit belief?

or what would *explain* α being true?

Deduction: given $(p \supset q)$, from p, deduce q

Abduction: given $(p \supset q)$, from q, abduce p

p is sufficient for q or one way for q to be true is for p to be true

Also induction: given $p(t_1)$, $q(t_1)$, ..., $p(t_n)$, $q(t_n)$, induce $\forall x\,(p(x) \supset q(x))$

Can be used for causal reasoning: (*cause* $\supset$ *effect*)

Diagnosis

One simple version of diagnosis uses abductive reasoning

KB has facts about symptoms and diseases

including: ($Disease \wedge Hedges \supset Symptoms$)

Goal: find disease(s) that best explain observed symptoms

Observe: we typically do not have knowledge of the form

($Symptom \wedge \ldots \supset Disease$)

so reasoning is not deductive

Example:

(tennis-elbow ⊃ sore-elbow)
(tennis-elbow ⊃ tennis-player)
(arthritis ∧ untreated ⊃ sore-joints)
(sore-joints ⊃ sore-elbow ∧ sore-hip)

Explain: sore-elbow

Want: tennis-elbow,
(arthritis ∧ untreated),
...

Non-uniqueness: multiple equally good explanations

+ logical equivalences: (untreated ∧ ¬¬arthritis)

Adequacy criteria

Given KB, and β to be explained, we want an α such that

1. α is sufficient to account for β

   KB $\cup \{\alpha\} \models \beta$ or KB $\models (\alpha \supset \beta)$

2. α is not ruled out by KB

   KB $\cup \{\alpha\}$ is consistent or KB $\not\models \neg\alpha$

   otherwise $(p \wedge \neg p)$ would count as an explanation

3. α is as simple as possible

   parsimonious : as few *terms* as possible

   explanations should not unnecessarily strong or unnecessarily weak

   e.g. KB = $\{(p \supset q), \neg r\}$ and $\beta = q$

   $\alpha = (p \wedge s \wedge \neg t)$ is too strong

   $\alpha = (p \vee r)$ is too weak

4. α is in the appropriate vocabulary

   atomic sentences of α should be drawn from **H,** possible hypotheses in terms of which explanations are to be phrased

   e.g. diseases, original causes

   e.g. sore-elbow explains sore-elbow

   <u>trivial</u> explanation

   sore-joints explains sore-elbow

   may or may not be suitable

Call such α an <u>explanation</u> of β wrt KB

Some simplifications

From criteria of previous slide, we can simplify explanations in the propositional case, as follows:

- To explain an arbitrary wff β, it is sufficient to choose a new letter p, add $(p \equiv \beta)$ to KB, and then explain p.

  KB $\models (E \supset \beta)$ iff KB $\cup \{(p \equiv \beta)\} \models (E \supset p)$

- Any explanation will be (equivalent to) a conjunction of literals (that is, the negation of a clause)

  Why? If α is a purported explanation, and DNF[α] = $(d_1 \vee d_2 \vee ... \vee d_n)$ then each d_i is also an explanation that is no less simple than α

A simplest explanation is then the negation of a clause with a *minimal* set of literals

So: to explain a literal ρ, it will be sufficient to find the minimal clauses C (in the desired vocabulary) such that

1. KB $\models (\neg C \supset \rho)$ or KB $\models (C \cup \{\rho\})$ sufficient
2. KB $\not\models C$ consistent

Prime implicates

A clause C is a <u>prime implicate</u> of a KB iff

1. KB $\models C$
2. For no $C^* \subset C$, KB $\models C^*$

Note: For any clause C, if KB $\models C$, then some subset of C is a prime implicate

Example: KB = $\{(p \wedge q \wedge r \supset g), (\neg p \wedge q \supset g), (\neg q \wedge r \supset g)\}$

Prime implicates:

$(p \vee \neg q \vee g)$,
$(\neg r \vee g)$, and
$(p \vee \neg p)$, $(g \vee \neg g)$, ...

Note: tautology $(a \vee \neg a)$ is always a prime implicate unless KB $\models a$ or KB $\models \neg a$

For explanations:

- want minimal C such that KB $\models (C \cup \{\rho\})$ and KB $\not\models C$
- so: find prime implicates C such that $\rho \in C$;
  then $\neg(C - \rho)$ must be an explanation for ρ

Example: explanations for g in example above

- 3 prime implicates contain g, so get 3 explanations: $(\neg p \wedge q)$, r, and g

Computing explanations

Given KB, to compute explanations of literal ρ in vocabulary **H**:

calculate the set $\{\neg(C - \rho) \mid C$ is a prime implicate and $\rho \in C\}$

prime implicates containing ρ

But how to compute prime implicates?

Can prove: Resolution is complete for non-tautologous prime implicates

KB |= C iff KB $\rightarrow C$ completeness for [] is a special case!

So: assuming KB is in CNF, generate *all* resolvents in language **H**, and retain those containing ρ that are minimal

Could pre-compute all prime implicates, but there may be *exponentially* many, even for a Horn KB

Example: atoms: $p_i,\ q_i,\ E_i,\ O_i,\ 0 \leq i < n\ +\ E_n, O_n$

wffs: $E_i \wedge p_i \supset O_{i+1},\ E_i \wedge q_i \supset E_{i+1},$
$O_i \wedge p_i \supset E_{i+1},\ O_i \wedge q_i \supset O_{i+1},$
$E_0,\ \neg O_0$

explain: E_n

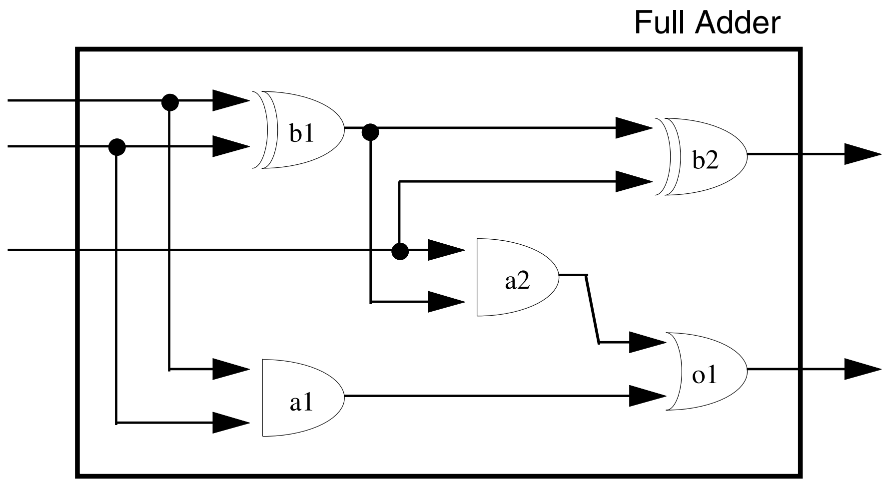
Circuit example

Components

Gate(x) ≡ Andgate(x) ∨ Orgate(x) ∨ Xorgate(x)

Andgate(a1), Andgate(a2),
Orgate(o1),
Xorgate(b1), Xorgate(b2)

Fulladder(f) the whole circuit



Connectivity

in1(b1) = in1(f), in2(b1) = in2(f)

in1(b2) = out(b1), in2(b2) = in3(f)

in1(a1) = in1(f), in2(a1) = in2(f)

in1(a2) = in3(f), in2(a2) = out(b1)

in1(o1) = out(a2), in2(o1) = out(a1)

out1(f) = out(b2), out2(f) = out(o1)

Circuit behaviour

Truth tables for logical gates

and(0,0) = 0,   and(0,1) = 0, ...   or(0,0) = 0,   or(0,1) = 1, ...
xor(0,0) = 0,   xor(0,1) = 1, ...

Normal behaviour

$\mathrm{Andgate}(x) \wedge \neg\mathrm{Ab}(x) \supset \mathrm{out}(x) = \mathrm{and}(\mathrm{in1}(x), \mathrm{in2}(x))$
$\mathrm{Orgate}(x) \wedge \neg\mathrm{Ab}(x) \supset \mathrm{out}(x) = \mathrm{or}(\mathrm{in1}(x), \mathrm{in2}(x))$
$\mathrm{Xorgate}(x) \wedge \neg\mathrm{Ab}(x) \supset \mathrm{out}(x) = \mathrm{xor}(\mathrm{in1}(x), \mathrm{in2}(x))$

Abnormal behaviour: fault models

Examples

$[\mathrm{Orgate}(x) \vee \mathrm{Xorgate}(x)] \wedge \mathrm{Ab}(x) \supset \mathrm{out}(x) = \mathrm{in2}(x)$ (short circuit)

Other possibilities ...

- some abnormal behaviours may be inexplicable
- some may be compatible with normal behaviour on certain inputs

Abductive diagnosis

Given KB as above + input settings

e.g. KB $\cup$ {in1(f) = 1, in2(f) = 0, in3(f) = 1}

we want to explain observations at outputs

e.g. (out1(f) = 1 $\wedge$ out2(f) = 0)

in the language of Ab

We want conjunction of Ab literals α such that KB $\cup$ *Settings* $\cup$ $\{\alpha\}$ |= *Observations*

Compute by "propositionalizing":

For the above, x ranges over 5 components and u, v range over 0 and 1.

Easiest to do by preparing a table ranging over all Ab literals, and seeing which conjunctions entail the observations.

Table for abductive diagnosis

	Ab(b1)	Ab(b2)	Ab(a1)	Ab(a2)	Ab(o1)	Entails observation?
1.	Y	Y	Y	Y	Y	N
2.	Y	Y	Y	Y	N	N
3.	Y	Y	Y	N	Y	N
4.	Y	Y	Y	N	N	N
5.	Y	Y	N	Y	Y	Y
6.	Y	Y	N	Y	N	N
7.	Y	Y	N	N	Y	Y
8.	Y	Y	N	N	N	Y
9.	Y	N	Y	Y	Y	N
10.	Y	N	Y	Y	N	N
11.	Y	N	Y	N	Y	N
12.	Y	N	Y	N	N	N
13.	Y	N	N	Y	Y	Y
14.	Y	N	N	Y	N	N
15.	Y	N	N	N	Y	Y
...						
32.	N	N	N	N	N	N

Example diagnosis

Using the table, we look for minimal sets of literals.

For example, from line (5), we have that

$$\mathrm{Ab(b1)} \wedge \mathrm{Ab(b2)} \wedge \neg\mathrm{Ab(a1)} \wedge \mathrm{Ab(a2)} \wedge \mathrm{Ab(o1)}$$

entails the observations. However, lines (5), (7), (13) and (15) together lead us to a smaller set of literals (the first explanation below).

The explanations are

1. $\mathrm{Ab(b1)} \wedge \neg\mathrm{Ab(a1)} \wedge \mathrm{Ab(o1)}$
2. $\mathrm{Ab(b1)} \wedge \neg\mathrm{Ab(a1)} \wedge \neg\mathrm{Ab(a2)}$
3. $\mathrm{Ab(b2)} \wedge \neg\mathrm{Ab(a1)} \wedge \mathrm{Ab(o1)}$

Note: not all components are mentioned since for these settings, get the same observations whether or not they are working

but for this fault model only

Can narrow down diagnosis by looking at a number of different settings

differential diagnosis

Diagnosis revisited

Abductive definition has limitations

- often only care about what is not working
- may not be able to characterize all possible failure modes
- want to prefer diagnoses that claim as few broken components as possible

Consistency-based diagnosis:

Assume KB uses the predicate Ab as before, but perhaps only characterizes the normal behaviour

e.g. $\mathrm{Andgate}(x) \land \neg\mathrm{Ab}(x) \supset \mathrm{out}(x) = \mathrm{and}(\mathrm{in1}(x), \mathrm{in2}(x))$

Want a minimal set of components D, such that

$$\{\mathrm{Ab(c)} \mid \mathrm{c} \in D\} \cup \{\neg\mathrm{Ab(c)} \mid \mathrm{c} \notin D\}$$

is consistent with KB $\cup$ *Settings* $\cup$ *Observations*

can use table as before with last column changed to “consistency”

In previous example, get 3 diagnoses: {b1}, {b2, a2} and {b2,o1}

Note: more complex to handle non-minimal diagnoses

Some complications

1. negative evidence
   - allow for missing observations

     e.g. ensure that KB $\cup \{\alpha\} \not\models$ fever

2. variables and quantification
   - same definition, modulo "simplicity", (but how to use Resolution?)
   - useful to handle open wffs also

     KB $\cup \{x = 3\} \models P(x)$ handles WH-questions

3. probabilities
   - not all simplest explanations are equally likely
   - also: replace (*Disease* $\wedge$ ... $\supset$ *Symptom*) by a probabilistic version

4. defaults
   - instead of requiring KB $\cup \{\alpha\} \models \beta$, would prefer that given α, it is reasonable to believe β

     e.g. being a bird explains being able to fly

Other applications

1. object recognition

   what scene would account for image elements observed?

   what objects would account for collection of properties discovered?

2. plan recognition

   what high-level goals of an agent would account for the actions observed?

3. hypothetical reasoning

   instead of asking: what would I have to be told to believe β?
   ask instead: what would I learn if I was told that α?

Dual of explanation: want β such that

- $KB \cup \{\alpha\} \models \beta$
- $KB \not\models \beta$
- simplicity, parsimony
- using correct vocabulary

Solution: you learn β on being told α
iff
$\neg\beta$ is an explanation for $\neg\alpha$

can use the abduction procedure

14.

Actions

Situation calculus

The situation calculus is a dialect of FOL for representing dynamically changing worlds in which all changes are the result of named actions.

There are two distinguished sorts of terms:

- actions, such as
  - put(x,y) put object x on top of object y
  - walk(loc) walk to location loc
  - pickup(r,x) robot r picks up object x
- situations, denoting possible world histories. A distinguished constant S_0 and function symbol do are used
  - S_0 the initial situation, before any actions have been performed
  - $do(a,s)$ the situation that results from doing action a in situation s

for example: $do(\text{put}(A,B),do(\text{put}(B,C),S_0))$

the situation that results from putting A on B after putting B on C in the initial situation

Fluents

Predicates or functions whose values may vary from situation to situation are called <u>fluents</u>.

These are written using predicate or function symbols whose last argument is a situation

for example: $\mathrm{Holding}(r, x, s)$: robot r is holding object x in situation s

can have: $\neg\mathrm{Holding}(r, x, s) \wedge \mathrm{Holding}(r, x, do(\mathrm{pickup}(r,x),s))$

the robot is not holding the object x in situation s, but is holding it in the situation that results from picking it up

Note: there is no distinguished “current” situation. A sentence can talk about many different situations, past, present, or future.

A distinguished predicate symbol $Poss(a,s)$ is used to state that a may be performed in situation s

for example: $Poss(\mathrm{pickup}(r,x), S_0)$

it is possible for the robot r to pickup object x in the initial situation

This is the entire language.

Preconditions and effects

It is necessary to include in a KB not only facts about the initial situation, but also about world dynamics: what the actions do.

Actions typically have <u>preconditions</u>: what needs to be true for the action to be performed

- $Poss(\mathrm{pickup}(r,x),\ s)\ \equiv\ \forall z.\ \neg\mathrm{Holding}(r,z,s) \wedge \neg\mathrm{Heavy}(x) \wedge \mathrm{NextTo}(r,x,s)$

  a robot can pickup an object iff it is not holding anything, the object is not too heavy, and the robot is next to the object

  Note: free variables assumed to be universally quantified

- $Poss(\mathrm{repair}(r,x),\ s)\ \equiv\ \mathrm{HasGlue}(r,s) \wedge \mathrm{Broken}(x,s)$

  it is possible to repair an object iff the object is broken and the robot has glue

Actions typically have <u>effects</u>: the fluents that change as the result of performing the action

- $\mathrm{Fragile}(x)\ \supset\ \mathrm{Broken}(x,\ do(\mathrm{drop}(r,x),s))$

  dropping a fragile object causes it to break

- $\neg\mathrm{Broken}(x,\ do(\mathrm{repair}(r,x),s))$

  repairing an object causes it to be unbroken

The frame problem

To really know how the world works, it is also necessary to know what fluents are *unaffected* by performing an action.

- $\text{Colour}(x,c,s) \supset \text{Colour}(x, c, do(\text{drop}(r,x),s))$

  dropping an object does not change its colour
- $\neg\text{Broken}(x,s) \wedge [\, x \neq y \vee \neg\text{Fragile}(x)\,] \supset \neg\text{Broken}(x, do(\text{drop}(r,y),s))$

  not breaking things

These are sometimes called frame axioms.

Problem: need to know a vast number of such axioms. (Few actions affect the value of a given fluent; most leave it invariant.)

an object's colour is unaffected by picking things up, opening a door, using the phone, turning on a light, electing a new Prime Minister of Canada, *etc.*

The frame problem:

- in building KB, need to think of these $\sim 2 \times A \times F$ facts about what does not change
- the system needs to reason efficiently with them

What counts as a solution?

- Suppose the person responsible for building a KB has written down *all* the effect axioms

  for each fluent F and action A that can cause the truth value of F to change, an axiom of the form $[R(s) \supset \pm F(do(A,s))]$, where $R(s)$ is some condition on s

- We want a systematic procedure for generating all the frame axioms from these effect axioms
- If possible, we also want a *parsimonious* representation for them (since in their simplest form, there are too many)

Why do we want such a solution?

- frame axioms are necessary to reason about actions and are not entailed by the other axioms
- convenience for the KB builder
- for theorizing about actions

  – modularity: only add effect axioms

  – accuracy: no inadvertent omissions

The projection task

What can we do with the situation calculus?

We will see later that it can be used for planning.

A simpler job we can handle directly is called the <u>projection task.</u>

Given a sequence of actions, determine what would be true in the situation that results from performing that sequence.

This can be formalized as follows:

Suppose that $R(s)$ is a formula with a free situation variable s.

To find out if $R(s)$ would be true after performing $\langle a_1,...,a_n\rangle$ in the initial situation, we determine whether or not

$$KB \models R(do(a_n,do(a_{n-1},...,do(a_1,S_0)...)))$$

For example, using the effect and frame axioms from before, it follows that $\neg\mathrm{Broken}(\mathrm{B},s)$ would hold after doing the sequence

$$\langle \mathrm{pickup(A)},\ \mathrm{pickup(B)},\ \mathrm{drop(B)},\ \mathrm{repair(B)},\ \mathrm{drop(A)}\rangle$$

The legality task

The projection task above asks if a condition would hold after performing a sequence of actions, but not whether that sequence can in fact be properly executed.

We call a situation legal if it is the initial situation or the result of performing an action whose preconditions are satisfied starting in a legal situation.

The legality task is the task of determining whether a sequence of actions leads to a legal situation.

This can be formalized as follows:

> To find out if the sequence $\langle a_1,...,a_n \rangle$ can be legally performed in the initial situation, we determine whether or not
>
> $$KB \models Poss(a_i, do(a_{i-1},...,do(a_1,S_0)...))$$
>
> for every i such that $1 \leq i \leq n$.

Limitations of the situation calculus

This version of the situation calculus has a number of limitations:

- no time: cannot talk about how long actions take, or when they occur
- only known actions: no hidden exogenous actions, no unnamed events
- no concurrency: cannot talk about doing two actions at once
- only discrete situations: no continuous actions, like pushing an object from A to B.
- only hypotheticals: cannot say that an action has occurred or will occur
- only primitive actions: no actions made up of other parts, like conditionals or iterations

We will deal with the last of these below.

First we consider a simple solution to the frame problem ...

Normal form for effect axioms

Suppose there are two positive effect axioms for the fluent *Broken*:

$$\text{Fragile}(x) \supset \text{Broken}(x, do(\text{drop}(r,x),s))$$

$$\text{NextTo}(b,x,s) \supset \text{Broken}(x, do(\text{explode}(b),s))$$

These can be rewritten as

$$\exists r\, \{a{=}\text{drop}(r,x) \wedge \text{Fragile}(x)\} \;\vee\; \exists b\, \{a{=}\ \text{explode}(b) \wedge \text{NextTo}(b,x,s)\}$$
$$\supset\ \text{Broken}(x, do(a,s))$$

Similarly, consider the negative effect axiom:

$$\neg\text{Broken}(x, do(\text{repair}(r,x),s))$$

which can be rewritten as

$$\exists r\, \{a{=}\text{repair}(r,x)\} \supset \neg\text{Broken}(x, do(a,s))$$

In general, for any fluent F, we can rewrite all the effect axioms as as two formulas of the form

$$\mathrm{P}_F(\boldsymbol{x}, a, s) \supset F(\boldsymbol{x}, do(a,s)) \qquad (1)$$

$$\mathrm{N}_F(\boldsymbol{x}, a, s) \supset \neg F(\boldsymbol{x}, do(a,s)) \qquad (2)$$

where $\mathrm{P}_F(\boldsymbol{x}, a, s)$ and $\mathrm{N}_F(\boldsymbol{x}, a, s)$ are formulas whose free variables are among the x_i, a, and s.

Explanation closure

Now make a completeness assumption regarding these effect axioms:

> assume that (1) and (2) characterize *all* the conditions under which an action a changes the value of fluent F.

This can be formalized by <u>explanation closure axioms</u>:

$$\neg F(\boldsymbol{x}, s) \wedge F(\boldsymbol{x}, do(a,s)) \supset \mathrm{P}_F(\boldsymbol{x}, a, s) \qquad (3)$$

if F was false and was made true by doing action a
then condition P_F must have been true

$$F(\boldsymbol{x}, s) \wedge \neg F(\boldsymbol{x}, do(a,s)) \supset \mathrm{N}_F(\boldsymbol{x}, a, s) \qquad (4)$$

if F was true and was made false by doing action a
then condition N_F must have been true

These explanation closure axioms are in fact disguised versions of frame axioms!

$$\neg F(\boldsymbol{x}, s) \wedge \neg \mathrm{P}_F(\boldsymbol{x}, a, s) \supset \neg F(\boldsymbol{x}, do(a,s))$$

$$F(\boldsymbol{x}, s) \wedge \neg \mathrm{N}_F(\boldsymbol{x}, a, s) \supset F(\boldsymbol{x}, do(a,s))$$

Successor state axioms

Further assume that our KB entails the following

- integrity of the effect axioms: $\neg\exists\, \boldsymbol{x}, a, s.\ \mathrm{P}_\mathrm{F}(\boldsymbol{x}, a, s) \wedge \mathrm{N}_\mathrm{F}(\boldsymbol{x}, a, s)$
- unique names for actions:

$$A(x_1,\ldots,x_n) = A(y_1,\ldots,y_n) \supset (x_1=y_1) \wedge \ldots \wedge (x_n=y_n)$$

$A(x_1,\ldots,x_n) \neq B(y_1,\ldots,y_m)$ where A and B are distinct

Then it can be shown that KB entails that (1), (2), (3), and (4) together are logically equivalent to

$$F(\boldsymbol{x}, do(a,s)) \equiv \mathrm{P}_\mathrm{F}(\boldsymbol{x}, a, s) \vee (F(\boldsymbol{x}, s) \wedge \neg\mathrm{N}_\mathrm{F}(\boldsymbol{x}, a,s))$$

This is called the <u>successor state axiom</u> for F.

For example, the successor state axiom for the *Broken* fluent is:

$$\begin{aligned} &\mathrm{Broken}(x, do(a,s)) \equiv \\ &\quad \exists r\, \{a=\mathrm{drop}(r,x) \wedge \mathrm{Fragile}(x)\} \\ &\vee\ \exists b\, \{a=\mathrm{explode}(b) \wedge \mathrm{NextTo}(b,x,s)\} \\ &\vee\ \mathrm{Broken}(x, s) \wedge \neg\exists r\, \{a=\mathrm{repair}(r,x)\} \end{aligned}$$

An object x is broken after doing action a
iff
a is a dropping action and x is fragile,
or a is a bomb exploding
where x is next to the bomb,
or x was already broken and
a is not the action of repairing it

Note universal quantification: for *any* action a ...

A simple solution to the frame problem

This simple solution to the frame problem (due to Ray Reiter) yields the following axioms:

- one successor state axiom per fluent
- one precondition axiom per action
- unique name axioms for actions

Moreover, we do not get fewer axioms at the expense of prohibitively long ones

the length of a successor state axioms is roughly proportional to the number of actions which affect the truth value of the fluent

The conciseness and perspicuity of the solution relies on

- quantification over actions
- the assumption that relatively few actions affect each fluent
- the completeness assumption (for effects)

Moreover, the solution depends on the fact that actions always have deterministic effects.

Limitation: primitive actions

As yet we have no way of handling in the situation calculus complex actions made up of other actions such as

- conditionals: If the car is in the driveway then drive else walk
- iterations: while there is a block on the table, remove one
- nondeterministic choice: pickup up some block and put it on the floor

and others

Would like to *define* such actions in terms of the primitive actions, and inherit their solution to the frame problem

Need a compositional treatment of the frame problem for complex actions

Results in a novel programming language for discrete event simulation and high-level robot control

The Do formula

For each complex action A, it is possible to define a formula of the situation calculus, $Do(A, s, s')$, that says that action A when started in situation s may legally terminate in situation s'.

Primitive actions: $Do(A, s, s') \;=\; Poss(A,s) \wedge s'=do(A,s)$

Sequence: $Do([A;B], s, s') \;=\; \exists s''.\, Do(A, s, s'') \wedge Do(B, s'', s')$

Conditionals: $Do([\textbf{if}\ \phi\ \textbf{then}\ A\ \textbf{else}\ B], s, s') \;=$

$$\phi(s) \wedge Do(A, s, s') \;\vee\; \neg\phi(s) \wedge Do(B, s, s')$$

Nondeterministic branch: $Do([A \mid B], s, s') \;=\; Do(A, s, s') \vee Do(B, s, s')$

Nondeterministic choice: $Do([\pi x.\, A], s, s') \;=\; \exists x.\, Do(A, s, s')$

etc.

Note: programming language constructs with a purely logical situation calculus interpretation

GOLOG

GOLOG (Algol in logic) is a programming language that generalizes conventional imperative programming languages

- the usual imperative constructs + concurrency, nondeterminism, more...
- bottoms out *not* on operations on internal states (assignment statements, pointer updates) but on *primitive actions* in the world (*e.g.* pickup a block)
- what the primitive actions do is user-specified by precondition and successor state axioms

What does it mean to “execute” a GOLOG program?

- find a sequence of primitive actions such that performing them starting in some initial situation s would lead to a situation s' where the formula $Do(A, s, s')$ holds
- give the sequence of actions to a robot for actual execution in the world

Note: to find such a sequence, it will be necessary to reason about the primitive actions

A ; ***if*** Holding(x) ***then*** B ***else*** C

to decide between B and C we need to determine if the fluent *Holding* would be true after doing A

GOLOG example

Primitive actions: pickup(x), putonfloor(x), putontable(x)

Fluents: Holding(x,s), OnTable(x,s), OnFloor(x,s)

Action preconditions:

$Poss(\text{pickup}(x), s) \equiv \forall z.\neg\text{Holding}(z, s)$

$Poss(\text{putonfloor}(x), s) \equiv \text{Holding}(x, s)$

$Poss(\text{putontable}(x), s) \equiv \text{Holding}(x, s)$

Successor state axioms:

$\text{Holding}(x, do(a,s)) \equiv a=\text{pickup}(x) \vee$
$\quad \text{Holding}(x,s) \wedge a\neq\text{putontable}(x) \wedge a\neq\text{putonfloor}(x)$

$\text{OnTable}(x, do(a,s)) \equiv a=\text{putontable}(x) \vee \text{OnTable}(x,s) \wedge a\neq\text{pickup}(x)$

$\text{OnFloor}(x, do(a,s)) \equiv a=\text{putonfloor}(x) \vee \text{OnFloor}(x,s) \wedge a\neq\text{pickup}(x)$

Initial situation:

$\forall x.\neg\text{Holding}(x, S_0)$

$\text{OnTable}(x, S_0) \equiv x=\text{A} \vee x=\text{B}$

Complex actions:

proc ClearTable : ***while*** $\exists b.\text{OnTable}(b)$ ***do*** $\pi b\ [\text{OnTable}(b)?\ ;\ \text{RemoveBlock}(b)]$

proc RemoveBlock(x) : pickup(x) **;** putonfloor(x)

Running GOLOG

To find a sequence of actions constituting a legal execution of a GOLOG program, we can use Resolution with answer extraction.

For the above example, we have

$KB \models \exists s.\, Do(\mathrm{ClearTable}, S_0, s)$

The result of this evaluation yields

$s = do(\mathrm{putonfloor}(\mathrm{B}), do(\mathrm{pickup}(\mathrm{B}), do(\mathrm{putonfloor}(\mathrm{A}), do(\mathrm{pickup}(\mathrm{A}), S_0))))$

and so a correct sequence is

$\langle$ pickup(A), putonfloor(A), pickup(B), putonfloor(B)$\rangle$

When what is known about the actions and initial state can be expressed as Horn clauses, the evaluation can be done in Prolog.

The GOLOG interpreter in Prolog has clauses like

```
do(A,S1,do(A,S1)) :- prim_action(A), poss(A,S1).

do(seq(A,B),S1,S2) :- do(A,S1,S3), do(B,S3,S2).
```

This provides a convenient way of controlling a robot at a high level.

15.

Planning

Planning

So far, in looking at actions, we have considered how an agent could figure out what to do given a high-level program or complex action to execute.

Now, we consider a related but more general reasoning problem: figure out what to do to make an arbitrary condition true. This is called planning.

- the condition to be achieved is called the goal
- the sequence of actions that will make the goal true is called the plan

Plans can be at differing levels of detail, depending on how we formalize the actions involved

- “do errands” vs. “get in car at 1:32 PM, put key in ignition, turn key clockwise, change gears,…”

In practice, planning involves anticipating what the world will be like, but also observing the world and replanning as necessary...

Using the situation calculus

The situation calculus can be used to represent what is known about the current state of the world and the available actions.

The planning problem can then be formulated as follows:

> Given a formula $Goal(s)$, find a sequence of actions $\boldsymbol{a}$ such that
>
> $$KB \models Goal(do(\boldsymbol{a}, S_0)) \land Legal(do(\boldsymbol{a}, S_0))$$
>
> where $do(\langle a_1,...,a_n \rangle, S_0)$ is an abbreviation for
>
> $$do(a_n, do(a_{n-1}, ..., do(a_2, do(a_1, S_0)) ...))$$
>
> and where $Legal(\langle a_1,...,a_n \rangle, S_0)$ is an abbreviation for
>
> $$Poss(a_1, S_0) \land Poss(a_2, do(a_1, S_0)) \land \; ... \land Poss(a_n, do(\langle a_1,...,a_{n-1} \rangle, S_0))$$

So: given a goal formula, we want a sequence of actions such that

- the goal formula holds in the situation that results from executing the actions,   and
- it is possible to execute each action in the appropriate situation

Planning by answer extraction

Having formulated planning in this way, we can use Resolution with answer extraction to find a sequence of actions:

$$KB \models \exists s.\ Goal(s) \wedge Legal(s)$$

We can see how this will work using a simplified version of a previous example:

An object is on the table that we would like to have on the floor. Dropping it will put it on the floor, and we can drop it, provided we are holding it. To hold it, we need to pick it up, and we can always do so.

$$\left.\begin{array}{ll}
\bullet\ \text{Effects:} & \mathrm{OnFloor}(x, do(\mathrm{drop}(x),s)) \\
 & \mathrm{Holding}(x, do(\mathrm{pickup}(x),s)) \\
 & \quad \text{Note: ignoring frame problem} \\
\bullet\ \text{Preconds:} & \mathrm{Holding}(x, s) \supset Poss(\mathrm{drop}(x), s) \\
 & Poss(\mathrm{pickup}(x), s) \\
\bullet\ \text{Initial state:} & \mathrm{OnTable}(\mathrm{B}, S_0)
\end{array}\right] \quad \text{KB}$$

- The goal: $\mathrm{OnFloor}(\mathrm{B}, s)$

Deriving a plan

Negated query + answer predicate

Axiom 1

[¬OnFloor(B,*s1*), ¬Legal(*s*1), A(*s1*)]

expand *Legal*

[¬Legal(do(drop(B),*s2*)), A(do(drop(B),*s2*))]

Axiom 3

[¬Legal(*s2*), ¬Poss(drop(B),*s2*), A(do(drop(B),*s2*))]

Axiom 2

[¬Legal(*s2*), ¬Holding(B,*s2*), A(do(drop(B),*s2*))]

[A(do(drop(B),do(pickup(B),*s3*))), ¬Legal(do(pickup(B),*s3*))]

expand *Legal*

[¬Legal(*s3*), A(do(drop(B),do(pickup(B),*s3*))), ¬Poss(pickup(B),*s3*),]

Axiom 4

[¬Legal(*s3*), A(do(drop(B),do(pickup(B),*s3*)))]

Legal for S_0

[A(do(drop(B), do(pickup(B), S0)))]

Here is the plan: in the initial situation, pickup block B, and in the resulting situation, drop B.

Using Prolog

Because all the required facts here can be expressed as Horn clauses, we can use Prolog directly to synthesize a plan:

```
onfloor(X,do(drop(X),S)).
holding(X,do(pickup(X),S)).
poss(drop(X),S) :- holding(X,S).
poss(pickup(X),S).
ontable(b,s0).
legal(s0).
legal(do(A,S)) :- poss(A,S), legal(S).
```

With the Prolog goal `?- onfloor(b,S), legal(S).`
we get the solution `S = do(drop(b),do(pickup(b),s0))`

But planning problems are rarely this easy!

Full Resolution theorem-proving can be problematic for a complex set of axioms dealing with actions and situations explicitly...

The STRIPS representation

STRIPS is an alternative representation to the pure situation calculus for planning.

from work on a robot called Shaky at SRI International in the 60's.

In STRIPS, we do not represent histories of the world, as in the situation calculus.

Instead, we deal with a single world state at a time, represented by a database of ground atomic wffs (e.g., $\mathrm{In}(\mathrm{robot},\mathrm{room}_1)$)

This is like the database of facts used in procedural representations and the working memory of production systems

Similarly, we do not represent actions as part of the world model (cannot reason about them directly), as in the situation calculus.

Instead, actions are represented by operators that syntactically transform world models

An operator takes a DB and transforms it to a new DB

STRIPS operators

Operators have pre- and post-conditions

- precondition = formulas that need to be true at start
- "delete list" = formulas to be removed from DB
- "add list" = formulas to be added to DB

Example: PushThru(o,d,r_1,r_2)

"the robot pushes object o through door d from room r_1 to room r_2"

- precondition: InRoom(robot,r_1), InRoom(o,r_1), Connects(d,r_1,r_2)
- delete list: InRoom(robot,r_1), InRoom(o,r_1)
- add list: InRoom(robot,r_2), InRoom(o,r_2)

STRIPS problem space =
- initial world model, DB_0 (list of ground atoms)
- set of operators (with preconds and effects)
- goal statement (list of atoms)

desired plan: sequence of ground operators

STRIPS Example

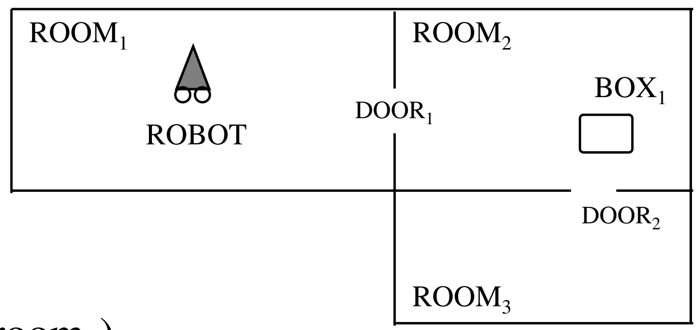
In addition to PushThru, consider

GoThru(d,r_1,r_2):

precondition: InRoom(robot,r_1), Connects(d,r_1,r_2)

delete list: InRoom(robot,r_1)

add list: InRoom(robot,r_2)



DB$_0$:

InRoom(robot,room$_1$) InRoom(box$_1$,room$_2$)

Connects(door$_1$,room$_1$,room$_2$) Box(box$_1$)

Connects(door$_2$,room$_2$,room$_3$) …

Goal: [Box(x) $\wedge$ InRoom(x,room$_1$)]

Progressive planning

Here is one procedure for planning with a STRIPS like representation:

Input : a world model and a goal
Output : a plan or fail.

(ignoring variables)

ProgPlan[DB,Goal] =

If Goal is satisfied in DB, then return empty plan

For each operator o such that precond(o) is satisfied in the current DB:

Let DB´ = DB + addlist(o) – dellist(o)

Let plan = ProgPlan[DB´,Goal]

If plan ≠ fail, then return [act(o) **;** plan]

End for

Return fail

This depth-first planner searches forward from the given DB_0 for a sequence of operators that eventually satisfies the goal

DB´ is the progressed world state

Regressive planning

Here is another procedure for planning with a STRIPS like representation:

Input : a world model and a goal
Output : a plan or fail.

(ignoring variables)

RegrPlan[DB,Goal] =

If Goal is satisfied in DB, then return empty plan

For each operator o such that dellist(o) $\cap$ Goal = {}:

Let Goal´ = Goal + precond(o) – addlist(o)

Let plan = RegrPlan[DB,Goal´]

If plan $\neq$ fail, then return [plan **;** act(o)]

End for

Return fail

This depth-first planner searches backward for a sequence of operators that will reduce the goal to something satisfied in DB_0

Goal´ is the regressed goal

Computational aspects

Even without variables, STRIPS planning is NP-hard.

Many methods have been proposed to avoid redundant search

e.g. partial-order planners, macro operators

One approach: application dependent control

Consider this range of GOLOG programs:

pick an action

< any deterministic program > → ***while*** $\neg Goal$ ***do*** $\pi a \,.\, a$

fully specific about sequence of actions required	any sequence such that $Goal$ holds at end
easy to execute	as hard as planning!

In between, the two extremes we can give domain-dependent guidance to a planner:

while $\neg Goal$ ***do*** $\pi a \,.\, [\mathrm{Acceptable}(a)?\, ;\, a]$

where Acceptable is formalized separately

This is called <u>forward filtering</u> .

Hierarchical planning

The basic mechanisms of planning so far still preserve all detail needed to solve a problem

- attention to too much detail can derail a planner to the point of uselessness
- would be better to first search through an *abstraction space*, where unimportant details were suppressed
- when solution in abstraction space is found, account for remaining details

ABSTRIPS

precondition wffs in abstraction space will have fewer literals than those in ground space

e.g., PushThru operator

- high abstraction: applicable whenever an object is pushable and a door exists
- lower: robot and obj in same room, connected by a door to target room
- lower: door must be open
- original rep: robot next to box, near door

predetermined partial order of predicates with “criticality” level

Reactive systems

Some suggest that explicit, symbolic production of formal plans is something to be avoided (especially considering computational complexity)

even propositional case is intractable; first-order case is undecidable

Just "react": observe conditions in the world and decide (or look up) *what to do next*

can be more robust in face of unexpected changes in the environment

$\Rightarrow$ <u>reactive systems</u>

"Universal plans": large lookup table (or boolean circuit) that tells you exactly what to do based on current conditions in the world

Reactive systems have impressive performance on certain low-level problems (e.g. learning to walk), and can even look "intelligent"

but what are the limitations? ...

16.

The Tradeoff between Expressiveness and Tractability

Limit expressive power?

Defaults, probabilities, *etc.* can all be thought of as extensions to FOL, with obvious applications

Why not strive for the *union* of all such extensions? all of English?

Problem: automated reasoning

Lesson here:

> reasoning procedures required for more expressive languages
> may not work very well in practice

Tradeoff: expressiveness vs. tractability

Overview:

- a Description Logic example
- limited languages
- the problem with cases
- vivid reasoning as an extreme case
- less vivid reasoning
- hybrid reasoning systems

Simple description logic

Consider the language FL defined by:

<concept> ::= atom
| [AND <concept> ... <concept>]
| [ALL <role> <concept>]
| [SOME <role>] (= [EXISTS 1 <role>])

<role> ::= atom
| [RESTR <role> <concept>]

Example: [ALL :Child [AND Female Student]]

an individual whose children are female students

[ALL [RESTR :Child Female] Student]

an individual whose female children are students

there may or may not be male children and they may or may not be students

Interpretation $\mathfrak{I} = \langle D, I\rangle$ as before, but with

$I[[\mathrm{RESTR}\ r\ c]] = \{ (x,y) \mid (x,y) \in I[r] \text{ and } y \in I[c] \}$

So [RESTR :Child Female] is the :Child relation restricted to females = :Daughter

Subsumption defined as usual

Computing subsumption

First for FL$^-$ = FL without the `RESTR` operator

- put the concepts into normalized form
- to see if C subsumes D make sure that
  1. for every $p \in C, \quad p \in D$
  2. for every [`SOME` r] $\in C$, [`SOME` r] $\in D$
  3. for every [`ALL` s c] $\in C$, find an [`ALL` s d] $\in D$ such that c subsumes d.

[`AND` $p_1 \ldots p_k$
[`SOME` r_1] ... [`SOME` r_m]
[`ALL` s_1 c_1] ... [`ALL` s_n c_n]]

Can prove that this method is sound and complete relative to definition based on interpretations

Running time:

- normalization is $O(n^2)$
- structural matching: for each part of C, find a part of D. Again $O(n^2)$

What about all of FL, including `RESTR`?

Subsumption in FL

- cannot settle for part-by-part matching

  [ALL [RESTR :Friend [AND Male Doctor]] [AND Tall Rich]]

  subsumes

  [AND [ALL [RESTR :Friend Male] [AND Tall Bachelor]]
  [ALL [RESTR :Friend Doctor] [AND Rich Surgeon]]]

- complex interactions

  [SOME [RESTR r [AND a b]]]

  subsumes

  [AND [SOME [RESTR r [AND c d]]] [ALL [RESTR r c] [AND a e]]
  [ALL [RESTR r [AND d e]] b]]

In general: FL is powerful enough to encode *all* of propositional logic.

There is a mapping Ω from CNF wffs to FL where

$\models (\alpha \supset \beta)$ iff $\Omega(\alpha)$ is subsumed by $\Omega(\beta)$

But $\models (\alpha \supset (p \wedge \neg p))$ iff α is unsatisfiable

Conclusion: there is no good algorithm for FL   unless P=NP

Moral

Even small doses of expressive power can come at a significant computational price

Questions:

- what properties of a representation language control its difficulty?
- how far can expressiveness be pushed without losing good algorithms
- when is easy reasoning adequate for KR purposes?

These questions remain unanswered, but some progress:

- need for case analyses is a major factor
- tradeoff for DL languages is reasonably well understood
- best addressed (perhaps) by looking at working systems

Useful approach:

- find reasoning tasks that are tractable
- analyze difficulty in extending them

Limited languages

Many reasoning problems that can be formulated in terms of FOL entailment (KB $\models^?$ α) admit very specialized methods because of the restricted form of either KB or α

although problem could be solved using full resolution, there is no need

Example 1: Horn clauses

- SLD resolution provides more focussed search
- in propositional case, a linear procedure is available

Example 2: Description logics

Can do DL subsumption using Resolution

Introduce predicate symbols for concepts, and "meaning postulates" like

$$\forall x[P(x) \equiv \forall y(\mathrm{Friend}(x,y) \supset \mathrm{Rich}(y)) \\ \wedge \forall y(\mathrm{Child}(x,y) \supset \\ \forall z(\mathrm{Friend}(y,z) \supset \mathrm{Happy}(z)))]$$

[AND [ALL :Friend Rich]
[ALL :Child
[ALL :Friend Happy]]]

Then ask if MP $\models \forall x[P(x) \supset Q(x)]$

Equations

Example 3: linear equations

Let E be the usual axioms for arithmetic:

$$\forall x \forall y (x+y = y+x),\ \ \forall x (x+0 = x),\ \ \ldots$$

Peano axioms

Then we get the following:

$$E \models (x+2y=4 \land x-y=1) \supset (x=2 \land y=1)$$

Can “solve” linear equations using Resolution!

But there is a much better way:

Gauss-Jordan method with back substitution

- subtract (2) from (1): $3y = 3$
- divide by 3: $y = 1$
- substitute in (1): $x = 2$

In general, a set of linear equations can be solved in $O(n^3)$ operations

This idea obviously generalizes!

always advantageous to use a specialized procedure when it is available, rather than a general method like Resolution

When is reasoning hard?

Suppose that instead of linear equations, we have something like

$$(x+2y=4 \ \vee \ 3x-y=7) \ \wedge \ x-y=1$$

Can still show using Resolution: $y > 0$

To use GJ method, we need to split cases:

$$x+2y=4 \wedge x-y=1 \quad \Rightarrow \quad y=1$$
$$3x-y=7 \wedge x-y=1 \quad \Rightarrow \quad y=2$$
$$\therefore \ y > 0$$

What if 2 disjunctions? $(eqnA_1 \vee eqnB_1) \ \wedge \ (eqnA_2 \vee eqnB_2)$

there are *four* cases to consider with GJ method

What if n binary disjunctions? $(eqnA_1 \vee eqnB_1) \ \wedge \dots \ \wedge \ (eqnA_n \vee eqnB_n)$

there are 2^n cases to consider with GJ method

with $n=30$, would need to solve 10^9 systems of equations!

Conclusion: case analysis is still a *big* problem.

Question: can we avoid case analyses??

Expressiveness of FOL

Ability to represent <u>incomplete</u> knowledge

$P(a) \vee P(b)$	but which?
$\exists x\, P(x)$	$P(a) \vee P(b) \vee P(c) \vee \ldots$

and even

$c \neq 3$	$c{=}1 \vee c{=}2 \vee c{=}4 \vee \ldots$

Reasoning with facts like these requires somehow "covering" all the implicit cases

languages that admit efficient reasoning do *not* allow this type of knowledge to be represented

- Horn clauses,
- description logics,
- linear equations, ...

only limited forms of disjunction, quantification etc.

Complete knowledge

One way to ensure tractability:

somehow restrict contents of KB so that reasoning by cases is not required

But is <u>complete</u> knowledge enough for tractability?

suppose KB $\models \alpha$ or KB $\models \neg\alpha$, as in the CWA

Get: queries reduce to KB $\models \rho$, literals

But: it can still be hard to answer for literals

Example: KB = $\{(p \vee q), (\neg p \vee q), (\neg p \vee \neg q)\}$

Have: KB $\models \neg p \wedge q$ complete!

But to find literals may require case analysis

So complete knowledge is not enough to avoid case analyses if the knowledge is “hidden” in the KB.

Need a form of complete knowledge that is more explicit...

Vivid knowledge

Note: If KB is complete and consistent, then it is satisfied by a *unique* interpretation I

Why? define I by $I \models p$ iff KB $\models p$ — ignoring quantifiers for now

Then for any I^*, if $I^* \models$ KB then I^* agrees with I on all atoms p

Get: KB $\models \alpha$ iff $I \models \alpha$

entailments of KB are sentences that are true at *I*

explains why queries reduce to atomic case

$(\alpha \vee \beta)$ is true iff α is true or β is true, *etc.*

if we have the I, we can easily determine what is or is not entailed

Problem: KB can be complete and consistent, but unique interpretation may be hard to find

Solution: a KB is <u>vivid</u> if it is a complete and consistent set of literals (for some language)

e.g. KB = $\{\neg p, q\}$ specifies I directly

Quantifiers

As with the CWA, we can generalize the notion of vivid to accommodate queries with quantifiers

A first-order KB is <u>vivid</u> iff for some finite set of positive function-free ground literals KB^+, $KB = KB^+ \cup$ *Negs* $\cup$ *Dc* $\cup$ *Un.*

Get a simple recursive algorithm for KB $\models \alpha$:

KB $\models \exists x.\alpha$ iff KB $\models \alpha[x/c]$, for some $c \in KB^+$

KB $\models (\alpha \vee \beta)$ iff KB $\models \alpha$ or KB $\models \beta$

KB $\models \neg\alpha$ iff KB $\not\models \alpha$

KB $\models (c = d)$ iff c and d are the same constant

KB $\models p$ iff $p \in KB^+$

This is just database retrieval

- useful to store KB^+ as a collection of relations
- only KB^+ is needed to answer queries, but *Negs, Dc,* and *Un* are required to *justify* the correctness of the procedure

Analogues

Can think of a vivid KB as an analogue of the world

there is a 1-1 correspondence between

- objects in the world and constants in the KB^+
- relationships in the world and syntactic relationships in the KB^+

for example, if constants c_1 and c_2 stand for objects in the world o_1 and o_2

there is a relationship R holding between objects o_1 and o_2 in the world

iff

constants c_1 and c_2 appear as a tuple in the relation represented by R

Not true in general

for example, if KB = $\{P(a)\}$ then it only uses 1 constant, but could be talking about a world where there are 5 individuals of which 4 satisfy P

Result: certain reasoning operations are easy

- how many objects satisfy P (by counting)
- changes to the world (by changes to KB^+)

Beyond vivid

Requirement of vividness is very strict.

Want weaker alternatives with good reasoning properties

Extension 1

ignoring quantifiers again

Suppose KB is a finite set of literals

- not necessarily a complete set (no CWA)
- assume consistent, else trivial

Cannot reduce KB $\models \alpha$ to literal queries

if KB = $\{p\}$ then KB $\models (p \wedge q \;\vee\; p \wedge \neg q)$ but KB $\not\models p \wedge q$ and KB $\not\models p \wedge \neg q$

But: assume α is small. Can put into CNF

$\alpha \Rightarrow (c_1 \wedge \ldots \wedge c_n)$

- KB $\models \alpha$ iff KB $\models c_i$, for every clause in CNF of α
- KB $\models c$ iff c has complimentary literals – tautology
  or KB $\cap\, c$ is not empty

Extension 2

Imagine KB vivid as before + new definitions:

$\forall xyz[R(x,y,z) \equiv$ ... wff in vivid language ...]

Example: have vivid KB using predicate ParentOf

add: $\forall xy[\mathrm{MotherOf}(x,y) \equiv \mathrm{ParentOf}(x,y) \wedge \mathrm{Female}(x)]$

To answer query containing $R(t_1,t_2,t_3)$, simply macro expand it with definition and continue

- can handle arbitrary logical operators in definition since they become part of *query*, not KB
- can generalize to handle predicates not only in vivid KB, provided that they bottom out to KB^+

$$\forall xy[\mathrm{AncestorOf}(x,y) \equiv \mathrm{ParentOf}(x,y) \vee \exists z\, \mathrm{ParentOf}(x,z) \wedge \mathrm{AncestorOf}(z,y)]$$

- clear relation to Prolog

  a version of logic programming based on inductive definitions,
  not Horn clauses

Other extensions

Vivification: given non-vivid KB, attempt to make vivid e.g. by eliminating disjunctions *etc.*

for example,

- use taxonomies to choose between disjuncts

  Flipper is a whale or a dolphin.

- use intervals to encompass disjuncts

  The picnic will be on June 2, 3,or 4th.

- use defaults to choose between disjuncts

  Serge works in Toronto or Montreal.

Problem: what to do with function symbols, when Herbrand universe is not finite?

partial Herbrand base?

Hybrid reasoning

Want to be able to incorporate a number of special-purpose efficient reasoners into a single scheme such as Resolution

Resolution will be the glue that holds the reasoners together

Simple form: <u>semantic attachment</u>

- attach procedures to functions and predicates

  e.g. numbers: procedures on plus, LessThan, ...

- ground terms and atomic sentences can be *evaluated* prior to Resolution
  - $P(\text{factorial}(4), \text{times}(2,3)) \;\Rightarrow\; P(24, 6)$
  - $\text{LessThan}(\text{quotient}(36,6), 5) \vee \alpha \;\Rightarrow\; \alpha$

- much better than reasoning directly with axioms

More complex form: <u>theory resolution</u>

- build theory into unification process (the way paramodulation builds in =)
- extended notion of complimentary literals

  $\{\alpha, \text{LessThan}(2,x)\}$ and $\{\text{LessThan}(x,1), \beta\}$ resolve to $\{\alpha,\beta\}$

Using descriptions

Imagine that predicates are defined elsewhere as concepts in a description logic

$$\text{Married} \doteq [\texttt{AND} \ \ldots] \qquad\qquad \text{Bachelor} \doteq [\texttt{AT-MOST} \ \ldots]$$

then $\{P(x), \text{Married}(x)\}$ and $\{\text{Bachelor}(\text{john}), Q(y)\}$ resolve to $\{P(\text{john}), Q(y)\}$

Can use description logic procedure to decide if two predicates are complimentary

instead of explicit meaning postulates

Residues: for “almost” complimentary literals

$\{P(x), \text{Male}(x)\}$ and $\{\neg\text{Bachelor}(\text{john}), Q(y)\}$

resolve to

$\{P(\text{john}), Q(y), \text{Married}(\text{john})\}$

since the two literals are contradictory *unless* John is married

Main issue: what resolvents are necessary to get the same conclusions as from meaning postulates?

residues are necessary for completeness

THE END