The language of the first-order logic (FOL)

Three things define a declarative language:

- syntax - what groups of symbols are valid and in what order
  "the car that ⊃ drive"
  "drive the car ⊃ that"
- semantics - what the well-formed sentences mean - some expressions may not mean anything
  "blue holiday runs"
- pragmatics - how meaningful expressions are used
  "There is someone behind you"

The syntax in FOL

There are two types of symbols: logical and nonlogical.

1. The logical symbols (like the reserved words in a programming language)
   a) punctuation: "(", ")", "."
   b) connectives: ¬, ∧, ∨ (in descending order of priority)
      quantifiers ∃, ∀
      = logical equality (special symbol, not a predicate)
   c) variables: an infinit set of symbols (denoted by x, y, z with/without subscripts and superscripts)
2. The nonlogical symbols (like the identifiers in a programming language)
   - they have an application-dependent meaning or use
   - they have arity
   a) function symbols - start with a lower-case letter; written in mixed case

   examples: bestFriend
   a, b, c, f, g, h (with/without sub/superscripts)

   by convention - if arity is zero, we use symbols a, b, c - they are called constants.

b) predicate symbols - start with an upper-case letter; written in mixed case

examples: OlderThan

P, Q, R (with/without sub/superscripts)

- predicate symbols of arity 0 are called propositional symbols.

In FOL there are two types of valid syntactic expressions

1) Terms

a) every variable is a term;

b) if $t_1, \ldots, t_n$ are terms and f is a function symbol with n arguments, then $f(t_1, \ldots, t_n)$ is a term

2) Formulas

a) if $t_1, \ldots, t_n$ are terms and P is a predicate symbol with n arguments, then $P(t_1, \ldots, t_n)$ is a formula (called atomic formulas (or atoms))

b) if t_1 and t_2 are terms, then $t_1 = t_2$ is a formula (called atomic formulas (or atoms))

c) if α and β are formulas and x is a variable, then $\neg\alpha$, $\alpha \wedge \beta$, $\alpha \vee \beta$, $\forall x.\alpha$ and $\exists x.\alpha$ are formulas

The propositional subset of FOL is the language with no terms or quantifiers, but only propositional symbols are used:

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

Abbreviations:

$\alpha \supset \beta$ for $(\neg\alpha \vee \beta)$ (also denoted by $\rightarrow$ or $\Rightarrow$)

$\alpha \equiv \beta$ for $((\alpha \supset \beta) \wedge (\beta \supset \alpha))$ (also denoted by $\Leftrightarrow$)

A variable occurrence is bound in a formula if it lies within the scope of a quantifier. Otherwise it is free.

$$\forall y.\ P(x) \wedge \exists x\,[P(y) \vee Q(x)]$$

free

bound

bound

Obs. In some books, the occurrence of the variable just after the quantifiers is neither free nor bound.

Definition. A sentence in FOL is any formula without free variables

$$\forall y . \exists x [P(y) \vee Q(x)]$$

The semantics in FOL

The meaning of a sentence derives from the interpretation of the nonlogical symbols involved.
Nonlogical symbols are application-dependent, so no definitive answers can be offered.
We are looking for a clear specification of the meaning of a sentence as a function of the interpretation of the predicate and function symbols.
For example, the meaning of "DemocraticCountry" can be specified by "objects" that represent those countries that we consider to be democratic. We may agree or not on which those countries are, but in this case we just talk about different interpretations. In terms of FOL, we are not interested to say what "DemocraticCountry" means according to a dictionary (free elections, representative government etc.)

Interpretations in FOL

An interpretation I is a pair $\langle D, I \rangle$, where
- D is a non-empty set of objects, called the domain of the interpretation (it can be anything)
- I is the interpretation mapping, that assigns a meaning to the predicate and function symbols.

If P is a predicate symbol of arity n, then

$I[P] \subseteq \underbrace{D \times \dots \times D}_{n \text{ times}}$ $\quad I[P]$ is a n-ary relation over D.

if f is a function symbol of arity n, then

$I[f] \in [\underbrace{D \times \dots \times D}_{n \text{ times}} \to D]$ $I[f]$ is a n-ary function over D

Examples

$D = \{d_1, d_2, d_3, ana, maria, costel, george, ion, \dots\}$

Dog - unary predicate symbol

$I[Dog] = \{d_1, d_2, d_3\}$ - the set of dogs in this interpretation

OlderThan - binary predicate symbol

$I[OlderThan] = \{(ana, maria), (costel, maria)\}$

bestFriend - unary function symbol

$I[bestFriend] : D \to D$

$I[bestFriend](ana) = maria$

firstChildOf - binary function symbol

$I[firstChildOf] : D \times D \to D$

$I[firstChildOf](maria, vasile) = george$

$I[motherOfThreeChildren] : D \times D \times D \to D$

$I[motherOfThreeChildren](george, ion, ana) = maria.$

Obs. johnSmith is a constant $I[johnSmith] = ion \in D$

A useful alternative to interpret predicates symbols is in terms of their characteristic function. Thus, for P a predicate of arity n, we view $I[P]$ as an n-ary function to $\{0,1\}$

$$I[P] \in [\underbrace{D \times \dots \times D}_{n \text{ times}} \to \{0,1\}]$$

The two specifications are related as following: a tuple of objects is considered to be in the relation over D iff the characteristic function over those objects has value 1.

If P is a predicate of arity 0, $I[P]$ is either 0 or 1 (false/true)

For the propositional subset of FOL, the domain D can be ignored and see the interpretation as a mapping I from the propositional symbols to either 0 or 1.

Denotation

It means to indicate which element of Δ is denoted by a term in an interpretation $\mathcal{I} = \langle \Delta, I \rangle$

1. if a term does not contain any variable

$$\| \text{bestFriend}(\text{johnSmith}) \|_{\mathcal{I}} = I[\text{bestFriend}](I(\text{johnSmith}))$$

2. if a term contains variables, we first define μ – a variable assignment over Δ.

for a variable x, $\mu[x] \in \Delta$.

The denotation of a term t, given $\mathcal{I}$ and μ, is written $\|t\|_{\mathcal{I},\mu}$ and it is defined by the rules:

a) if x is a variable, then $\|x\|_{\mathcal{I},\mu} = \mu[x]$;

b) if $t_1, \ldots, t_n$ are terms and f is a function symbol of arity n, then

$$\| f(t_1, \ldots, t_n) \|_{\mathcal{I},\mu} = I[f](\|t_1\|_{\mathcal{I},\mu}, \ldots, \|t_n\|_{\mathcal{I},\mu})$$

$$\| \text{bestFriend}(x) \|_{\mathcal{I},\mu} = I[\text{bestFriend}](\mu[x])$$

if $\mu[x] = \text{ion}$ then $I[\text{bestFriend}](\text{ion})$

The denotation of a term is an element of Δ.

Satisfaction

We can say which sentences in FOL are true and which are false, according to an interpretation $\mathcal{I}$ and a variable assignment μ

For example, $\text{Dog}(\text{bestFriend}(\text{johnSmith}))$ is true in $\mathcal{I}$ iff

1. we use I to obtain the subset of Δ denoted by "Dog"
2. find the object in Δ denoted by "bestFriend(johnSmith)"
3. the object found at step 2 belongs to the subset found at 1.

Given $\mathcal{I}$ and μ, we say that the formula α is satisfied in $\mathcal{I}, \mu$ and we write $\mathcal{I}, \mu \models \alpha$ according to the following rules:

1. $\mathcal{J}, \mu \models P(t_1, \ldots t_n)$ iff $\langle \|t_1\|_{\mathcal{J},\mu}, \ldots, \|t_n\|_{\mathcal{J},\mu} \rangle \in I[P]$;
2. $\mathcal{J}, \mu \models t_1 = t_2$ iff $\|t_1\|_{\mathcal{J},\mu}$ & $\|t_2\|_{\mathcal{J},\mu}$ are the same element of Δ;
3. $\mathcal{J}, \mu \models \neg\alpha$ iff it is not the case that $\mathcal{J}, \mu \models \alpha$;
4. $\mathcal{J}, \mu \models (\alpha \wedge \beta)$ iff $\mathcal{J}, \mu \models \alpha$ and $\mathcal{J}, \mu \models \beta$;
5. $\mathcal{J}, \mu \models (\alpha \vee \beta)$ iff $\mathcal{J}, \mu \models \alpha$ or $\mathcal{J}, \mu \models \beta$;
6. $\mathcal{J}, \mu \models \exists x.\alpha$ iff $\mathcal{J}, \mu' \models \alpha$ for some variable assignment μ' that differs from μ on at most x
7. $\mathcal{J}, \mu \models \forall x.\alpha$ iff $\mathcal{J}, \mu' \models \alpha$ for every variable assignment μ' that differs from μ on at most x

- if α is a sentence, satisfaction does not depend on any μ We write $\mathcal{J} \models \alpha$ and read "α is true in the interpretation $\mathcal{J}$"
- For the propositional subset of FOL, we write $I[\alpha] = 1$ or 0 according to whether $\mathcal{J} \models \alpha$ or not
- if S is a set of sentences, we write $\mathcal{J} \models S$ if all the sentences in S are true in $\mathcal{J}$. We say that $\mathcal{J}$ is a logical model of S.

The pragmatics in FOL

It specifies how well-formed expressions are to be used.

To be able to reason, concepts like "Dog", "Democratic Country" should be given an intended interpretation.

Logical consequence

Although the interpretation of the nonlogical symbols defines the semantic interpretation in FOL, still there are connections between sentences that do not depend on the meaning of those symbols.

Example: α, β sentences in FOL

let $\gamma = \neg(\beta \wedge \neg\alpha)$

if $\mathcal{I}$ is an interpretation where α is true then γ is true under $\mathcal{I}$ and its truth value does not depend on how we understand the nonlogical symbols in α and β.

We say that α logically entails γ, or γ is a logical consequence of α.

For a set of sentences S and a sentence α, we say that α is a logical consequence of S (or S logically entails α) iff for every interpretation $\mathcal{I}$ with $\mathcal{I} \models S$ then $\mathcal{I} \models \alpha$.

Or equivalently, there is no interpretation $\mathcal{I}$ where $\mathcal{I} \models S \cup \{\neg\alpha\}$.

We write $S \models \alpha$

A sentence α is logically valid, which we write $\models \alpha$, if it is a logical consequence of the empty set (i.e. α is valid iff $\forall\, \mathcal{I}, \mathcal{I} \models \alpha$).

if $S = \{\alpha_1, \ldots, \alpha_n\}$ finite and α is a sentence, then

$S \models \alpha$ iff $[(\alpha_1 \wedge \ldots \wedge \alpha_n) \supset \alpha]$ is valid.

The logical entailment is the key of a knowledge-based system.

For example, if Fido is a dog then a reasoning system should be able to conclude that Fido is a mammal.

if a set of sentences S entails a sentence α, then α is true in every interpretation where S is true. Other sentences that are not entailed by S may or may not be true, but a knowledge-based system must conclude that the entailed sentences are true.

If we have an interpretation J where $Dog(fido)$ is true, then the system can conclude that $\neg\neg Dog(fido)$ and $(Dog(fido) \vee Happy(john))$ are true. These conclusions are logically safe but this is not the kind of reasoning we would be interested in.
Something more useful would be if a system concludes from $Dog(fido)$ that $Mammal(fido)$.
We can find an interpretation where $Dog(fido)$ is true and $Mammal(fido)$ is false.

For example, $J = \langle D, I \rangle$

$D = \{d\}$

$I[Dog] = \{d\}$

$I[P] = \{\}$ for every other predicate $P \neq Dog$

$I[f](d, \ldots, d) = d$

We have $J \models Dog(fido)$ but $J \not\models Mammal(fido)$.
So, there is no logical connection between the two sentences. To create it, we need to include in S a statement that connects the nonlogical symbols involved:

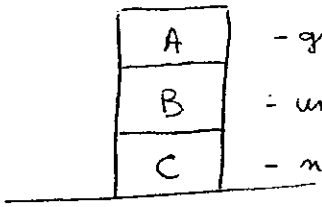
$$\forall x.\ Dog(x) \Rightarrow Mammal(x)$$

Thus, $Mammal(x)$ becomes the logical consequence of $Dog(x)$ and we rule out all the interpretations where the set of dogs is not included in the set of mammals.

"Truth in the intended interpretation"

Reasoning based on logical consequence allows only safe, logically guaranteed conclusions in a knowledge-based system.

Exercise



Is there a green block directly on top of a nongreen one?

Formalization in FOL

a, b, c the names of the blocks
G unary predicate symbol for "green"
O binary predicate symbol for "on"

$S = \{ O(a,b),\ O(b,c),\ G(a),\ \neg G(c) \}$

the sentence α is $\exists x \exists y .\ G(x) \wedge \neg G(y) \wedge O(x,y)$
We want to prove that $S \models \alpha$

Let $\mathcal{I}$ a logical model for S $\quad \mathcal{I} \models S$

1. Suppose that $\left.\begin{array}{l} \mathcal{I} \models G(b) \\ \neg G(c),\ O(b,c) \in S \end{array}\right\} \Rightarrow \mathcal{I} \models G(b) \wedge \neg G(c) \wedge O(b,c$

$\Rightarrow \mathcal{I} \models \exists x \exists y .\ G(x) \wedge \neg G(y) \wedge O(x,y)$

2. Suppose thad $\left.\begin{array}{l} \mathcal{I} \models \neg G(b) \\ G(a),\ O(a,b) \in S \end{array}\right\} \Rightarrow \mathcal{I} \models G(a) \wedge \neg G(b) \wedge O(a,$

$\Rightarrow \mathcal{I} \models \exists x \exists y .\ G(x) \wedge \neg G(y) \wedge O(x,y)$

Thus, α is a logical consequence of S.

There is no automated procedure in FOL to decide in all cases whether a sentence is entailed or not from others.

A reasoning process is
- <u>logically sound</u> if whenever it produces α, then α is guaranteed to be a logical consequence
- <u>logically complete</u> if it is guaranteed to produce α whenever α is entailed

The barber's paradox (formulated by Bertrand Russell)

- Anyone who does not shave himself must be shaved by the barber
- Whomever the barber shaves, must not shave himself

Show that no barber can fulfill these requirements.

$$\forall x.\ \text{Person}(x) \wedge (\neg \text{Shave}(x,x) \supset \text{Shave}(\text{barber}, x))$$

$$\forall x.\ \text{Person}(x) \wedge (\text{Shave}(\text{barber}, x) \supset \neg \text{Shave}(x,x))$$

$$\mathcal{I} \models \forall x.\ \text{Person}(x) \wedge (\neg \text{Shave}(x,x) \supset \text{Shave}(\text{barber}, x))$$

$$\mathcal{I} \models \text{Person}(\text{barber}) \wedge (\neg \text{Shave}(\text{barber}, \text{barber}) \supset \text{Shave}(\text{barber}, \text{barber}))$$

$$\mathcal{I} \models \neg \text{Shave}(\text{barber}, \text{barber}) \supset \text{Shave}(\text{barber}, \text{barber})$$

$$\mathcal{I} \models \text{Shave}(\text{barber}, \text{barber}) \supset \neg \text{Shave}(\text{barber}, \text{barber})$$

$\alpha \supset \beta$ is $\neg\alpha \vee \beta$

$$\mathcal{I} \models \text{Shave}(\text{barber}, \text{barber})$$

$$\mathcal{I} \models \neg \text{Shave}(\text{barber}, \text{barber})$$

Expressing Knowledge – creating a Knowledge-base

Knowledge engineering – is the first step when creating a knowledge base – and it means deciding on the representatio language followed by determining the kinds of objects important to the agent, the properties those objects have and the relationships among them.

Vocabulary

We start by identifying the essential entities in the agent's world:

- constant symbols
  - persons: johnSmith
  - institutions: government
  - places: centralStation
- description of the basic types of objects:
  - Person(x), County(x), Restaurant(x)
- the set of attributes of objects:
  - Rich, Nice, Smart
- express relationships:
  - DaughterOf (ana, mary)
  - MarriedTo (ana, ion)
- functions:
  - bestFriendOf (john)
  - firstChildOf (ana, john)

Basic Facts

They are represented by atomic sentences and negations of atomic sentences ($P(t_1, \ldots, t_n)$ and $t_1 = t_2$)

Man(john), Rich(mary), WorksFor (john, george)

$\neg$ HappilyMarried(john)

bestFriendOf(john) = george

$y \neq$ ana

Complex Facts

Connectors are used to express various beliefs

$$\forall y \; [\, Rich(y) \wedge Man(y) \Rightarrow Loves(y, mary)\,]$$

$$\forall y \; [\, Woman(y) \wedge y \neq jane \Rightarrow Loves(y, john)\,]$$

We can express general facts

$$\forall x \; \forall y \; [\, Loves(x, y) \Rightarrow \neg Blackmails(x, y)\,]$$

or incomplete knowledge

$$Loves(jane, john) \vee Loves(jane, jim)$$

$$\exists x \; [\, Adult(x) \wedge Blackmails(x, john)\,]$$

Relationships among predicates

If john is Man then Women(john) should be false.

If MariedTo(ana, john) is true then MariedTo(john, ana) should be true.

But a KB does not generate by itself such inferences. We need to provide a set of facts about the terminology we are using.

- Disjointness - the assertion of one implies the negation of the other

$$\forall x \; [\, Man(x) \Rightarrow \neg Woman(x)\,]$$

- Subtypes - $\forall x \; [\, Surgeon(x) \Rightarrow Doctor(x)\,]$

- Exhaustiveness - two or more subtypes completely account for a supertype

$$\forall x \; [\, Adult(x) \Rightarrow (Man(x) \vee Women(x))\,]$$

- Symmetry - $\forall x, y \; [\, MariedTo(x, y) \Rightarrow MariedTo(y, x)\,]$

- Type restrictions - defining the meaning of a predicate requires arguments of certain types

$$\forall x, y \; [\, MariedTo(x, y) \Rightarrow Person(x) \wedge Person(y)\,]$$

- Full definitions: predicates that are completely defined by a logical combination of other predicates

$$\forall x \, [RichMan(x) \equiv Rich(x) \wedge Man(x)]$$

Entailments

It means to derive implicit conclusions from explicit knowledge in KB.

Example 1

KB:
1. $Rich(john)$
2. $Man(john)$
3. $\forall y \, [Rich(y) \wedge Man(y) \supset Loves(y, jane)]$
4. $john = ceoOf(insuranceCompany)$
5. $Company(insuranceCompany)$

Question: is there a company whose CEO loves Jane?

in FOL $\exists x \, [Company(x) \wedge Loves(ceoOf(x), jane)]$

Let J an interpretation that is a logical model for KB.

J satisfies 1, 2, 3 from KB $\Rightarrow$ $J \models Loves(john, jane)$
$john = ceoOf(insuranceCompany)$ $\Rightarrow$

$J \models Loves(ceoOf(insuranceCompany), jane)$
$Company(insuranceCompany)$ $\Rightarrow$

$J \models Company(insuranceCompany) \wedge Loves(ceoOf(insuranceCompany), jane)$

$\Rightarrow J \models \exists x \, [Company(x) \wedge Loves(ceoOf(x), jane)]$ □

Obs. $KB \models (\alpha \supset \beta)$ iff $\forall J$, $J \models KB$ then $J \models \neg\alpha \vee \beta$

a) if $J \models \neg\alpha$ then $J \models \neg\alpha \vee \beta$

b) if $J \models \alpha$ then $J \models (\alpha \supset \beta)$ iff $J \models \beta$

so if $J \models \alpha$ then $KB \models (\alpha \supset \beta)$ iff $KB \cup \{\alpha\} \models \beta$

Example 2

KB

1. $\exists x [Adult(x) \wedge Blackmails(x, john)$
2. $\forall x [Adult(x) \supset (Man(x) \vee Woman(x))]$
3. $Loves(john, jane)$
4. $\forall y [(Woman(y) \wedge y \neq jane) \supset Loves(y, john)]$
5. $\forall x \forall y [Loves(x, y) \supset \neg Blackmails(x, y)]$

Question: if no man blackmails John, then is he blackmailed by someone he loves?

in FOL $\forall x [Man(x) \supset \neg Blackmails(x, john)] \supset$

$\exists y [Loves(john, y) \wedge Blackmails(y, john)]$

Let $J \models KB$ and $J \models \forall x [Man(x) \supset \neg Blackmails(x, john)]$

We want to show that $J \models \exists y [Loves(john, y) \wedge Blackmails(y, john)]$

1. $\exists x [Adult(x) \wedge Blackmails(x, john)]$
2. $\forall x [Adult(x) \supset (Man(x) \vee Woman(x))]$
   $\forall x [Man(x) \supset \neg Blackmails(x, john)]$

$\Rightarrow$

6. $J \models \exists x [Women(x) \wedge Blackmails(x, john)]$

4. $\forall y [(Woman(y) \wedge y \neq jane) \supset Loves(y, john)]$
5. $\forall x \forall y [Loves(x, y) \supset \neg Blackmails(x, y)]$

$\Rightarrow$

7. $J \models \forall y [(Women(y) \wedge y \neq jane) \supset \neg Blackmails(y, john)$

6, 7 $\Rightarrow$ $J \models Blackmails(jane, john)$

3. $J \models Loves(john, jane)$

$\Rightarrow$

$J \models Loves(john, jane) \wedge Blackmails(jane, john)$ $\Rightarrow$

$J \models \exists y [Loves(john, y) \wedge Blackmails(y, john)]$

□

Abstract individuals

In FOL we represent facts in a domain, but we can get a greater flexibility in representation if we map objects onto predicates and functions.

Reification means to transform a sentence into an object, by creating new abstract individuals.

For instance, we can say that John purchases a bike:

Purchases (john, bike)

Purchases (john, bike, oct11)

Purchases (john, bike, oct11, 1000RON) ...

The arity of "Purchases" depends on the level of the details that we want to express.

We will consider "purchase" to be an abstract individual called, for instance, p17. We can now describe this purchase using predicates and functions:

$$Purchase(p17) \wedge agent(p17) = john \wedge object(p17) = bike \wedge$$
$$amount(p17) = 1000RON \wedge time(p17) = 16$$

Instead of $time(p17) = 16$ we can say $time(p17) = t19 \wedge$
$hour(t19) = 16 \wedge minute(t19) = 23$

The advantage now is that the arities of the predicate and function symbols are determined in advance.

Other types of facts

- Statistical and probabilistic facts

Half of the companies are profitable.

Most of the students work.

There are 10% chances that tomorrow will be sunny.

- Default and prototypical facts - usually true, unless stated otherwise

  Cars have four wheels.

  Companies do not allow employees that work together to be married.

- intentional facts

  John believes that Henry blackmails him.

  Jane does not want Jim to know that she loves him.

Exercise

KB
- Tony, Mike and John belong to the Alpine Club.
- Every member of the Alpine Club who is not a skier is a mountain climber.
- Mountain climbers do not like rain and anyone who does not like snow is not a skier.
- Mike dislikes whatever Tony likes and likes whatever Tony dislikes.
- Tony likes rain and snow

a) Represent the above sentences in FOL, using a consistent vocabulary (which you must define)

b) Prove that the sentences logically entail that there is a member of the Alpine Club who is a mountain climber but not a skier.

Resolution

Until now, we have seen how logical reasoning could be used to discover new facts in a knowledge base (through logical entailment). The reasoning was done by hand, in a kind of informal manner.

We are looking for a procedure that can determine whether or not $KB \models \alpha$, where KB is a given knowledge base and α is a sentence.

Also, if $B[x_1, \ldots, x_n]$ is a formula with free variables among the x_i, we want a procedure that determines terms t_i, if they exist, such that $KB \models B[t_1, \ldots, t_n]$.

But there is no automated procedure to fully satisfy this requirement (in all cases).

We are looking for a procedure that does deductive reasoning in a manner as sound and complete as possible and in a language as close as possible to full FOL.

A reasoning process is logically sound if whenever it produces α, then α is guaranteed to be a logical consequence (this would exclude the possibility of producing facts that may be true in the intended interpretation but are not strictly entailed).

A reasoning process is logically complete if it guarantees to produce α, whenever α is entailed (this would exclude the possibility of missing some entailments, for instance when their status is too difficult to determine).

If KB is a finite set of sentences $\{\alpha_1, \ldots, \alpha_n\}$, the deductive reasoning can be formulated in several equivalent ways:

1. $KB \models \alpha$
2. $\models [(\alpha_1 \wedge \ldots \wedge \alpha_n) \supset \alpha]$
3. $KB \cup \{\neg\alpha\}$ is not satisfiable
4. $KB \cup \{\neg\alpha\} \models \neg TRUE$

where TRUE is any valid sentence (for example $\forall x . x = x$)

4 ⇒ 3
if there is J so that J ⊨ KB ∪ {¬α} then
KB ∪ {¬α} ⊨ TRUE in J — contradiction with
KB ∪ {¬α} ⊨ ¬TRUE

if we have a procedure for testing the validity of sentences, or for testing the satisfiability of sentences, or for determining whether or not ¬TRUE is entailed, then that procedure can also be used to find the entailments of a finite KB.

The propositional case of Resolution

The propositional logic is a restricted form of formulas.
Every formula α of propositional logic can be transformed into α', a conjunction of disjunctions of literals (i.e. atoms or its negation), such that ⊨ (α ≡ α')

α' is in conjunctive normal form CNF

Example: $(p \vee q \vee \neg r) \wedge (p \vee \neg s \vee \neg q) \wedge (\neg q \vee r)$

Att: lowercase letters are used for propositional symbols to be consistent with common practice

The procedure for convertion of any propositional formula to CNF:

1. replace ⊃, ≡ with the formulas they represent
2. move ¬ inward so that it appears in front of an atom
   ⊨ $(\neg\neg\alpha \equiv \alpha)$
   ⊨ $\neg(\alpha \wedge \beta) \equiv \neg\alpha \vee \neg\beta$
   ⊨ $\neg(\alpha \vee \beta) \equiv \neg\alpha \wedge \neg\beta$
3. distribute ∧ over ∨
   ⊨ $(\alpha \vee (\beta \wedge \gamma)) \equiv ((\beta \wedge \gamma) \vee \alpha) \equiv (\alpha \vee \beta) \wedge (\alpha \vee \gamma)$
4. collect terms
   ⊨ $(\alpha \vee \alpha) \equiv \alpha$
   ⊨ $(\alpha \wedge \alpha) \equiv \alpha$

Obs. The result is a logically equivalent CNF formula which can be exponentially larger than the initial formula

$$((p \supset q) \equiv r) \longrightarrow (\neg(\neg p \vee q) \vee r) \wedge (\neg r \vee (\neg p \vee q))$$

$$((p \wedge \neg q) \vee r) \wedge (\neg r \vee \neg p \vee q)$$

$$(p \vee r) \wedge (\neg q \vee r) \wedge (\neg r \vee \neg p \vee q) \quad \text{CNF}$$

We will write CNF using a shorthand representation.

A clause is a finite set of literals (understood as a disjunction of its literals).

A clausal formula is a finite set of clauses (understood as a conjunction of its clauses).

Notations: $\overline{\ell}$ is the complement of the literal ℓ

$\overline{\ell} \stackrel{def}{=} \neg\ell$ and $\neg\overline{\ell} = \ell$

{set of clausal formulas}

[set of literals]

For example, $[p, \neg q, r]$ represents $(p \vee \neg q \vee r)$

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

A clause with a single literal is called a unit clause

$[p]$, $[\neg q]$

Obs. $\{\} \neq \{[]\}$

$\{\}$ - the empty clausal formula = conjunction of no constraints is a representation of TRUE

$[]$ - disjunction of no possibilities - is a representation of $\neg$TRUE

$\{[]\}$ stands for $\neg$TRUE

In order to determine whether or not $KB \models \alpha$, it is sufficient to do the following:

1. convert the sentences in KB and $\neg\alpha$ into CNF;
2. determine whether or not the resulting set of clauses is satisfiable.

Resolution derivations

The rule of inference called Resolution is the following.

Given a clause $C_1 \cup \{P\}$ where P is a literal, and a clause $C_2 \cup \{\bar{P}\}$, then $C_1 \cup C_2$ is inferred (C_1 and C_2 may be empty).

We say that $C_1 \cup C_2$ is a resolvent of the two input clauses with respect to P.

For example $[\underline{p}, q, r]$ and $[q, \neg\underline{p}, s]$

$[q, r, s]$ is a resolvent with respect to p.

$[p, q]$ and $[\neg p, \neg q]$ have two resolvents:

$[q, \neg q]$ with respect to p

$[p, \neg p]$ with respect to q

Obs. The only way to get $[\,]$ is by resolving two complementary unit clauses like $[p]$ and $[\neg p]$.

Def. A Resolution derivation of a clause c from a set of clauses S is a sequence of clauses $c_1, \ldots, c_n$, where $c_n = c$ and each c_i is either an element of S or a resolvent of two prior clauses in the derivation.
We write $S \vdash c$ if there is a derivation of c from S.

The Resolution derivations are important because these symbol-level operation on finite sets of literals is directly connected to knowledge-level logical interpretations.

Obs. The resolvent is always the logical consequence of the two input clauses.

$$\{C_1 \cup \{p\}, C_2 \cup \{\neg p\}\} \models C_1 \cup C_2$$

Let J be an interpretation so that $J \models C_1 \cup \{p\}$ and $J \models C_2 \cup \{\neg p\}$

1. if $J \models p$ then $J \not\models \neg p$
   but $J \models c_2 \cup \{\neg p\}$
   $\Rightarrow J \models c_2 \Rightarrow J \models c_1 \cup c_2$

2. if $J \not\models p$ but $J \models c_1 \cup \{p\} \Rightarrow J \models c_1 \Rightarrow J \models c_1 \cup c_2$

Obs. Any clause derivable by Resolution from S is logically entailed by S, that is if $S \vdash c$ then $S \models c$.

Proof – by induction on the length of the derivation, we show that for every c_i it follows that $S \models c_i$

[$S \vdash c$ if $\exists c_1, \dots, c_n = c$ so that either $c_i \in S$ or c_i is the resolvent of two earlier clauses in the derivation.

if $c_i \in S$ then $S \models c_i$

if c_i is a resolvent of c_j and $c_k \Rightarrow \{c_j, c_k\} \models c_i$
from the induction $S \models c_j$
hypothesis $S \models c_k$
$\Rightarrow$

$S \models c_i$.

The converse does not hold – we can have $S \models c$ without $S \vdash c$.

For example, $S = \{[\neg p]\}$ and $c = [\neg q, q]$

$S \models c$ but $c \notin S$ and there are no resolvents, so $S \not\vdash c$.

Obs. The resolution derivations are not logically complete (do not guarantee to produce α whenever $S \models \alpha$).

Obs. But Resolution is both sound and complete when $c = []$.

$S \vdash []$ iff $S \models []$ (S is unsatisfiable)

Thus, the problem of determining the satisfiability of any set of clauses is reduced to the search for a derivation of the empty clause.

The entailment procedure

We want to determine whether or not $KB \models \alpha$ (equivalent to $KB \cup \{\neg\alpha\}$ is unsatisfiable).
Let S be the set of clauses obtained by converting $KB \cup \{\neg\alpha\}$ in CNF.
We check if S is unsatisfiable by searching for a derivation of the empty clause.

The nondeterministic procedure:

input: a finite set S of propositional clauses

1. if $([\,] \in S)$ then return unsatisfiable
   else if (there are two clauses in S
   that can resolve to produce
   another clause not already in S)
   then (add the new resolvent clause
   to S and go to step 1)
   else return satisfiable

output: satisfiable or unsatisfiable

Obs. The procedure terminates because each added clause is a resolvent of previous clauses and contains only literals from the initial set of clauses S (a finite number) — eventually nothing new can be added.

Obs. The procedure can be made deterministic — we set a strategy for choosing the pair of clauses to produce a new resolvent — e.g. the first pair encountered; the pair that produces the shortest resolvent.

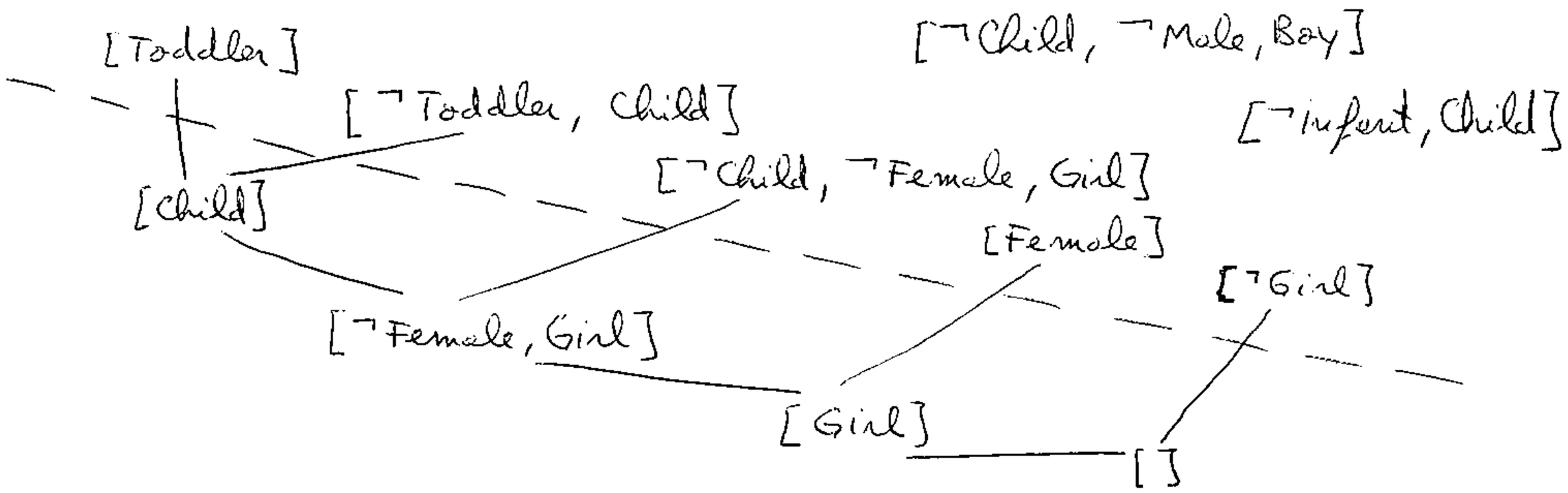
if we are interested in returning the derivation, for each resolvent we should store pointers to its input clauses.

Example 1 We have the following knowledge base

$$KB \begin{cases} Toddler \\ Toddler \supset Child \\ Child \wedge Male \supset Boy \\ infant \supset Child \\ Child \wedge Female \supset Girl \\ Female \end{cases}$$

Question: Girl

$KB \models Girl$ iff $KB \cup \{\neg Girl\}$ is unsatisfiable

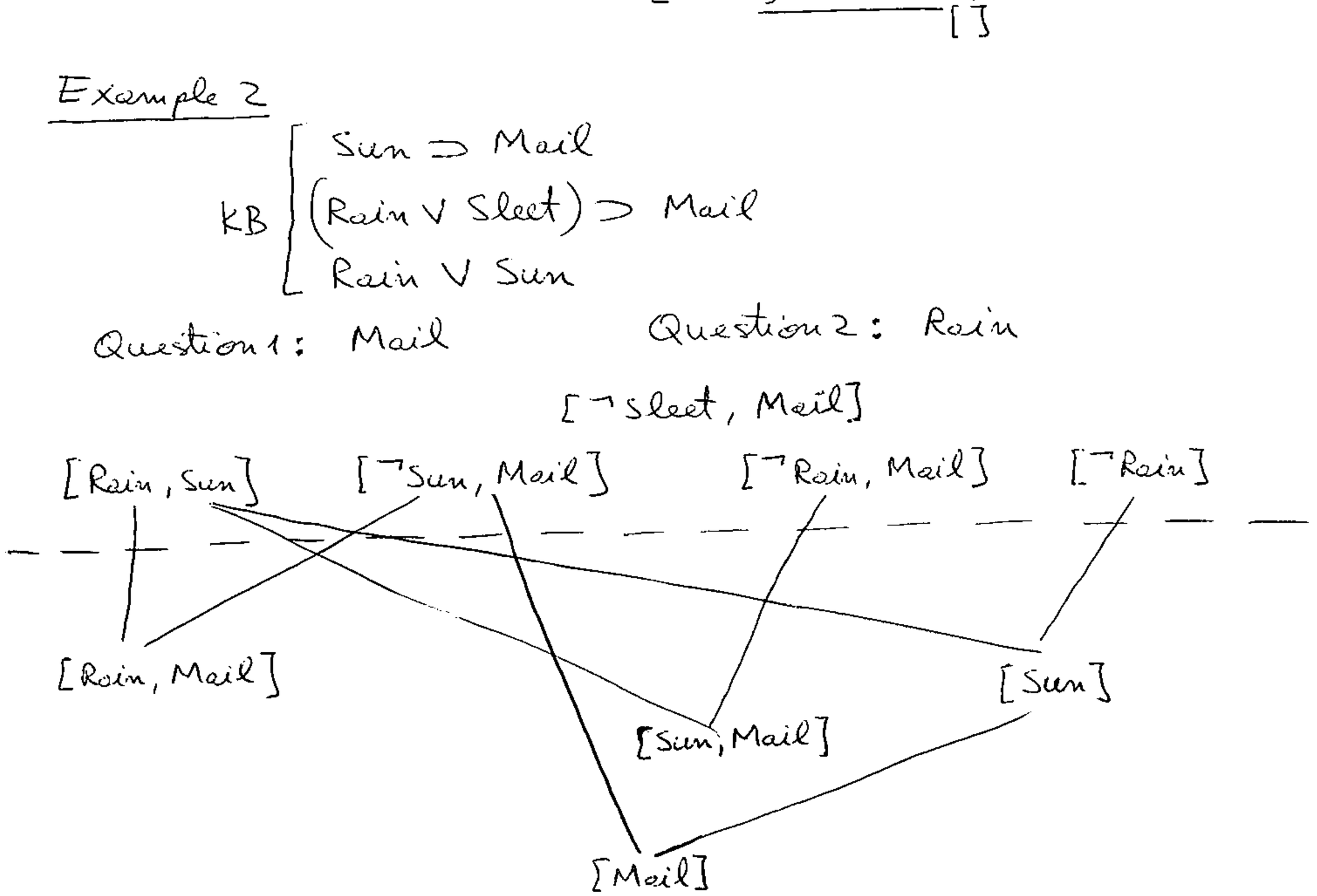


Example 2

$$KB \begin{cases} Sun \supset Mail \\ (Rain \vee Sleet) \supset Mail \\ Rain \vee Sun \end{cases}$$

Question 1: Mail

Question 2: Rain



$\Rightarrow KB \not\models Rain$

Handling variables and quantifiers

We transform formulas into an equivalent clausal form:

1. replace $\supset$ and $\equiv$ as indicated before (page 2)
2. move $\neg$ inward, adding the following two.

$$\models \neg \forall x . \alpha \equiv \exists x . \neg \alpha$$

$$\models \neg \exists x . \alpha \equiv \forall x . \neg \alpha$$

3. rename variables (if necessary) so that the variables in the two input clauses of Resolution are distinct.
4. eliminate all remaining existentials.
5. move universals outside the scope of $\wedge$ and $\vee$ using the following equivalences (provided that x does not occur free in α)

$$\models (\alpha \wedge \forall x . \beta) \equiv \forall x (\alpha \wedge \beta)$$

$$\models (\alpha \vee \forall x . \beta) \equiv \forall x (\alpha \vee \beta)$$

6. distribute $\wedge$ over $\vee$ as before
7. collect terms as before

For the beginning, we consider the case where no existentials appear. We drop the quantifiers (they are all universals).
Atoms have the form $P(t_1, \ldots, t_n)$ (we ignore now $t_1 = t_2$).
For example, the clausal formula

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

represents the CNF formula

$$\forall x \forall y \left([P(x,y) \vee \neg R(a, f(b,x))] \wedge [Q(y) \vee T(x, g(a))] \right)$$

Def. A substitution Θ is a finite set of pairs $\{x_1/t_1, \ldots, x_n/t_n\}$ where x_i are distinct variables and t_i are terms.
if Θ substitution, ζ literal, we write $\zeta\Theta$ = the literal that results from simultaneously replacing each x_i in ζ by t_i.

For example, $\theta = \{ x/f(a,y),\ y/g(x,z) \}$

$$\rho = P(h(x, b, y), z)$$

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

If c is a clause, $c\theta$ is the clause resulting from making the substitution on each literal.

We say that a term, literal or clause is ground if it contains no variables.

We say that ρ is an instance of ρ' if there is θ so that $\rho = \rho'\theta$.

First-Order Resolution

Since the clauses with variables are universally quantified, we want to allow Resolution to be applied to any of their instances.

For example $[P(x, f(a))]$ and $[\neg P(g(b,z), y), \neg Q(z, f(b))]$

$x/g(b,z)$ $\quad$ $y/f(a)$

$[P(g(b,z), f(a))]$ and $[\neg P(g(b,z), f(a)), \neg Q(z, f(b))]$

the resolvent is $[\neg Q(z, f(b))]$

We define the general rule of Resolution as follows:

We are given the clauses $C_1 \cup \{\rho_1\}$ and $C_2 \cup \{\bar{\rho}_2\}$, where ρ_1 and ρ_2 are literals.

We rename the variables in the two clauses (if necessary) so that each clause has distinct variables.

Suppose there is a substitution θ such that $\rho_1\theta = \rho_2\theta$.

Then we can infer the clause $(C_1 \cup C_2)\theta$.

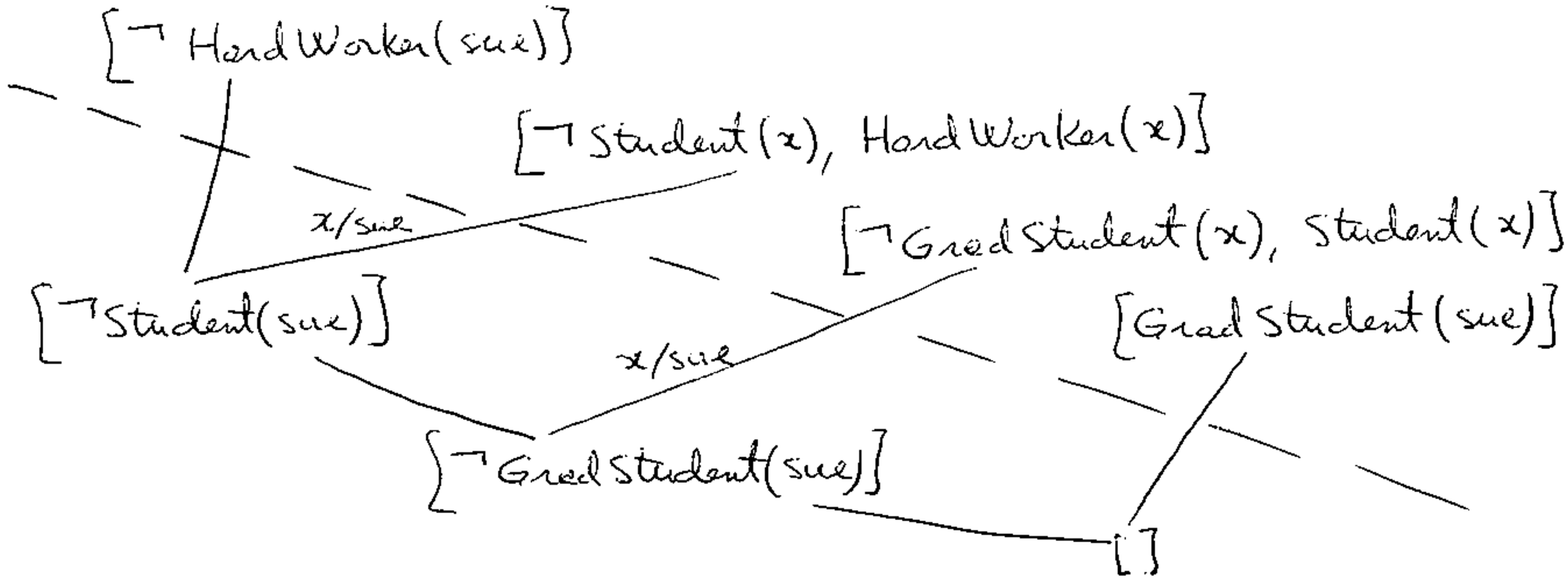
We say that θ is a unifier of ρ_1 and ρ_2.

With this general rule of Resolution, it is the case that $S \vdash [\,]$ iff $S \models [\,]$.

Example 3

$$KB \begin{cases} \forall x.\ GradStudent(x) \supset Student(x) \\ \forall x.\ Student(x) \supset HardWorker(x) \\ GradStudent(sue) \end{cases}$$

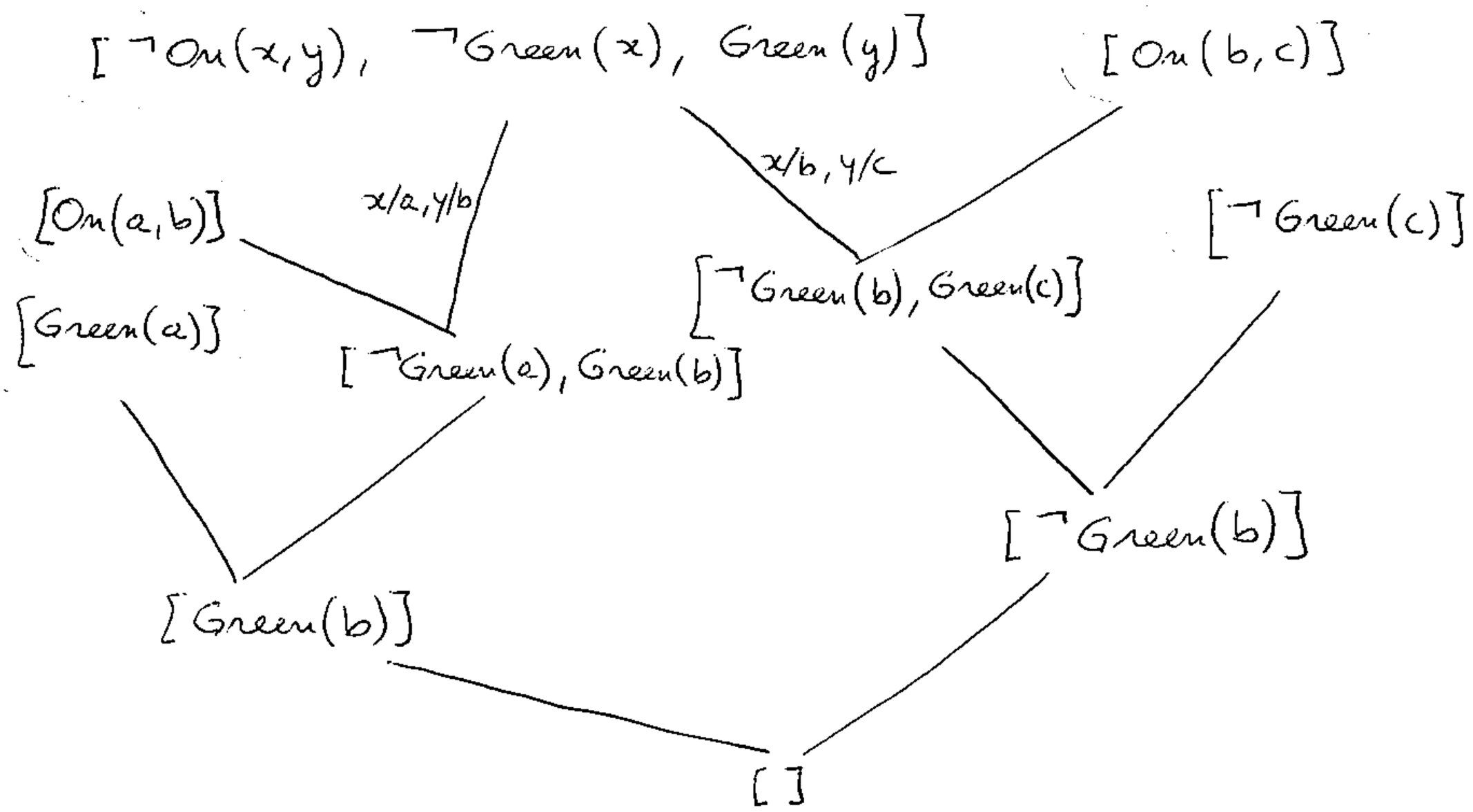
$KB \models HardWorker(sue)$



Example 4 The three-block problem

KB: $On(a,b)$, $On(b,c)$, $Green(a)$, $\neg Green(c)$

Question: $\exists x \exists y.\ On(x,y) \wedge Green(x) \wedge \neg Green(y)$



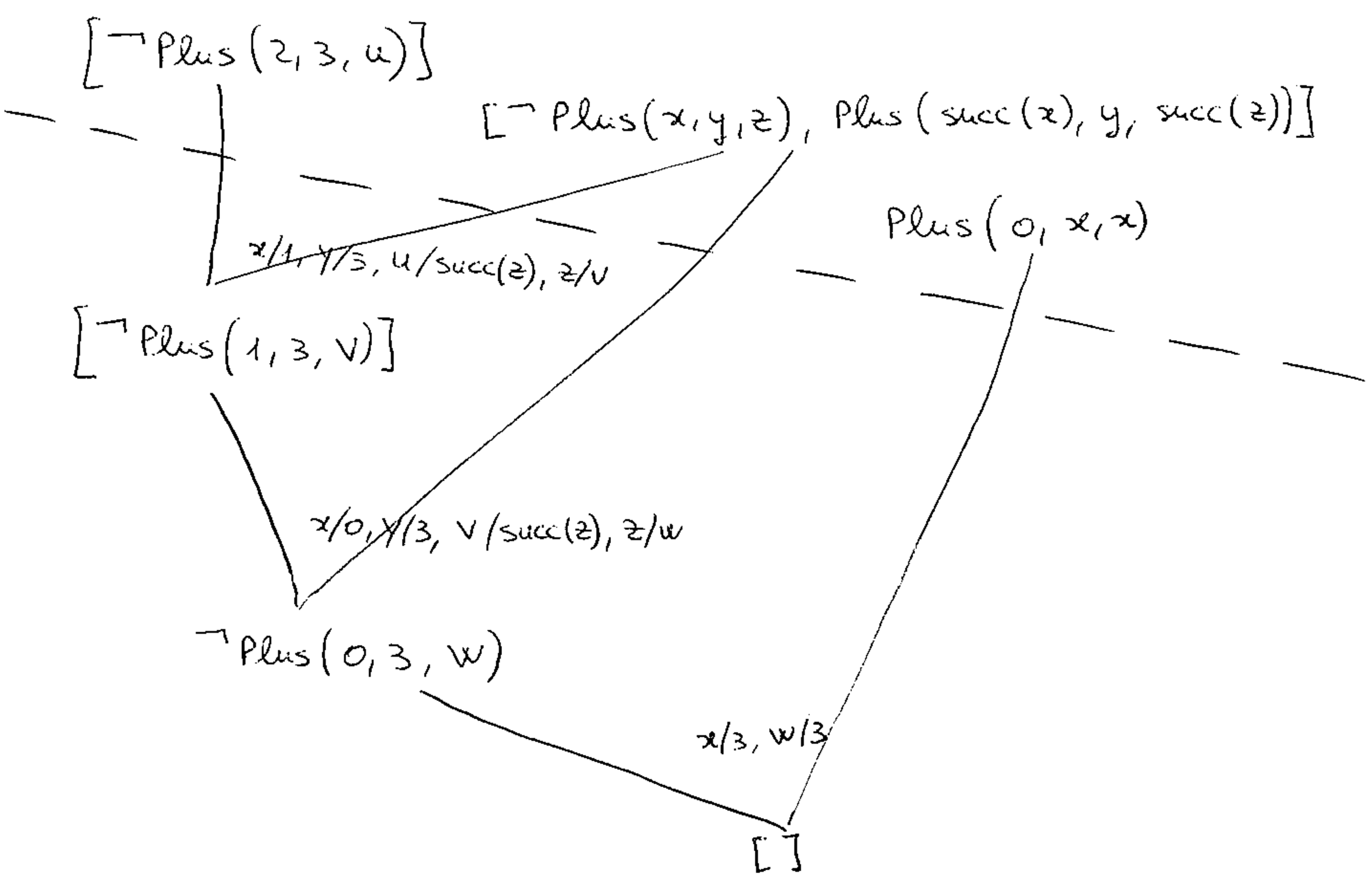
Example 5 - The necessity of renaming variables

$$KB \begin{cases} \forall x.\ Plus(zero, x, x) \\ \forall x\, \forall y\, \forall z.\ Plus(x, y, z) \supset Plus(succ(x), y, succ(z)) \end{cases}$$

Question: $\exists u.\ Plus(2, 3, u)$

$Plus(x, y, z)$ represents $x + y = z$

$succ(succ(succ(zero)))$ represents 3



We can identify the value of u:

u is bound to succ(v); v is bound to succ(w);

w is bound to 3 $\Rightarrow$ $u = 5$

Resolution (continuation)

Answer extraction

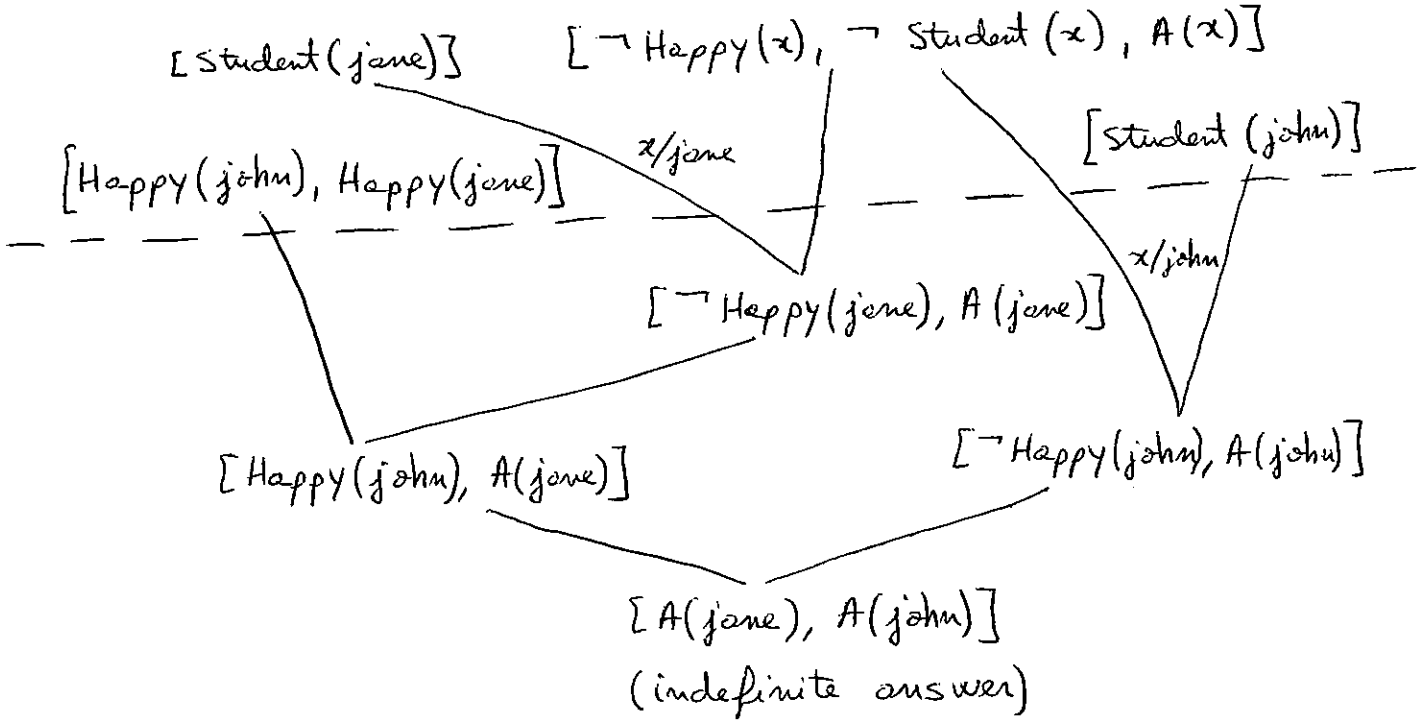
It is often possible to get answers to questions by looking at the bindings of variables in a derivation of an existential (see Example 5 in the previous course). But more complicated situations may appear in FOL. In Example 4 (prev. course), we know that there is a block that satisfies a condition, but we don't know which block. That is to say that $KB \models \exists x.P(x)$ without entailing $P(t)$ for a specific t.

Idea: replace a question such as $\exists x.P(x)$ by $\exists x.P(x) \wedge \neg A(x)$, where A is a new predicate symbol that occurs nowhere else. A is called the answer predicate. Since A does not appear anywhere else, it will not be possible to derive the empty clause. Therefore, the derivation ends when we produce a clause containing only the answer predicate.

Example 1

$$KB \begin{cases} \text{Student}(john) \\ \text{Student}(jane) \\ \text{Happy}(john) \vee \text{Happy}(jane) \end{cases}$$

Question: $\exists x.\ \text{Student}(x) \wedge \text{Happy}(x)$



Obs. We can get answers containing variables

For example,

$$KB \begin{cases} \forall z.\ Child(f(g(a)), z) \\ \forall x\, \forall y.\ Play(f(x), g(y)) \end{cases}$$

Question: $\exists x\, \exists y.\ Child(x,y) \wedge Play(x,y)$

The negation of the question is $[\neg Child(x,y), \neg Play(x,y), A(x,y)]$ and we get a derivation with the final clause $[A(f(g(a)), g(y))]$ interpreted as "answer is any instance of the terms $f(g(a)), g(y)$".

Skolemization

It is the process of removing existential quantifiers by elimination.

Idea: we introduce unique names for each variable quantified with an existential.

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

we name x a

z $f(y)$

} we will use instead $\forall y.\ P(a, y, f(y))$

a, f are called Skolem symbols (they do not appear anywhere else).

In general, Skolemization replace each existential variable by a new function symbol with as many arguments as there are universal variables dominating the existential.

$$\alpha \quad \forall x_1 (\ldots \forall x_2 (\ldots \forall x_3 (\ldots \exists y [\ldots y \ldots] \ldots) \ldots) \ldots)$$

$$\Big\downarrow \text{ Skolemization}$$

$$\alpha' \quad \forall x_1 (\ldots \forall x_2 (\ldots \forall x_3 (\ldots [\ldots f(x_1, x_2, x_3) \ldots] \ldots) \ldots) \ldots)$$

where f appears nowhere else.

Obs. $\not\models (\alpha \equiv \alpha')$

For example, $\exists x.\, kill(x, victim)$ strictly speaking is not logically equivalent to $kill(murderer, victim)$.

Att: Do not confuse these substitutions (i.e. Skolem constants) with the interpretations used to define the semantics of quantifiers. The substitution replaces a variable with a term (it's syntax) to produce new sentences, whereas an interpretation maps a variable to an object in the domain.

Obs: it can be proven that α and α' are inferentially equivalent:

$KB \cup \{\alpha\}$ is satisfiable iff $KB \cup \{\alpha'\}$ is satisfiable

Att: For logical correctedness, it is important to have the dependence of variables right.

For example, $\exists x \forall y\, R(x,y) \models \forall y \exists x\, R(x,y)$ but the converse does not hold.

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

$$\downarrow \text{CNF}$$

$$\{[R(a,y)],\ [\neg R(x,b)]\} \qquad a, b \text{ Skolem constants}$$

$$\downarrow x/a,\ y/b$$

$$[\,]$$

The converse $\{\forall y \exists x\, R(x,y),\ \neg \exists x \forall y\, R(x,y)\}$

$$\downarrow \text{CNF}$$

$$\{[R(f(y),y)],\ [\neg R(x,g(x))]\} \qquad f, g \text{ Skolem functions}$$

do not unify

Example 2 (taken from https://www.cs.utexas.edu/users/novak/reso.html)

KB
- All hounds howl at night.
- Anyone who has any cats will not have any mice.
- Light sleepers do not have anything that howls at night.
- John has either a cat or a hound.

Question: if John is a light sleeper, then John does not have any mice.

KB
- $\forall x \, (Hound(x) \supset Howl(x))$
- $\forall x \, \forall y \, \forall z \, ((Have(x,y) \wedge Cat(y)) \supset \neg (Have(x,z) \wedge Mouse(z)))$
- $\forall x \, \forall y \, (Ls(x) \supset \neg (Have(x,y) \wedge Howl(y)))$
- $\exists x \, (Have(john, x) \wedge (Cat(x) \vee Hound(x)))$

Q: $\forall x \, (Ls(john) \supset \neg (Have(john, x) \wedge Mouse(x)))$

↓ CNF

$[\neg Hound(x), Howl(x)]$

$[\neg Have(x,y), \neg Cat(y), \neg Have(x,z), \neg Mouse(z)]$

$[\neg Ls(x), \neg Have(x,y), \neg Howl(y)]$

$[Have(john, a)]$ a - Skolem constant

$[Cat(a), Hound(a)]$

$[Ls(john)]$

$[Have(john, b)]$ b - Skolem constant

$[Mouse(b)]$

... apply Resolution

$[\,]$

Equality

If we treated $=$ as a predicate, we would miss, for example, that $\{a=b, b=c, a\neq c\}$ is unsatisfiable. For that reason, it is necessary to add the clausal versions of the axioms of equality:

- reflexivity $\quad \forall x.\ x=x$
- symmetry $\quad \forall x \forall y.\ x=y \supset y=x$
- transitivity $\quad \forall x \forall y \forall z.\ x=y \wedge y=z \supset x=z$
- substitution for functions

$$\forall x_1 \forall y_1 \ldots \forall x_n \forall y_n.\ x_1=y_1 \wedge \ldots \wedge x_n=y_n \supset f(x_1,\ldots,x_n)=f(y_1,\ldots,y_n)$$

for every function symbol f of arity n

- substitution for predicates

$$\forall x_1 \forall y_1 \ldots \forall x_n \forall y_n.\ x_1=y_1 \wedge \ldots \wedge x_n=y_n \supset P(x_1,\ldots,x_n) \equiv P(y_1,\ldots,y_n)$$

for every predicate symbol P of arity n

Now $=$ can be treated as a binary predicate

Example 3

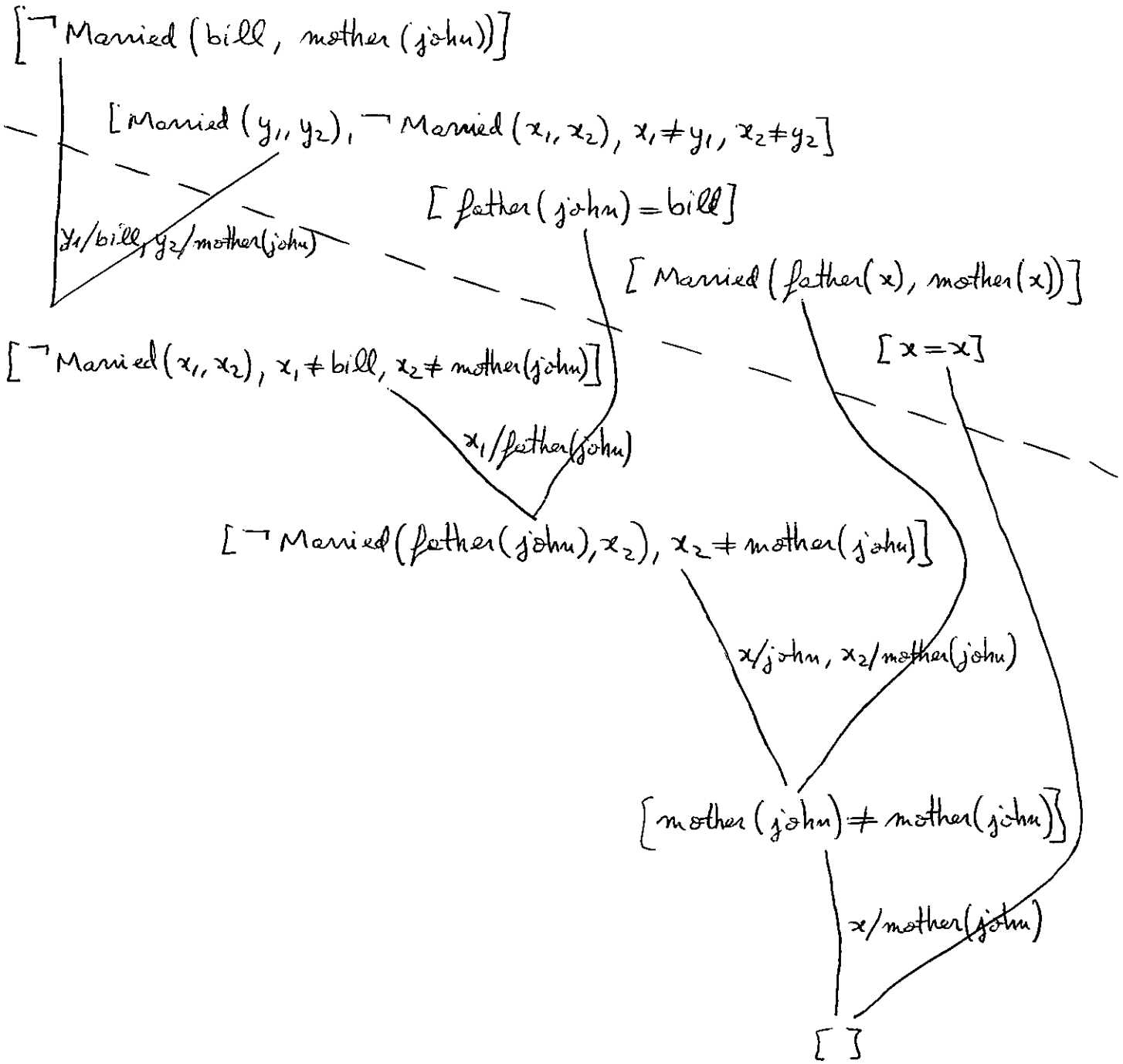
$$KB \begin{cases} \forall x.\ \text{Married}(\text{father}(x), \text{mother}(x)) \\ \text{father}(\text{john}) = \text{bill} \end{cases}$$

Question: $\text{Married}(\text{bill}, \text{mother}(\text{john}))$

$$\forall x_1 \forall y_1 \forall x_2 \forall y_2.\ x_1=y_1 \wedge x_2=y_2 \supset \text{Married}(x_1,x_2) \equiv \text{Married}(y_1,y_2)$$

$\downarrow$ CNF (replace $\supset$ and $\equiv$; distribute $\wedge$ over $\vee$; and collect terms)

$$[\text{Married}(y_1,y_2), \neg \text{Married}(x_1,x_2), x_1 \neq y_1, x_2 \neq y_2]$$



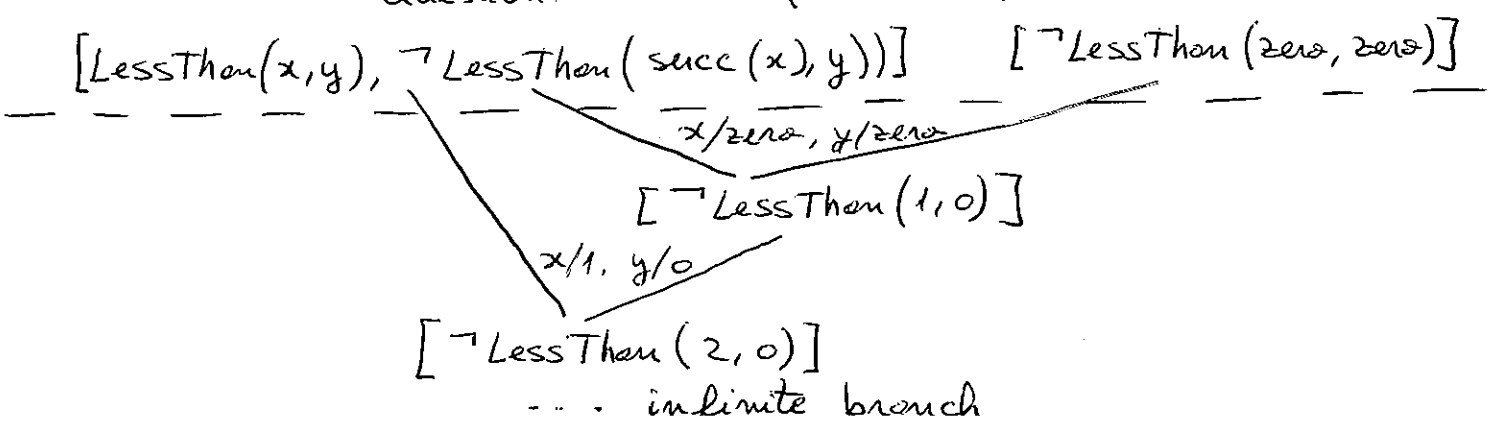
Dealing with computational intractability

Resolution does not provide a general effective solution for automated reasoning.

The FOL Case

KB: $\forall x \forall y.\ LessThan(succ(x), y) \supset LessThan(x, y)$

Question: LessThan(zero, zero).



if we apply a depth-first procedure to search for the empty clause, we can go on an infinite branch.
We cannot detect whether a branch will continue indefinitely. But we know that $S \models [\,]$ iff $S \vdash [\,]$, meaning that if a set of sentences is unsatisfiable, then there is a branch in the derivation containing the empty clause. So, a breadth-first search would guarantee to find it (the unsatisfiable case). But when the clauses are satisfiable, the search may or may not end.

The Herbrand Theorem

In the propositional case, Resolution procedure always terminates (in the initial set of clauses there is a finite number of literals).
In some cases, Resolution in FOL reduces to the propositional case.

Def. Given S a set of clauses, the Herbrand universe of S, written H_S, is the set of all ground terms (i.e. terms with no variables) formed using just the constants and the function symbols in S (if S has no constants or function symbols, we use just a constant a).

For example, if $S = \{[\neg P(x, f(x,a)),\ \neg Q(x,a),\ R(x,b)]\}$
then $H_S = \{a, b, f(a,a), f(a,b), f(b,a), f(b,b), f(a, f(a,a)), \ldots\}$

Def. The Herbrand base of S, written $H_S(S)$, is the set of all ground clauses $C\theta$, where $C \in S$ and θ assigns the variables in C to terms in the Herbrand universe.

For the some S as before, we have:

$$H_S(S) = \{ [\neg P(a, f(a,a)), \neg Q(a,a), R(a,b)],$$
$$[\neg P(b, f(b,a)), \neg Q(b,a), R(b,b)],$$
$$[\neg P(f(a,a), f(f(a,a),a)), \neg Q(f(a,a),a), R(f(a,a),b)],$$
$$[\neg P(f(b,a), f(f(b,a),a)), \neg Q(f(b,a),a), R(f(b,a),b)], \ldots \}$$

Herbrand's Theorem - A set of clauses is satisfiable iff its Herbrand base is satisfiable.

This is important because the Herbrand base is a set of clauses without variables, so it is essentially propositional. The difficulty is that typically the Herbrand base is an infinite set of propositional clauses (but finite when the Herbrand universe is finite - no function symbols and a finite number of constants in S).
Sometimes, the Herbrand univers can be kept finite by looking at the type of the arguments and values of functions, and including terms like $f(t)$ only if the type of t is appropiate for f (e.g. we may exclude terms like birthday(birthday(john)) from the Herbrand universe).

The propositional case

How long may it take for the Resolution procedure on a finite set of propositional clauses to terminate?

In 1985, Armin Haken proved that there exist unsatisfiable propositional clauses $c_1, \ldots, c_n$ so that the shortest derivation of the empty clause has the length of order 2^n.
Resolution takes an exponential time, no matter how well we choose the derivations.

Is there a better way to determine whether a set of propositional clauses is satisfiable? - one of the most difficult questions in computer science.

In 1972, Stephen Cook proved that the satisfiability problem was NP-complete. Any search problem (e.g. scheduling, routing) where we are searching for an element that satisfies a certain property, and where we can test in polynomial time whether a candidate satisfies the property, can be converted into a propositional satisfiability problem. Thus, a polynomial time algorithm for satisfiability would imply a polynomial time for all these search problems.

We may need to consider alternative options for Resolution:

- procedural representations - give more control over the reasoning process to the user.
- using representation languages that are less expressive than FOL - e.g. description languages

The research in KRR approaches both directions.

In some applications it may be worth waiting (for a long time) for answers.

There is an area of AI called automated theorem-proving, that uses Resolution (among other procedures) for this purpose (e.g. to determine whether or not Goldbach's Conjecture follows from the axioms of number theory).

SAT Solvers

They are procedures that determine the satisfiability of a set of clauses more efficiently than the Resolution. They search for an interpretation that would prove the clauses to be satisfiable. They are often applied to clauses that are known to be satisfiable, but the satisfying interpretation is not known.

When C is a set of clauses and m is a literal, $C \bullet m$ is defined as following:

$$C \bullet m = \{c \mid c \in C,\ m \notin c,\ \overline{m} \notin c\} \cup \{(c - \overline{m}) \mid c \in C,\ m \notin c,\ \overline{m} \in c\}$$

For example, if $C = \{[p,q], [\overline{p}, a, b], [\overline{p}, c], [d, e]\}$

then $C \bullet p = \{[a,b], [c], [d,e]\}$

$C \bullet \overline{p} = \{[q], [d,e]\}$

$(C \bullet \overline{p}) \bullet \overline{q} = \{[], [d,e]\}$

$(C \bullet \overline{p}) \bullet q = \{[d,e]\}$

$((C \bullet \overline{p}) \bullet q) \bullet d = \{\}$

Given an interpretation I,

- if p is true then C is satisfiable iff $C \bullet p$ is satisfiable
- if p is false then C is satisfiable iff $C \bullet \overline{p}$ is satisfiable.

Procedure DP (Davis - Putnam)

input: a set of clauses C
output: are the clauses satisfiable: YES or NO

```
procedure DP(C)
    if (C is empty) then return YES
    if (C contains [ ]) then return NO
    let p be some atom in C
    if (DP(C • p) = YES) then return YES
                         else return DP(C • p̄)
```

Strategies for choosing an atom p:
- p appears in the most clauses in C;
- p appears in the fewest clauses in C;
- p is the most balanced atom in C (i.e. the number of positive occurences in C is closest to the number of negative occurences);
- p is the least balanced atom in C;
- p appears in the shortest clause in C.

For the propositional case, the DP procedure is the fastest one in practice, among all the known SAT solvers. Thus, problems with tens of millions of variables can be approached. SAT solvers revolutionized fields such as hardware verification or security protocols verification.

Most general unifiers

The most efficient way to avoid unnecessary search in a first-order derivation is to keep the search as general as possible.

For example, consider the clause C_1 containing the literal $P(g(x), f(x), z)$ and the clause C_2 with $\neg P(y, f(w), a)$.

For unification, we may have the substitution

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

or

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

or

...

We can try to derive the empty clause using θ_1; if it doesn't work, we try with θ_2 and so on.

θ_1 and θ_2 are more specific than they should be (it is not necessary to give a value for x).

The substitution $\theta_3 = \{y/g(x), z/a, w/x\}$ unifies C_1 and C_2 without making unnecessary arbitrary choices that might exclude a path to the empty clause.
θ_3 is a most general unifier (MGU). It may not be unique — for example $\theta_4 = \{y/g(w), z/a, x/w\}$ is also an MGU.

Def. A most general unifier θ of literals ℓ_1 and ℓ_2 is a unifier that has the property that for any other unifier θ', there is a substitution θ^* such that $\theta' = \theta \cdot \theta^*$. By $\theta \cdot \theta^*$ we mean that we first apply θ and then apply θ^* to the result.

For example, from θ_3 we can get to θ_1 by further applying x/b; to θ_2 by applying $x/f(z)$; and to θ_4 by applying x/w.
By limiting Resolution to MGUs, the completeness is maintained and the number of resolvents is dramatically reduced.

The procedure for computing an MGU

input: literals ℓ_1 and ℓ_2
output: a substitution θ

1. $\theta = \{\}$
2. if $(\ell_1\theta = \ell_2\theta)$ then exit
3. determine the disagreement set DS, which is the pair of terms at the first place (from left to right) where the two literals disagree. For example,
   if $\ell_1\theta = P(a, f(a, \underline{g(z)}, \ldots))$ and $\ell_2\theta = P(a, f(a, \underline{u}, \ldots))$ then $DS = \{u, g(z)\}$
4. find a variable $v \in DS$ and a term $t \in DS$ not containing v; if none, fail.
5. otherwise, set $\theta = \theta \cdot \{v/t\}$ and go to 2.

The procedure is very efficient in practice.
All Resolution-based systems use MGUs.

Other optimizations to Resolution to improve the search

Clause elimination

There are types of clauses that do not participate in the (shortest) derivation to the empty clause:

- pure clauses – contain some literal ℓ such that $\bar{\ell}$ does not appear anywhere else.
- tautologies – contain both ℓ and $\bar{\ell}$ and they can be bypassed in any derivation.
- subsumed clauses – clauses for which there already exists another clause with a subset of the literals (i.e. clauses more specific than a clause in KB).
  For example, if $[P(x)] \in KB$ then we do not add $[P(a)]$ or $[P(a), Q(b)]$.
  if we have $[p, r]$, we don't need $[p, q, r]$.

Ordering strategies

- choose a predefined order to perform Resolution to maximize the chance of deriving the empty clause.
- the best strategy up-to-date is "unit preference", that is, to use unit clauses first.
  a unit clause + a clause with k literals ⇒ a clause of length k-1 ...

Special treatment of equality

The explicit use of the axioms of equality can generate many resolvents. A way to avoid that is by introducing a second rule of inference in addition to Resolution, called Paramodulation.

We are given two clauses:

$C_1 \cup \{t = s\}$ where t and s are terms

$C_2 \cup \{\rho[t']\}$ containing some term t'.

if necessary, we rename the variables in the two clauses to be distinct.

We assume that there is a substitution θ such that $t\theta = t'\theta$.

Then we can infer the clause $(C_1 \cup C_2 \cup \rho[s])\theta$, which eliminates $=$, replaces t' by s and perform substitution θ.

In Example 3, we have

$$[\underbrace{father(john)}_{t} = \underbrace{bill}_{s}] \qquad [\underbrace{Married}_{\rho}(\underbrace{father(x)}_{t'}, mother(x))]$$

$C_1 = \{\}$ $\qquad C_2 = \{\}$

$\theta = \{x/john\}$

We can derive $[Married(bill, mother(john))]$ in a single Paramodulation step.

Directional connectives

A clause like $[\neg p, q]$ representing $p \supset q$ can be used in two directions in derivation:

- forward – if we derive a clause containing p, then we derive the clause containing q
- backward – if we derive a clause containing $\neg q$, then we derive the clause containing $\neg p$.

We can mark clauses to be used in one or the other direction only (with care not to lose completeness).

For example, if we have in KB

$$\forall x.\ Battleship(x) \supset Gray(x)$$

it may be used only in the forward direction (it might not be such a good idea to prove that something is not a battleship if it is not gray.

Horn clauses

Horn clauses are a subset of FOL, where the Resolution procedure works well. This subset is sufficiently expressive for many problems.
In a Resolution-based system, the clauses are used for two purposes:
1. To express disjunctions like [Rain, Sleet, Snow] to represent incomplete knowledge.
2. To express a conditional - disjunctions like [¬Child, ¬Male, Boy] - although it can be read as "someone is not a child, or is not male, or is a boy", it is more natural to be understood as a conditional "if someone is a child and a male then is a boy".

Def. A Horn clause contains at most one positive literal.
A clause with no positive literals is called a negative Horn clause.

Obs. The empty clause is a negative Horn clause.

The positive Horn clause $[\neg p_1, \ldots, \neg p_n, q]$ can be read "if p_1 and ... and p_n then q". It is called "rule" and it is written as $p_1 \wedge \ldots \wedge p_n \Rightarrow q$ to emphasize the conditional.

Resolution derivations with Horn clauses

Obs. Two negative clauses cannot resolve together.

A negative and a positive clause produce a negative clause by Resolution.

Two positive clauses produce a positive clause.

Resolution over Horn clauses involves always a positive clause.

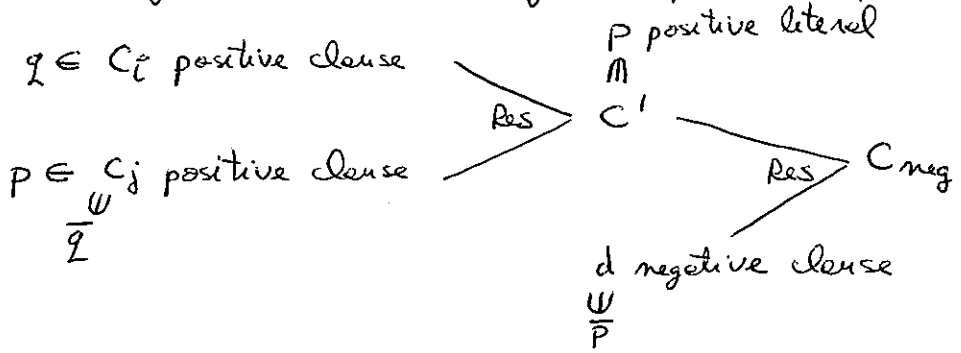
Prop. Given S a set of Horn clauses and $S \vdash c$, where c is a negative clause, then there exists a derivation of c where all the new clauses in the derivation (i.e. clauses not in S) are negative.

Proof. [$C_1, \ldots, C_n = C$ is a derivation iff $C_i \in S$ or C_i is a resolvent of two previous clauses in the sequence]

Suppose we have a derivation with new positive clauses. Let C' be the last one:

$$C_1, \ldots, \underbrace{C'}_{\text{positive}}, \underbrace{\ldots C_n = C}_{\text{negative}}$$

Instead of producing negative clauses using C', we will generate these negative clauses using the positive parents of C'.



C_i Res $C_j \longrightarrow C'$ $\qquad$ C' Res $d \longrightarrow C_{neg}$

Replaced by $(C_j \text{ Res } d)$ Res $C_i \longrightarrow C_{neg}$

The derivation still produces C_{neg}, but without using C'. We remove C' from the derivation and repeat this for every new positive clause introduced. Thus, we eliminate all of them.

Prop. Given S a set of Horn clauses and $S \vdash C$, where C is a negative clause, then there exists a derivation of C, where each new clause derived is negative and is a resolvent of the previous one in the derivation and a clause from S.

Proof. Using the previous Prop., we can assume that all new clauses in the derivation are negative. So, all the positive clauses are from S.

$$C_1, \ldots, \underbrace{C'}_{\text{new negative clause}}, \ldots C_n = C$$

C_i positive $\in S$, C_j negative $\xrightarrow{\text{Res}} C'$

$S \ni C_K$ negative $\xrightarrow{\text{Res}} C'_K$ positive $\in S$ $\quad \ldots \quad C_2$ negative $\xrightarrow{\text{Res}} C'_2$ pos $\in S$ $\quad C_1$ neg $\xrightarrow{\text{Res}} C'_1$ pos $\in S$ $\quad \xrightarrow{\text{Res}} C$

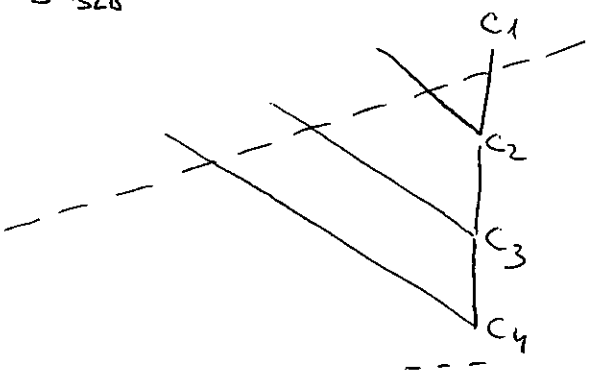
and we discard all the clauses that are not in this chain.

Given S a set of Horn clauses, there is a derivation of a negative clause (including $[\,]$) iff there is one where each new clause in the derivation is a negative resolvent of the previous clause in the derivation and a clause from S.

SLD Resolution (Selected literals, Linear pattern, over Definite clauses)

It is a restricted form of Resolution, where each new clause is a resolvent of the previous clause and a clause from the original set S. This version of Resolution is sufficient for Horn clauses.

Def. If S is a set of clauses (not necessarily Horn), an SLD derivation is a sequence $C_1, \ldots, C_n = C$ where $C_1 \in S$ and C_{i+1} is a resolvent of C_i and a clause in S. We write $S \vdash_{SLD} C$.



Except for C_1, the elements of S are not explicitly mentioned.

It is clear that if $S \vdash_{SLD} [\,]$ then $S \vdash [\,]$, but the converse doesn't hold.

For example, for $S = \{[p,q], [\neg p, q], [p, \neg q], [\neg p, \neg q]\}$ we have that $S \vdash [\,]$.

To generate $[\,]$, the last step in Resolution should involve $[\ell]$ and $[\bar{\ell}]$ for some literal ℓ. But S does not contain any unit clauses, so there is no element from S in the last step of Resolution. That means that $S \not\vdash_{SLD} [\,]$.

But for S a set of Horn clauses, then $S \vdash [\,]$ iff $S \vdash_{SLD} [\,]$.

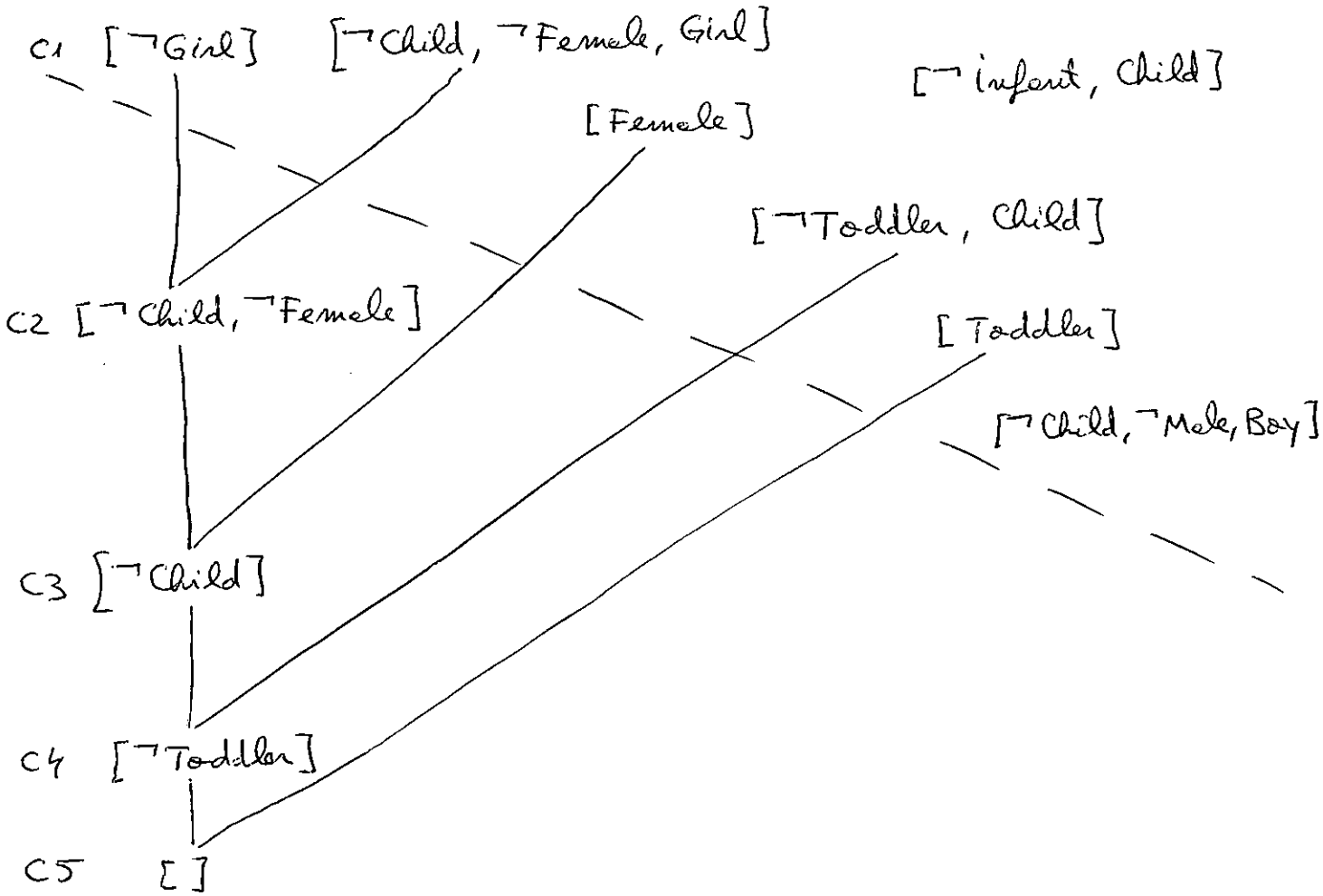
Moreover, each of the new clauses in the derivation $C_2, \ldots, C_n$ can be assumed to be negative.
C_2 has a negative and a positive parent, so C_1 can be chosen to be the negative one.
Thus, for Horn clauses, SLD derivations of the empty clause begin with a negative clause in S.

Example 1

KB Horn clauses:
- Toddler
- Toddler ⊃ Child
- Child ∧ Male ⊃ Boy
- infant ⊃ Child
- Child ∧ Female ⊃ Girl
- Female

Question: Girl

Obs. [¬Girl] is the only negative clause in S, so $c_1 = [\neg Girl]$.



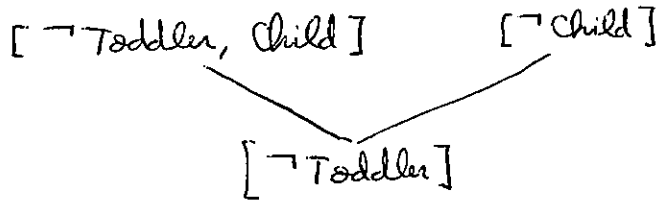
SLD derivation: C_1, C_2, C_3, C_4, C_5

Goal trees

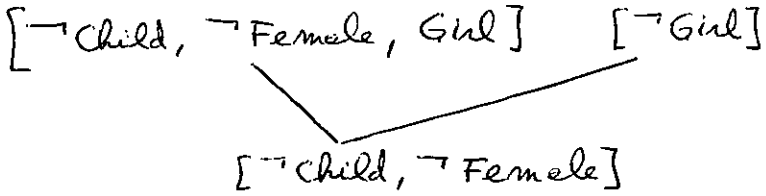
All the literals in all the clauses in a Horn SLD derivation of the empty clause are negative. To produce [ ], we need positive Horn clauses to eliminate the negative literals.

For example, if we have a unit positive clause in S [Toddler] and [¬ Toddler] in a derivation, we say that the goal Toddler is solved.

if a positive Horn clause introduces other negative literals



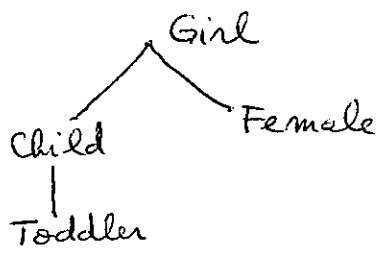
we say that the goal Child reduces to the subgoal Toddler.



the goal Girl reduces to two subgoals: Child and Female.

In Example 1, the SLD derivation can be reformulated as following: we start with the goal Girl; this reduces to two subgoals Child and Female; the goal Female is solved; Child reduces to Toddler; Toddler is solved.

The essociated goal tree is:



For a complete SLD derivation, the leaves of the tree are solved goals.

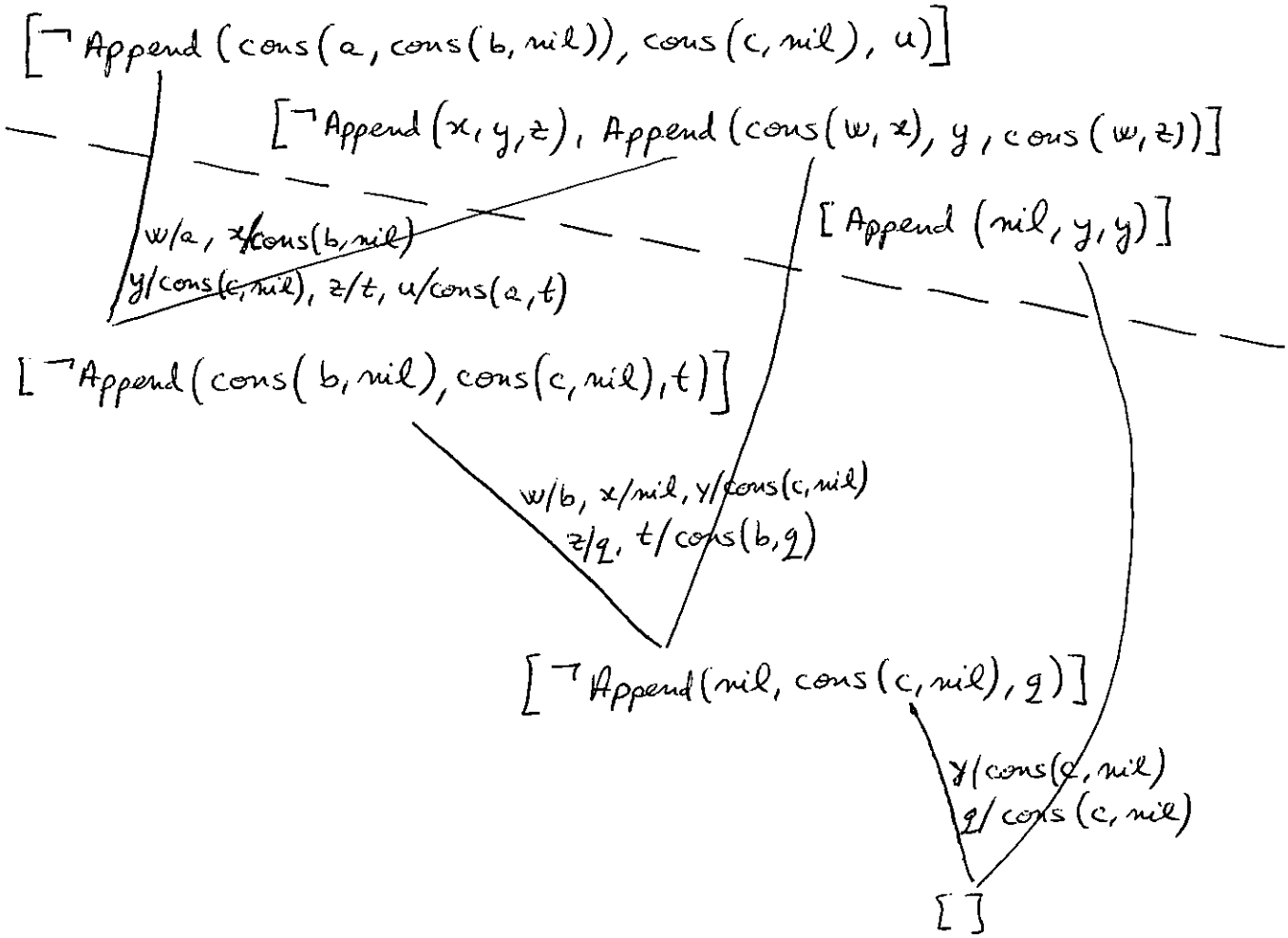
The Horn clauses and SLD derivations, represented as goal trees, are the basis of PROLOG.

Example 2 Concatenation of lists

KB
- Append(nil, y, y)
- Append(x, y, z) ⇒ Append(cons(w, x), y, cons(w, z))

Question: ∃u. Append(cons(a, cons(b, nil)), cons(c, nil), u)

nil - empty list
cons(a, nil) – [a]
cons(a, cons(b, nil)) – [a, b]



The associated goal tree is:

[Append(cons(a, cons(b, nil)), cons(c, nil), u)]

|

[Append(cons(b, nil), cons(c, nil), t)]

|

[Append(nil, cons(c, nil), q)]

The answer u = cons(a, cons(b, cons(c, nil))) can be extracted from the derivation:

q/cons(c, nil) ⟶ t/cons(b, q) ⟶ u/cons(a, t)

Computing SLD derivations

Given KB, a set of positive Horn clauses, we want to determine whether a set of atoms can be entailed from KB.

The case considered here consists of determining the satisfiability of a set of Horn clauses containing exactly one negative clause.

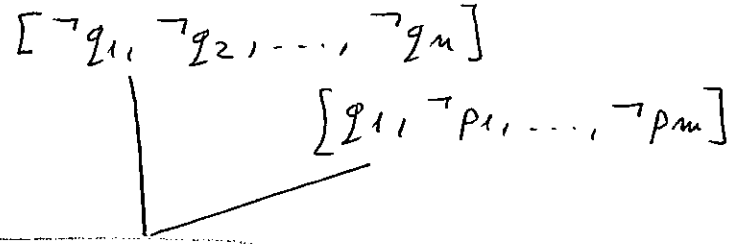
Backward chaining

input: KB and a finite list of atomic sentences $q_1, \dots, q_n$

output: YES or NOT - whether or not KB entails all q_i

```
procedure SOLVE [q1, ..., qn]
    if (n = 0) then return YES
    for each clause c ∈ KB
        if (c = [q1, ¬p1, ..., ¬pm] and SOLVE [p1, ..., pm, q2, ..., qn])
                then return YES
    return NO
```

The search goes backward, from goals to facts in KB.



the subgoals the goal q_1 reduces to

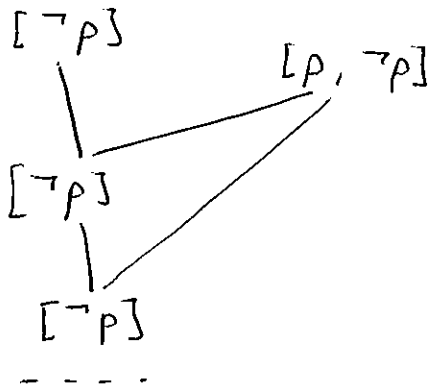
– if it fails, it tries with another clause in KB whose positive literal is q_1. if none is found then return NO.

The procedure works in a depth-first manner, as it attempts to solve the new goals p_i before the old ones q_i.

it is called left-to-right because it solves the goals $q_1, \dots, q_n$ in order $1, 2, \dots, n$.

This is how PROLOG solves goals.

Obs. The procedure can go into an infinite loop.



In other cases, the procedure can be exponential.

For example,

$$KB \begin{cases} p_{i-1} \Rightarrow p_i \\ p_{i-1} \Rightarrow q_i \\ q_{i-1} \Rightarrow p_i \\ q_{i-1} \Rightarrow q_i \end{cases} \qquad 0 < i < n \qquad 4n-4 \text{ clauses}$$

Question: p_i (or q_i) - neither is entailed by KB

$i=1$ $[\neg p_0, p_1]$ $[\neg q_0, p_1]$ $[\neg p_1]$ $[\neg p_0]$ $[\neg q_0]$ 2^1 steps

assume that for k-1, at least 2^{k-1} steps are necessary

$p_{k-1} \Rightarrow p_k$ at least $2^{k-1} + 2^{k-1} = 2^k$ steps necessary

$q_{k-1} \Rightarrow p_k$ to show that p_k is not entailed by KB.

Forward chaining

The procedure works from the facts in KB towards the goals.

input: KB and a finite list of atomic sentences $q_1, \dots, q_n$

output: YES or NOT - whether or not KB entails all q_i

procedure

1. if (all of the goals q_i are solved) then return YES
2. check if there is a clause $[p, \neg p_1, \dots, \neg p_m]$ in KB, such that all of its negative atoms $p_1, \dots, p_m$ are marked as solved and the positive atom p is not solved.
3. if (there is such a clause) then mark p as solved and go to step 1
   else return NO

□

We mark atoms as solved when we determine that they are entailed by KB.

In Example 1 [Toddler] has no negative atoms – it is marked as solved

[Child, ¬Toddler] – Child is marked

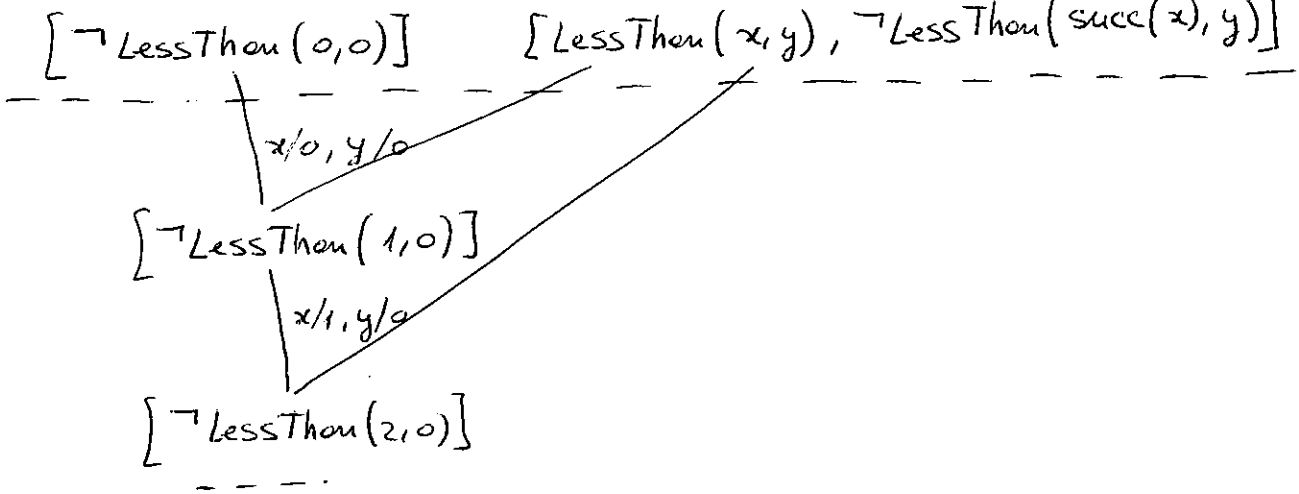
[Female] – has no negative atoms – it is marked

[Girl, ¬Child, ¬Female] – Girl is marked – return YES.

The forward chaining has a much better overall behavior than the backward chaining.

At each iteration, we search for a clause in KB with an atom that has not been marked. The overall result will not be exponential.

In the propositional case, we can determine whether or not a Horn KB entails an atom. But in FOL, the forward chaining procedure may not terminate.



The problem of determining whether a set of Horn clauses in FOL entails an atom is undecidable.

Procedural control of reasoning

Automated proving methods answer a question by trying all logically permissible options in the knowledge base. These reasoning methods are domain-independent. But in some situations it is not feasible to search all logically possible ways to find a solution.
We often have an idea about how to use knowledge and we can "guide" an automated procedure based on properties of the domain.
We will see how knowledge can be expressed to control the backward-chaining reasoning procedure.

Facts and rules

The clauses in a KB can be divided in two categories:

- facts – are ground atoms (without variables).
- rules – are conditionals that express new relations – they are universally quantified.

$Mother(jane, john)$
$Father(john, bill)$
...
$Parent(x, y) \Leftarrow Mother(x, y)$
$Parent(x, y) \Leftarrow Father(x, y)$

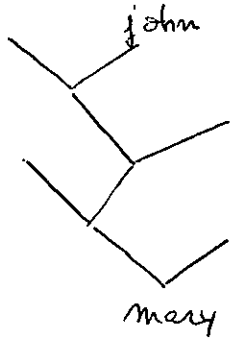
Rules involve chaining and the control issue regards the use of the rules to make it most effective.

Rule formation and search strategies

We can express the Ancestor relation in three logically equivalent ways:

1. $Ancestor(x, y) \Leftarrow Parent(x, y)$
   $Ancestor(x, y) \Leftarrow Parent(x, z) \wedge Ancestor(z, y)$
2. $Ancestor(x, y) \Leftarrow Parent(x, y)$
   $Ancestor(x, y) \Leftarrow Parent(z, y) \wedge Ancestor(x, z)$

3. $Ancestor(x,y) \Leftarrow Parent(x,y)$
$Ancestor(x,y) \Leftarrow Ancestor(x,z) \wedge Ancestor(z,y)$



1. We search top-down in the family tree.
2. We search down-top.
3. We search in both directions in the same time.

If people had on average one child, then 1) would be of order d and 2) of order 2^d, where d is the depth of search. if people had more than 2 children, 2) would be a better option.

Algorithm design

The Fibonacci series $\begin{cases} x_0 = 0 \\ x_1 = 1 \\ x_{n+2} = x_{n+1} + x_n, \; n \geq 0 \end{cases}$

KB $\begin{cases} Fib(0,1) \\ Fib(1,1) \\ Fib(s(s(n)),v) \Leftarrow Fib(n,y) \wedge Fib(s(n),z) \wedge Plus(y,z,v) \\ Plus(0,z,z) \\ Plus(s(x),y,s(z)) \Leftarrow Plus(x,y,z) \end{cases}$

Most of the computation is redundant

$Fib(10,\_)$ calls $Fib(9,\_)$ and $Fib(8,\_)$

$Fib(11,\_)$ calls $Fib(10,\_)$ and $Fib(9,\_)$

Each application of Fib calls Fib twice and it generates an exponential number of Plus subgoals.

An alternative is $\begin{cases} Fib(n,v) \Leftarrow F(n,1,0,v) \\ F(0,y,z,y) \\ F(s(n),y,z,v) \Leftarrow Plus(y,z,s) \wedge F(n,s,y,v) \end{cases}$

$F(n,\underbrace{-,-}_{\text{2 consecutive Fib numbers}},v)$ — the solution obtained when n is 0

starts from n towards 0

Goal order

From logical point of view, all ordering of subgoals are equivalent, but the computational differences can be significant.

For example

$$AmericanCousin(x,y) \Leftarrow American(x) \wedge Cousin(x,y)$$

we have two options:

- find an American and see if he/she is a cousin
- find a cousin and see if he/she is American

In this case, solving first $Cousin(x,y)$ and then $American(x)$ is better than the other way around.

Predicate ! ("cut") in PROLOG – backtracking control and negation as failure

! is always true; it prevents backtracking in the place of occurrence in the program.
If ! doesn't change the declarative meaning of the program, then it is called green; otherwise it is red.

The function $f(x) = \begin{cases} 0, & x \leq 3 \\ 2, & x \in (3,6] \\ 4, & x > 6 \end{cases}$ can be implemented as:

KB
```
f(X, 0) :- X =< 3.
f(X, 2) :- 3 < X, X =< 6.
f(X, 4) :- 6 < X.
```

```
?- f(1,Y), 2 < Y.
```

$X=1, Y=0$ $\quad 1 =< 3, 2 < 0$ false

$X=1, Y=2$ false

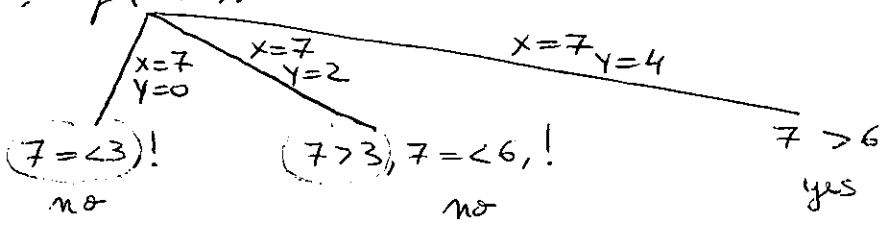
$X=1, Y=4$ false

The program should have stopped after the first check.

```
f(X,0) :- X =< 3, !.
f(X,2) :- 3 < X, X =< 6, !.
f(X,4) :- 6 < X.
```

green!

?- f(7,Y).



redundante tests

```
f(X,0) :- X =< 3, !.
f(X,2) :- X =< 6, !.
f(X,4).
```

red! - if we remove ! and ask:

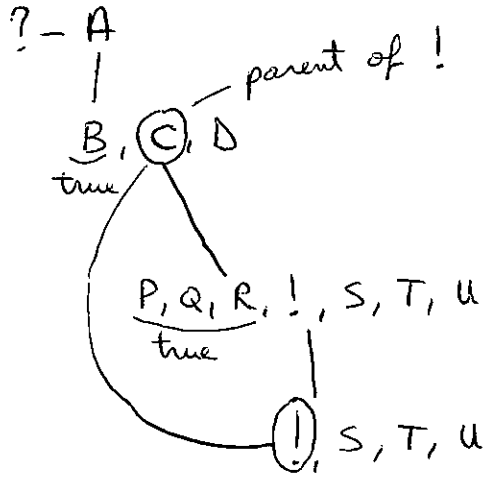
?- f(1,Y).
Y=0;
Y=2;
Y=4;
false

The parent of a "cut" is that PROLOG goal that matches the head of the rule that contains that "cut".

```
C :- P, Q, R, !, S, T, U.
C :- V.
A :- B, C, D.
```

?- A.

Backtracking is possible for P, Q, R, but as soon as ! is executed, all of the alternative solutions are suppressed. Also, the alternative C :- V will be suppressed.



- in the goal tree, backtracking is prevented between ! and its parent.
- ! affects only the execution of C.

```
max(X, Y, X) :- X >= Y, !.
max(X, Y, Y).
```
red!

```
member(X, [X|L]) :- !.
member(X, [_|L]) :- member(X, L).
```

Given KB
```
p(1).
p(2) :- !.
p(3).
```
what are PROLOG answers to the following questions:

```
?- p(X).
?- p(X), p(Y).
?- p(X), !, p(Y).
```

Negation as failure - predicate "fail" is always false.

John likes all animals, with the exception of snakes.

```
likes(john, X) :- snake(X), !, fail.
likes(john, X) :- animal(X).
```

We define the unary predicate "not" as following: not(G) fails if G succeeds; otherwise not(G) succeeds.

```
not(G) :- G, !, fail.
not(G).
```

```
likes(john, X) :- animal(X), not(snake(X)).
```

Procedurally, we distinguish between two types of negative situations with respect to a goal G:

- being able to solve $\neg G$
- being unable to solve G - this can happen when we run out of options when trying to prove that G is true.

"Not" in PROLOG doesn't correspond exactly to the mathematical negation. When PROLOG processes a "not" goal, it doesn't try to solve it directly, but to solve the opposite.

If the opposite cannot be demonstrated, then PROLOG assumes that the "not" goal is solved.

Such a reasoning is based on the Closed-World Assumption. That is to say that if something is not in the KB or it cannot be derived from the KB, then it is not true and consequently, its negation is true.

For example, if we ask:

```
?- not(human(mary)).
```

the answer is "yes" if human(mary) is not in KB. But it should not be understood as "Mary is not a human being", but rather "there is not enough information in the program to prove that Mary is a human being".

Usually, we do not assume the "Closed-World" – if we do not explicitly say "human(mary)", we do not implicitly understand that Mary is not a human being.

Other examples:

1)
```
composite(N) :- N>1, not(primeNumber(N)).
```

The failure to prove that a number greater than 1 is prime is sufficient to conclude that the number is composite.

2)
```
good(renault).
good(audi).
expensive(audi).
reasonable(Car) :- not(expensive(Car)).
```

```
?- good(x), reasonable(x).
```

```
?- reasonable(x), good(x).
```

! is useful and, in many situations, necessary, but it must be used with special attention.

Rules in production system

Production systems formalize knowledge in a certain form called production rules and use forward-chaining reasoning to derive new things. They are used in many practical applications (e.g. expert systems).

A production system maintains a working memory (WM) which contains assertions that are changing during the operation of the system.

A production rule consists of an antecedent set of conditions and a consequent set of actions:

IF conditions THEN actions

The antecedent conditions are tests applied to the current state of the WM; the consequent actions are a set of actions that modify the WM.

The production system operates in a three-step cycle that repeats until no more rules are applicable to the WM:

1. recognize – find the applicable rules, i.e. those rules whose antecedent conditions are satisfied by the current WM.
2. resolve conflicts – among the rules found at the first step (called conflict set), choose those to fire.
3. act – change the WM by performing the consequent actions of all the rules selected at step 2.

Working Memory

WM consists of a set of working memory elements (WMEs).

A WME has the form:

(type $attribute_1$: $value_1$ ... $attribute_n$: $value_n$)

where type, $attribute_i$ and $value_i$ are atoms.

Examples (person age: 21 home: bucharest)
(student name: john dept: computerScience)

Declaratively, a WME is an existential sentence:

$$\exists x \, [\, type(x) \wedge attribute_1(x) = value_1 \wedge \ldots \wedge attribute_n(x) = value_n \,]$$

WMEs represent objects and relationships between them can be handled by reification (i.e. transforming a sentence into an object).

$$\left[\begin{array}{l} Purchases(\, john,\ bike,\ nov\ 9) \\ \qquad\qquad \downarrow \\ Purchase(p3) \wedge agent(p3) = john \wedge object(p3) = bike \ldots \end{array}\right.$$

For example, (basicFact relation: olderThan firstArg: john secondArg: mary) may be used to express the fact that John is older than Mary.

Production rules

The conditions of a rule are connected by conjunctions. A condition can be positive or negative (written as -cond) and it has the form:

(type attribute$_1$: specification$_1$ ... attribute$_K$: specification$_K$)

where each specification is one of the following:

- an atom
- a variable
- an evaluable expression, within [ ]
- a test, within { }
- conjunction, disjunction or negation of a specification ($\wedge$, $\vee$, $\neg$)

For instance, (person age: [n+1] height: x) is satisfied if there is a WME whose type is person and whose age attribute is n+1, where n is specified elsewhere. If the variable x is not bound, then x will be bound with the value of the attribute height; otherwise, the value of the attribute height in the WME has to be the same as the value of x.

— (person age : {< 20 ∧ > 6}) is satisfied if there is no WME in WM whose type is person and age value is between 6 and 20 (negation as failure).

A WME matches a condition if the types are identical and for each attribute/specification pair in the condition, there is a corresponding attribute/value pair in the WME and the value matches the specification. The matching WME may have other attributes that are not mentioned in the condition:

condition (control phase : 5)
matching WME (control phase : 5 position : 7 ...)

The consequent set of actions of production rules have a procedural interpretation. All actions are executed in sequence and they can be of the following types:

1. ADD — adds a new WME to the WM.
2. REMOVE i — removes from WM the WME that matches the i-th condition in the antecedent of the rule; not applicable if the condition is negative.
3. MODIFY i (attribute specification) — modifies the WME that matches the i-th condition in the antecedent by replacing its current value for attribute by specification; not applicable if the condition is negative.

Obs. In ADD and MODIFY, the variables that appear refer to the values obtained when matching the antecedent of the rule.

Example 1 IF (student name : x) THEN ADD (person name : x)
(the equivalent of $\forall x.\ Student(x) \supset Person(x)$)

Example 2 - assume that a rule added a WME of type birthday

IF (person age : x name : n) (birthday who : n)
THEN MODIFY 1 (age [x+1])
REMOVE 2
↖ to prevent the rule from firing again

Example 3 - to indicate the phase of a computation

```
IF (starting) THEN REMOVE 1
                   ADD (control phase : 1)
...
IF (control phase : x) ... other conditions
      THEN MODIFY 1 (phase [x+1])

IF (control phase : 5) THEN REMOVE 1
```

Example 4 We have three cubes, each of different size. With a robotic hand, we want to move the cubes in the positions called 1, 2 and 3. The goal is to place the largest cube in position 1, the middle one in 2 and the smallest in position 3.

WM
1. (counter value : 1)
2. (cube name : A size : 10 position : heap)
3. (cube name : B size : 30 position : heap)
4. (cube name : C size : 20 position : heap)

Rules

```
1. IF (cube position : heap name : n size : s)
      - (cube position : heap size : {> s})
      - (cube position : hand)
   THEN MODIFY 1 (position hand)
2. IF (cube position : hand)
      (counter value : i)
   THEN MODIFY 1 (position i)
        MODIFY 2 (value [i+1])
```

there are no conflicts because only one rule can fire at a time

Rule 1 → n = B, s = 30 → WME 3 changes
(cube name : B size : 30 position : hand)

Rule 2 → i = 1 → WME3 changes
(cube name : B size : 30 position : 1)
WME 1 changes (counter value : 2)

Rule 1 → n=C, s=20 → WME 4 changes
(cube name: C size: 20 position: hand)

Rule 2 → i=2 → WME 4 changes
(cube name: C size: 20 position: 2)
WME 1 changes (counter value: 3)

Rule 1 → n=A, s=10 → WME 2 changes
(cube name: A size: 10 position: hand)

Rule 2 → i=3 → WME 2 changes
(cube name: A size: 10 position: 3)
WME 1 changes (counter value: 4)

WM
1. (counter value: 4)
2. (cube name: A size: 10 position: 3)
3. (cube name: B size: 30 position: 1)
4. (cube name: C size: 20 position: 2)

No rule is applicable now, so the production system halts.

Example 5 - compute how many days are in any given year.

if (year ⋮ 4) then (normal year)
else if (year ⋮ 100) then (leap year)
else if (year ⋮ 400) then (normal year)
else (leap year)

WM [(wantDays year: 1953)

Rules
1. IF (wantDays year: n)
THEN REMOVE 1
ADD (year mod4: [n%4] mod100: [n%100] mod400: [n%400])
2. if (year mod4: {≠0})
THEN REMOVE 1
ADD (hasDays days: 365)
3. if (year mod4: 0 mod100: {≠0})
THEN REMOVE 1
ADD (hasDays days: 366)

Rules

4. IF (year mod 100 : 0 mod 400 : {≠0})
   THEN REMOVE 1
   ADD (hasDays days : 365)
5. IF (year mod 400 : 0)
   THEN REMOVE 1
   ADD (hasDays days : 366)

All conditions are disjoint, so one rule fires at a time.

Example 6 - compute the greatest common factor of two integers.

if a > b then gcf(a, b) = gcf(a-b, b)
else gcf(a, b) = gcf(a, b-a)

gcf(0, a) = gcf(a, 0) = a

WM [(gcf val1 : 6 val2 : 9)

Rules

1. IF (gcf val1 : 0 val2 : 0)
   THEN REMOVE 1
   ADD (res val : 0)
2. IF (gcf val1 : 0 val2 : {≠0})
   (gcf val1 : 0 val2 : x)
   THEN REMOVE 1
   ADD (res val : x)
3. similar to 2, for val1 : {≠0} and val2 : 0
4. IF (gcf val1 : {≠0} val2 : {≠0})
   (gcf val1 : a val2 : b)
   (gcf val1 : a) val2 : {>b})
   THEN MODIFY 1 (val1 [a-b])
5. similar to 4, for a ≤ b

All conditions are disjoint, one rule fires at a time

WM [(res val : 3)

Solving conflicts

There are conflict resolution strategies to eliminate some applicable rules, if necessary. The most common approaches are:

- order: choose the first applicable rule in order of presentation (this strategy is implemented in PROLOG).
- specificity: choose the applicable rule whose conditions are most specific.

IF (bird) THEN ADD (canFly)

more specific than the first:
IF (bird weight: {>100}) THEN ADD (cannotFly)
IF (bird) (penguin) THEN ADD (cannotFly)

another criterion should be applied further to choose between the two rules

- recency - choose the applicable rule that matches the most/least recently created/modified WME.
- refractoriness - reject a rule that has just been applied with the same value of its variables - it prevents going into loops by repeated firing of a rule because of the same WME.

The efficiency of production systems

The rule matching operation consumes up to 90% of the operating time of a production system.

Two key observations led to a very efficient implementation:

- WM modifies very little during an execution cycle;
- many rules share common conditions.

The RETE algorithm (1974) creates a network from the rules antecedents. This network can be created in advance, as the rules do not change during the system operation.

During operation, new or changed WMEs are checked if they satisfy the conditions of a rule. In this way, only a small part of WM is re-matched against the conditions of the rules, reducing drastically the time to calculate the applicable rules.

The advantages of production systems

- modularity – because the rules work independently, they can be added or deleted relatively easily.
- control – a simple control structure.
- transparency – the rules use a terminology easy to understand and the reasoning can be traced and explained in natural language.

In real-life applications, these advantages tend to diminish. Nevertheless, production systems are successfully used in a wide variety of practical problems (e.g. medical diagnosis, checking for eligibility for credits, configuration of systems).

MYCIN – expert system for diagnosis of bacterial infections, developed at Stanford University in '70
- 500 production rules for recognizing 100 causes of infection
- the most significant contribution was the introduction of a level of certainty of evidences and confidence in hypotheses.

XCON – rule-based system for configuring computers, developed at Carnegie-Mellon University in 1978
- 10000 rules to describe hundreds of types of components.
- contributed to a growing commercial interest in rule-based expert systems.

Frames – object-oriented representation

FOL and production systems are representation methods that are "flat", in the sense that each representation unit is independent of the others. Knowledge about an object could be scattered all over the knowledge base, among unrelated sentences. In big KBs, this can become critical, therefore it is important to organize knowledge representation somehow.

In 1975, Marvin Minsky introduced "frames" as knowledge representation for object-oriented groups of procedures to recognize and deal with new situations.

In representations using frames, facts and rules are grouped in terms of the kind of objects they belong to. Thus, knowledge is not seen as just a collection of sentences, but rather it is structured in terms of what knowledge is about (i.e. the objects of knowledge).

Generic and individual frames

Individual frames represent objects; generic frames represent categories or classes of objects.

Individual frames are similar to WMEs in production systems. An individual frame is a named list of slots into which values, called fillers, can be dropped.

```
( Frame_name
      < slot_name1   filler1 >
      < slot_name2   filler2 >
      ... )
```

The names of individual frames begin with a lower-case; the names of generic frames begin with upper-case.

Fillers are atomic values (numbers or strings) or names of other individual frames or generic frames.

Slot names begin with upper-case and are prefixed with :.

```
(tripLeg10
    <: INSTANCE-OF  TripLeg>
    <: Destination  toronto> ...)

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

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

Individual frames have a special slot called :INSTANCE-OF, whose filler is the name of a generic frame which indicates the category of the object represented by that individual frame (the individual frame is an instance of the generic frame).

Generic frames have a special slot called :IS-A, whose filler is the name of a more general generic frame. We say that the generic frame is a specialization of the more general one.

Slots of generic frames can have attached procedures

IF-ADDED and IF-NEEDED

The rule $Parent(x,y) \Leftarrow Mother(x,y)$ can be procedurally interpreted as following:

1. IF-NEEDED – whenever we have to solve a goal that matches $Parent(x,y)$, we reduce it to solving $Mother(x,y)$ (backward chaining). Procedurally, the connection between mothers and parents is made when we must prove something about parents.
2. IF-ADDED – whenever a fact that matches $Mother(x,y)$ is added to the KB, we also add $Parent(x,y)$ (forward chaining). The connection between mothers and parents is made when we know something new about $Mother(x,y)$.

(Trip
  <: TotalCost [IF-NEEDED computeCost] > ...)

(Lecture
  <: DayOfWeek WeekDay>
  <: Date [IF-ADDED ComputeDayofWeek] > ...)

A slot can have both a filler and an attached procedure in the same frame.

Inheritance

In a frame system, the reasoning involves creating individual instances of generic frames and filling/inferring values into slots. Generic frames can be used to fill in values that are not explicitly given at the creation of the instances. Generic frames can also trigger additional actions when fillers are provided. We can determine a value in an instance by inheritance (the child frames inherit properties from their parents frames).

For example, if we are interested in the filler for the slot :Country of the toronto frame, we can use :INSTANCE-OF that indicates to the generic frame CanadianCity, where the value is given (toronto inherits the :Country property from CanadianCity). If toronto had not a filler for :Province, we would still know by inheritance that we should look for an instance of CanadianProvince.

Inheritance of attached procedures

(ExpensiveTrip
  <: IS-A Trip> ...)

(ExoticExpensiveTrip
  <: IS-A ExpensiveTrip> ...)

if we create an instance of the frame Trip and we want to find a filler for :TotalCost in this instance, we use the attached procedure IF-NEEDED.

The same procedure can be used through inheritance if we create an instance of ExoticExpensiveTrip.

If we create an instance of the frame Lecture with the date specified explicitly:

```
(lecture1
    <:INSTANCE-OF Lecture>
    <:Date 20NOV> ...)
```

then the attached IF-ADDED procedure would be executed, filling the slot :DayOfWeek.

If we later change the :Date slot, the procedure will execute again, changing the filler for :DayOfWeek.

In a frame system, we use an inherited value only if we cannot find a filler otherwise (it is defeasible).

A slot filler in a generic frame can be overridden explicitly in its instances and its specializations:

```
(Elephant
    <:is-A Mammal>
    <:EarSize large>
    <:Color grey> ...)

(raja
    <:INSTANCE-OF Elephant>
    <:EarSize small> ...)

(RoyalElephant
    <:is-A Elephant>
    <:Color white> ...)

(clyde
    <:INSTANCE-OF RoyalElephant> ...)
```

A frame system allows for multiple inheritance. An individual/generic frame can be an instance/a specialization of more than one generic frame.

```
(AfricanElephant
    <:is-A Elephant>
    <:is-A AfricanAnimal> ...)
```

Reasoning with frames

In a frame system, reasoning starts when the system recognizes an object being an instance of a generic frame and applies the procedures triggered by that instance. These procedures can produce new data or changes in the KB. The system operation stops when no more procedures are applicable.

The system operates in a three-step loop:

1. Someone (a user or an external system) declares that an object exists, instantiating a generic frame.
2. Any slot fillers that are not provided explicitly, but can be inherited by the new instance, are inherited.
3. For each slot with a filler, if an IF-ADDED procedure can be inherited, then it is executed. By its execution, new slots can be filled or new frames can be instantiated; then goto 1.

If a filler is requested by a user, an external system or an attached procedure, then:

1. if the filler exists, then the value is returned;
2. Otherwise, any IF-NEEDED procedure that can be inherited is executed, computing the filler. The execution can also fill other slots or instantiate new frames.

if no result is produced by the steps above, then the value of the slot is unknown.

Obs. The inheritance of the values is done when the individual frame is created, but IF-NEEDED procedures are invoked only by request.

The constraints between slots are expressed by the attached IF-ADDED and IF-NEEDED procedures. The programmer decides whether reasoning is data-directed or goal-directed.

Knowledge representation and reasoning (KRR)

References

1. Ronald J. Brachman, Hector J. Levesque. Knowledge representation and reasoning, Morgan Kaufmann, 2004.
2. Stuart J. Russell, Peter Norvig. Artificial Intelligence – a modern approach, 3rd edition, Pearson, 2010.

Introduction

Intelligence $\frac{\text{Explan. dictionary}}{}$ the ability to understand easily and well, to understand what is essential, to solve new situations or problems from previous experiences.

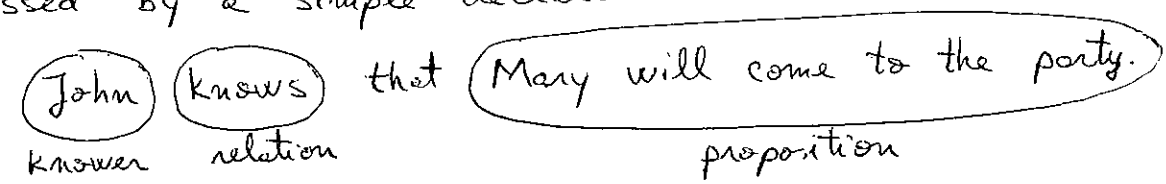
The intelligent behavior is clearly conditioned by knowledge.

Artificial intelligence (AI) studies the intelligent behavior acquired through computational means.

KRR is an area of AI concerned with the study of how an agent (e.g. human, hardware, software) uses what it knows when it decides what to do. It is the study of thinking as a computational process.

Knowledge

In an informal description, knowledge is a relation between a knower and a proposition. The proposition is the idea expressed by a simple declarative sentence.



Propositions are abstract entities that can be true or false, right or wrong.

There may be different types of relationships between an agent and a proposition:

John knows/hopes/doubts/regrets that Mary will come to the party.

However, what matters about the propositions is their truth value.

There are statements containing knowledge, which is not explicitly mentioned in propositions:

John knows how to get there.

In this case, we can imagine the implicit propositions:

John knows how to get to the party, he goes two blocks past the park, he turns left ...

On the other hand, in sentences like "John knows George well", it is not clear if any useful propositions are involved.

There are attitudes expressed by sentences like:

John is absolutely certain/confident/of the opinion that...

that differ only in the level of conviction attributed to a fact. The judgement may not be always exact.

Representation

It means a relationship between two domains, where the first takes the place of the second. The first one is more concrete, accessible than the second.

We shall consider the simbolic representation, that is a character or a group of characters from a predefined alphabet.

7, VII, seven - represent number 7

John - represents something concrete

righteousness, truth - represent abstractions

John loves Mary - symbolic representation of an abstract statement

Knowledge representation is the field that studies the use of formal symbols to represent a collection of propositions made by an agent.

Reasoning

It means the formal manipulation of the formal symbols that represent a collection of statements considered to be true, in order to produce representations of new symbols.

Therefore, the symbols must be "accessible" enough to be able to manipulate them (i.e. move, take apart, copy, concatenate), so that representations of new propositions may be constructed.

For instance, if we have the propositions

John loves Mary.
Mary comes to the party.

after a series of steps in reasoning, we can produce the sentence

Someone John loves comes to the party.

Reasoning is a form of calculation over symbols standing for propositions (rather than numbers in arithmetic) (Gottfried Leibniz).

Why is KRR relevant for AI?

- Because in some situations it is useful to describe the behavior of a complex system in terms like believes, desires, intends, hopes etc.
For example, if we play chess against the computer - in order to help us make the next move, it would be useful to understand the program's behaviour in terms of immediate goals pursued relative to its long-term intentions.

"It moved this way because it believed that the queen was vulnerable..."

- A knowledge base is a collection of symbolic structures that represent the knowledge but also the reasoning made during the system operation.

The behavior of a knowledge-based system is conditioned not only by the represented facts (that can be retrieved like in a database). Through reasoning, the "beliefs" of the system go beyond these.

A significant part of AI involves knowledge-based systems. Expert systems are a classical example, but applications of knowledge-based systems can be found in the areas of language understanding, diagnosis, planning.

Other AI systems are knowledge-based to a lesser extent – e.g. some games or high-level vision systems.

There are also AI systems that are not knowledge-based at all – for example low-level speech, vision. These systems encode the knowledge directly in the program.

Open question: How much of the intelligent behavior should be knowledge-based? The most serious challenge to the KRR is the connectionist approach, that computes weights between artificial neurons.

What are the advantages of a knowledge-based system?

Wouldn't it be more efficient to "compile" the knowledge-base (KB) and then distribute it to the procedures that need it? (this approach is known as procedural knowledge).

Why should we retrieve facts from the KB and make reasonings in the runtime to decide next actions?

Hubert Dreyfus observed a paradox of expert systems. These systems claim to be superior because are knowledge-based. But experts do not reason – they just see the solution. Novices are the ones who reason.

Dreyfus considers to be wrong the directions that attempts to copy the human intelligent behavior.

By design, a knowledge-based system has the ability to learn facts about the world and, consequently, to adjust its behavior.
The ability to make the behavior dependent on the knowledge is desirable when we cannot specify in advance how the knowledge will be used. This approach has some advantages:

- we can add new tasks and make them dependent on previous knowledge;
- we can extend the behavior by adding new beliefs;
- we can explain the behavior of the system.

The evaluation

Final grade = 50% Exam + 50% Lab

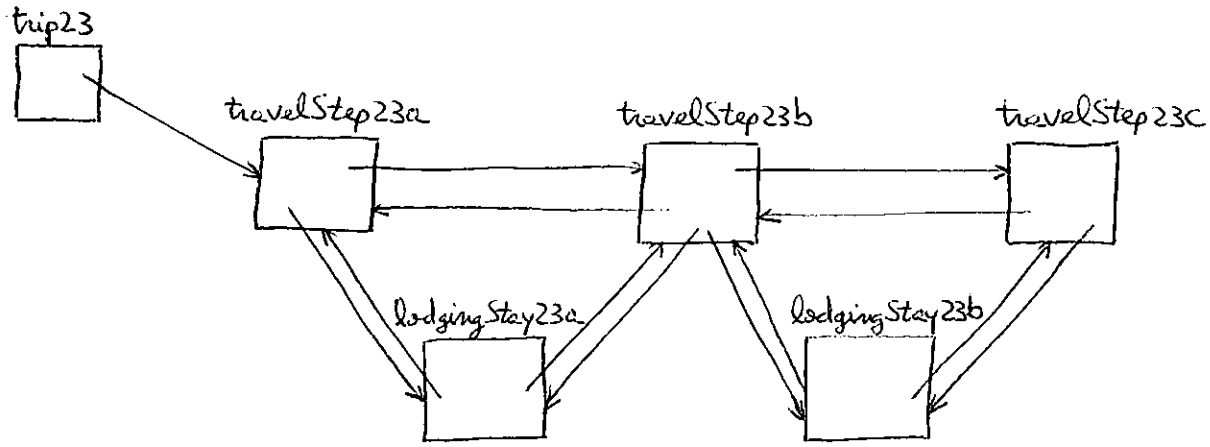
Exam ≥ 3

The exam
- in the session
- written
- 2 hours
- open book - no electronic devices

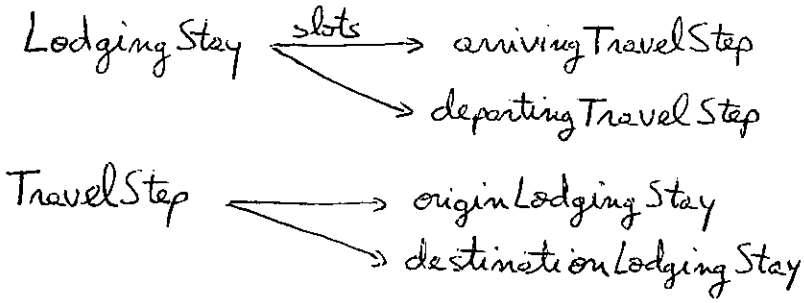
The lab
- programming language Prolog (for any other language, the grades are at most 8)
- 2 presentations
  - the first presentation during the 4th lab → grade G1
  - the second presentation during the 7th lab → grade G2
- the lab grade = $\frac{G1 + G2}{2}$

Frames – an example for planning a trip

We start with a general sketch of the basic structure of a trip.



There are two main frames: Trip and TravelStep. A Trip will have a sequence of TravelSteps. A TravelStep usually terminates with a LodgingStay, except when there are two travel steps in a single day or when we reach the last step of the trip.



The generic frame Trip and an instance of it may look as following:

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

```
(trip23
   <: INSTANCE-OF  Trip>
   <: FirstStep    travelStep23a>
   <: Traveler     davidF>
   <: BeginDate    23/11/19>
   <: EndDate      25/11/19>
   <: TotalCost    3400RON>
)
```

The frames TravelStep and LodgingStay share some properties that we group in the more general frame TripPart:

```
(TripPart
    <: BeginDate   Date>
    <: EndDate   Date>
    <: Cost   Price>
    <: PaymentMethod   FormOfPayment>
)
(LodgingStay
    <: IS-A   TripPart>
    <: Place   City>
    <: LodgingPlace   LodgingPlace>
    <: ArrivingTravelStep   TravelStep>
    <: DepartingTravelStep   TravelStep>
)
(TravelStep
    <: IS-A   TripPart>
    <: Origin   City>
    <: Destination   City>
    <: OriginLodgingStay   LodgingStay>
    <: DestinationLodgingStay   LodgingStay>
    <: Means   FormOfTransportation>
    <: DepartureTime   Time>
    <: ArrivalTime   Time>
    <: NextStep   TravelStep>
    <: PreviousStep   TravelStep>
)
```

Default fillers:

```
(TripPart
    <: PaymentMethod   visaCard>
...)
(TravelStep
    <: Means   airplane>
    <: PaymentMethod   masterCard>
...)
```

Notations: - if x is an individual frame and y is a slot, then xy refers to the filler of the slot in that frame;
- SELF is a reference to the current frame.

The filler of the slot : Origin from TravelStep can be calculated as following:

```
(TravelStep
   <: Origin [IF-NEEDED
                { if no SELF : PreviousStep then otopeni;
                     else SELF : PreviousStep : Destination;
                }
              ] > ...)

(Trip
  <: TotalCost [IF-NEEDED
                 { result ← 0
                   x ← SELF : FirstStep;
                  repeat
                   if exists x : NextStep then
                        { result ← result + x : Cost;
                          if exists x : DestinationLodgingStay then
                            { result ← result + x : DestinationLodgingStay : Cost;
                          x ← x : NextStep;
                        }
                   else return result + x : Cost;
                 }
               ] > ...)

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

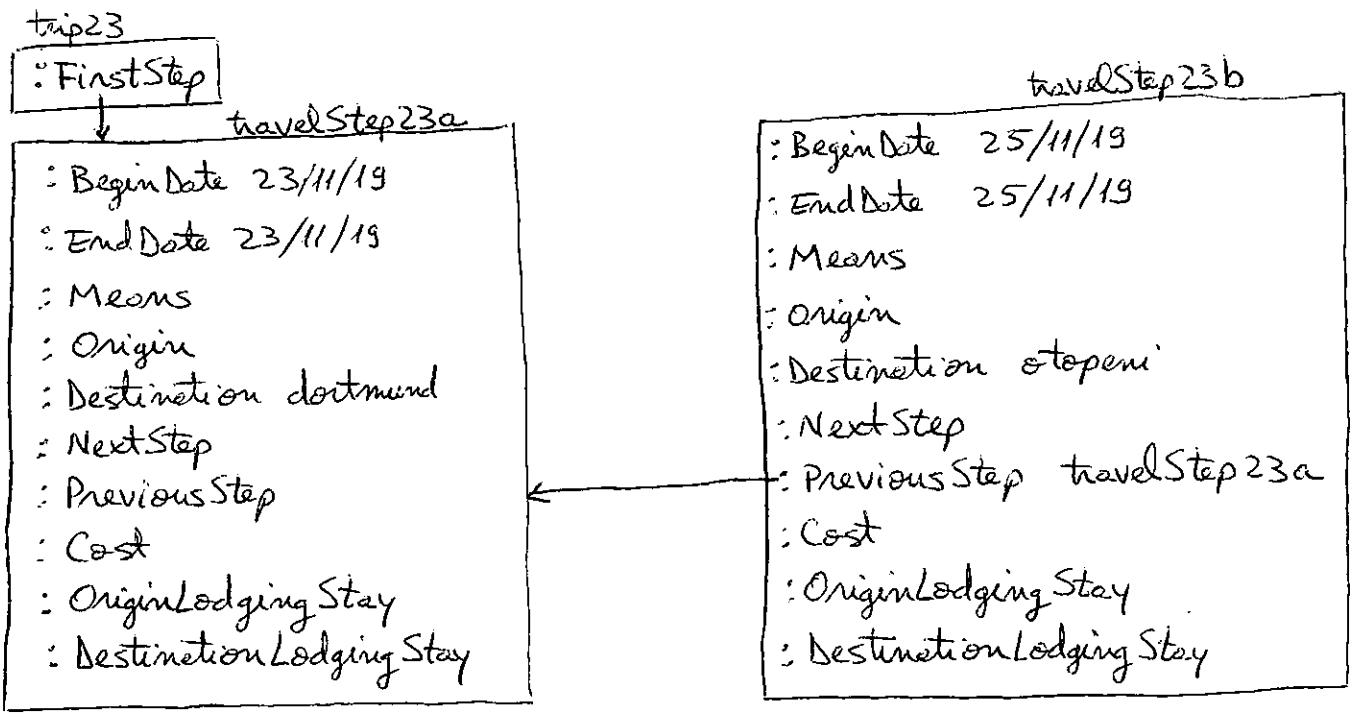
```
(LodgingStay
    <: Place [IF-NEEDED
                {SELF : ArrivingTravelStep : Destination}
              ] > ...)
```

For a certain trip, called trip23, we create an individual frame

```
(trip23
    <: INSTANCE-OF Trip>
    <: FirstStep travelStep23a>
)
```

and two instances of TravelStep

```
(travelStep23a
    <: INSTANCE-OF TravelStep>
    <: Destination dortmund>
    <: BeginDate 23/11/19>
    <: EndDate 23/11/19>
)
(travelStep23b
    <: INSTANCE-OF TravelStep>
    <: Destination otopeni>
    <: BeginDate 25/11/19>
    <: EndDate 25/11/19>
    <: PreviousStep travelStep23a>
)
```



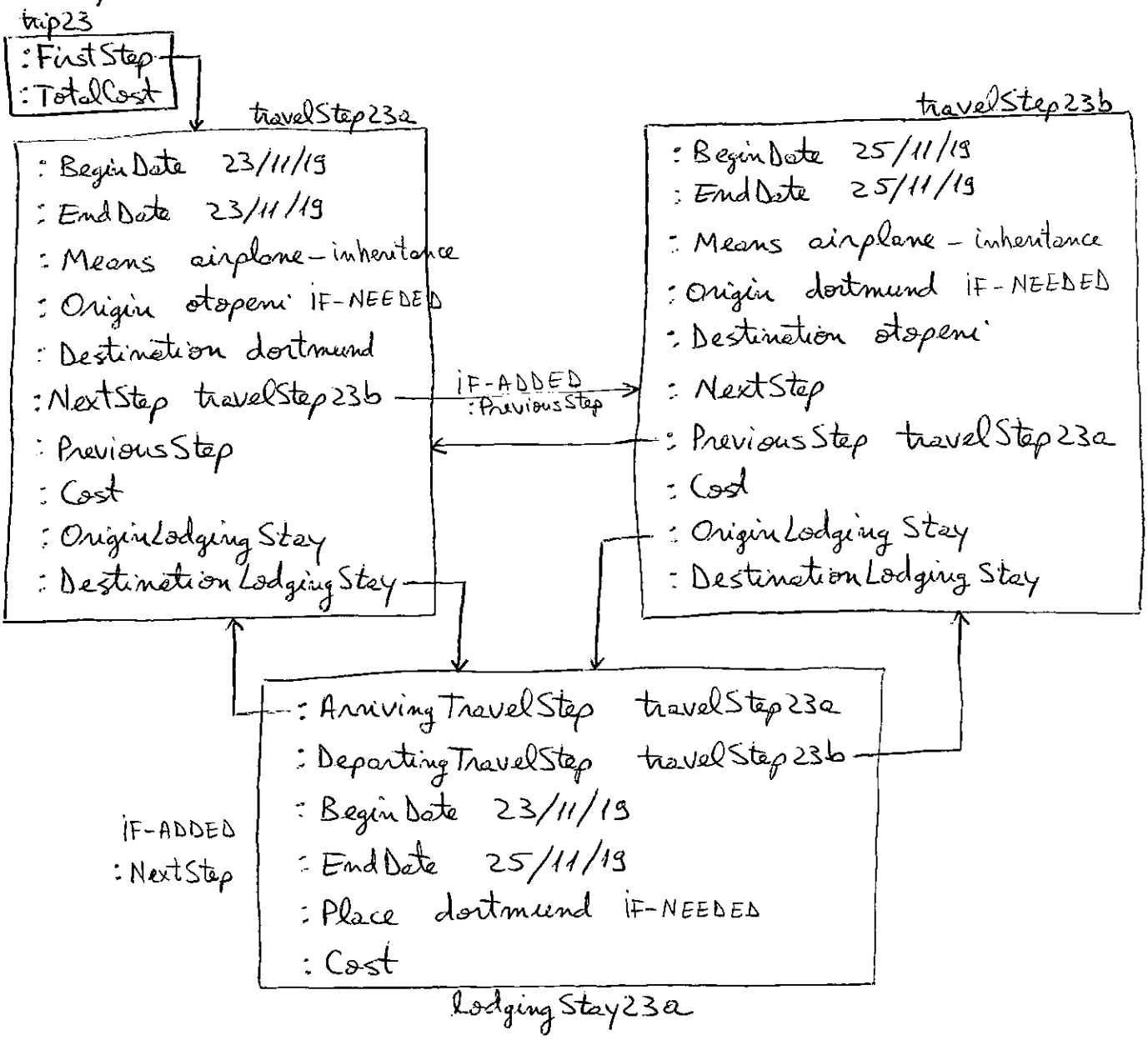
To complete the initial setup, we should have

travelStep23a : NextStep ← travelStep23b

```
(TravelStep
   <: PreviousStep [IF-ADDED
                      {SELF : PreviousStep : NextStep ← SELF ;}
                    ] > ...)
```

As a consequence of this assignment, the IF-ADDED attached procedure to the slot :NextStep is executed:

```
(lodgingStay23a
   <: INSTANCE-OF LodgingStay >
   <: BeginDate 23/11/19 >
   <: EndDate 25/11/19 >
   <: ArrivingTravelStep travelStep23a >
   <: DepartingTravelStep travelStep23b >
)
```



Structured descriptions

The syntax of FOL makes it easy to say things about objects. Frames organize knowledge in terms of categories of objects. Description logics are notations that are designed to make it easier to describe definitions and properties of categories, by adding structure to the definition of objects. The focus is on declarative aspects of object-oriented representation, going back to concepts like predicates and entailment from FOL.
Description logic systems evolved from frames/semantic networks by formalizing what the networks mean, while keeping the emphasis on taxonomic structure as an organizing principle (that helps in organizing a hierarchy of categories).
The principal inference tasks for description logics are subsumption (checking if a category is a subset of another by comparing their definitions) and satisfaction (checking whether an object belongs to a category).

In standard FOL systems, predicting the solution time is often impossible. In description logics, the subsumption testing can be solved in time polynomial in the size of the description. But (hard) problems either cannot be stated at all in description logics, or they require exponentially large descriptions.
In FOL, we represent categories of objects with simple predicates like Mother(x), Boat(x), Company(x). To represent more interesting types of constructions like "a man whose children are all girls", we need predicates with internal structure.
We would expect that if Child(x, y) and FatherOfOnlyGirls(x) were true, then y would have to be a girl (somehow) by definition.

We have category nouns like FatherOfOnlyGirls, Girl describing basic classes of objects and relational nouns like Child that are parts/attributes/properties of other objects.
In description logics, we refer to the first type as a concept and to the second type as a role (in frame systems we saw a similar distinction between frames/slots).
In contrast to the slots in frame systems, roles can have multiple fillers. Thus, it can be described naturally a person with several children, a salad made from more than one type of vegetable.
Although much of the reasoning in description logics concerns generic categories, constants are included to allow for descriptions to be applied to individuals.

A description language

In a description language (DL) there are two types of symbols:

- logical symbols, with a fixed meaning
- nonlogical symbols, which are application dependent

There are four types of logical symbols:

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

There are three types of nonlogical symbols:

- atomic concepts – the name starts with upper-case
  Person, FatherOfOnlyGirls
  Thing – a special atomic concept
- roles – the name starts with upper-case, prefixed by:
  :Age, :Child
- constants – the name starts with lower-case
  table17, johnSmith

There are four types of legal syntactic expressions:

- constants c
- roles r
- concepts d, e; atomic concepts a
- sentences

The set of concepts of DL satisfies the following:

- every atomic concept is a concept;
- if r is a role, d is a concept then [ALL r d] is a concept;
- if r is a role, $n \in \mathbb{N}^*$ then [EXISTS n r] is a concept;
- if r is a role, c is a constant then [FILLS r c] is a concept;
- if $d_1, \ldots, d_n$ are concepts then [AND $d_1 \ldots d_n$] is a concept.

There are three types of sentences in DL:

- if d_1, d_2 are concepts then $(d_1 \sqsubseteq d_2)$ is a sentence;
- if d_1, d_2 are concepts then $(d_1 \doteq d_2)$ is a sentence;
- if c is a constant and d concept then $(c \rightarrow d)$ is a sentence.

A knowledge base KB in DL is a collection of sentences.

Constants represent individuals in the application domain; concepts represent categories or classes of individuals; and roles represent binary relations between individuals.

The meaning of a complex concept derives from the meaning of its parts.

For example, [EXISTS n r] represent the class of individuals in the domain that are related by relation r to at least n other individuals. [EXISTS 1 : Child] represents someone who has at least one child.

if c is a constant that stands for some individual, the concept [FILLS r c] represents those individuals that are in relation r with c. [FILLS :Cousin george] represent someone whose cousin is George.

if concept d represents a class of individuals, [ALL r d] represent individuals who are in relation r only to individuals of class d. [ALL :Employee UnionMember] describes companies whose employees are all union members.

The concept [AND $d_1 \ldots d_n$] represents anything described by d_1 and ... d_n.

```
[AND  Wine
      [FILLS  :Color red]
      [EXISTS 2 :GrapeType]
]
```

```
(ProgressiveCompany ≐ [AND Company
                           [EXISTS 7 : Director]
                           [ALL : Manager [AND Women
                                               [FILLS : Degree phd]]]
                           [FILLS : MinSalary $5000]
                      ])
```

In DL, sentences are true or false in the domain (like in FOL).

d_1, d_2 concepts and c constant

$(d_1 \sqsubseteq d_2)$ says that d_1 is subsumed by d_2, that is all individuals that satisfy d_1 also satisfy d_2.

(Surgeon $\sqsubseteq$ Doctor)

$(d_1 \doteq d_2)$ says that d_1 and d_2 are equivalent, that is the individuals that satisfy d_1 are exactly those that satisfy d_2. It is the same as saying that both $(d_1 \sqsubseteq d_2)$ and $(d_2 \sqsubseteq d_1)$ are true.

$(c \longrightarrow d)$ says that the individual denoted by c satisfies the description expressed by d.

Interpretations in DL

An interpretation $\mathcal{I}$ is a pair $\langle \Delta, I \rangle$, where Δ is a non-empty set of objects called the domain of the interpretation and I is the interpretation mapping that assigns a meaning to the nonlogical symbols of DL, so that:

1. for every constant c, $I[c] \in \Delta$;
2. for every atomic concept a, $I[a] \subseteq \Delta$;
3. for every role r, $I[r] \subseteq \Delta \times \Delta$.

The set $I[d]$ is called the extension of the concept d:

- $I[\text{Thing}] = \Delta$;
- $I[[\text{ALL } r\ d]] = \{x \in \Delta \mid \forall y \text{ if } \langle x, y \rangle \in I[r] \text{ then } y \in I[d]\}$;
- $I[[\text{EXISTS } n\ r]] = \{x \in \Delta \mid$ there are at least n distinct y such that $\langle x, y \rangle \in I[r]\}$;
- $I[[\text{FILLS } r\ c]] = \{x \in \Delta \mid \langle x, I[c] \rangle \in I[r]\}$;
- $I[[\text{AND } d_1 \ldots d_n]] = I[d_1] \cap \ldots \cap I[d_n]$.

Truth in an interpretation

The sentence $(c \rightarrow d)$ is true in $\mathcal{I}$ if the object denoted by c is in the extension of d – $I[c] \in I[d]$.

The sentence $(d \sqsubseteq d')$ is true if the extension of d is a subset of the extension of d' – $I[d] \subseteq I[d']$.

The sentence $(d \doteq d')$ is true if $I[d] = I[d']$

If a sentence α is true in $\mathcal{I}$, we write $\mathcal{I} \models \alpha$.
If S is a set of sentences, we will write $\mathcal{I} \models S$ to say that all the sentences in S are true in $\mathcal{I}$.

Entailment

Let S be a set of sentences in DL and α a sentence.

S logically entails α, and we write $S \models \alpha$, iff for every interpretation $\mathcal{I}$, if $\mathcal{I} \models S$ then $\mathcal{I} \models \alpha$.

A sentence α is logically valid, and we write $\models \alpha$, if it is logically entailed by the empty set.

In DL, there are two basic types of reasoning: determining whether or not a constant c satisfies a concept d; and determining whether or not a concept d is subsumed by another concept d'.

$$KB \models (c \rightarrow d)$$
$$KB \models (d \sqsubseteq d')$$

Examples of valid sentences:

$$([\text{AND Doctor Female}] \sqsubseteq \text{Doctor})$$
$$(\text{john} \rightarrow \text{Thing}).$$

In more typical cases, the entailment depends on sentences in the KB. For example, if KB contains the sentence $(\text{Surgeon} \sqsubseteq \text{Doctor})$, then we can logically entail that

$$KB \models ([\text{AND Surgeon Female}] \sqsubseteq \text{Doctor}).$$

We can reach the same conclusion if we have in KB the sentence $(\text{Surgeon} \doteq [\text{AND Doctor [FILLS :Specialty surgery]}])$ instead of $(\text{Surgeon} \sqsubseteq \text{Doctor})$.

But with the empty KB, we would have no subsumption relation $([\text{AND Surgeon Female}] \sqsubseteq \text{Doctor})$ because we can choose an interpretation $\mathcal{I}$ in which the sentence is false (for example, $\mathcal{I}[\text{Doctor}] = \emptyset$ and $\mathcal{I}[\text{Surgeon}] = \mathcal{I}[\text{Female}] = \{1,2,3\}$).

Computing entailments

Given a KB, we want to determine if $KB \models \alpha$ for α of the form:

$(c \rightarrow d)$ where c constant and d concept

$(d \sqsubseteq e)$ where d, e concepts

$[KB \models (d \doteq e)$ iff $KB \models (d \sqsubseteq e)$ and $KB \models (e \sqsubseteq d)]$.

Simplifying the KB

Obs. It can be proven that subsumption entailments are not affected by the presence of sentences $(c \rightarrow d)$ in KB. That is to say that $KB \models (d \sqsubseteq e)$ iff $KB' \models (d \sqsubseteq e)$, $KB' = KB - \{$all sentences $(c \rightarrow d)\}$

For subsumption questions, we assume that the KB contains no $(c \rightarrow d)$ sentences.

Moreover, we can replace sentences of the form $(d \sqsubseteq e)$ by $(d \doteq [AND\ e\ a])$, where a is a new atomic concept used nowhere else.

We will consider the following restrictions in the KB:

- the left-hand sides of $\doteq$ is an atomic concept other than Thing
- each atom appears on the left-hand side of $\doteq$ exactly once in KB – such sentences provide definitions of the atomic concepts

  (RedBordeauxWine $\doteq$ [AND Wine
  [FILLS :Color red]
  [FILLS :Region bordeaux]])

- we assume that sentences $\doteq$ in KB are acyclic. We rule out a KB that contains

  $(d_1 \doteq [AND\ d_2 \ldots])$, $(d_2 \doteq [ALL\ r\ d_3])$, $(d_3 = [AND\ d_1 \ldots])$,

Under these restrictions, to determine if $KB \models (d \sqsubseteq e)$ we do the following:

1. put d and e into a special normalized form
2. determine whether each part of the normalized e is accounted for by some part of the normalized d. We are looking for a structural relation between two normalized concepts. For example, if e contains [ALL r e'] then d must contain [ALL r d'] with $d' \sqsubseteq e'$.

Normalization

It is a preprocessing that simplifies the structure-matching between concepts. It applies to one concept at a time and involves the following steps:

1. expand definitions — any atomic concept in the left-hand side of $\doteq$ is replaced by its definition.

Example: assume that we have the following sentence in KB

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

The concept [AND ... Surgeon ...] expands to

[AND ... [AND Doctor [FILLS : Specialty surgery]] ...]

2. flatten the AND operators

[AND ... [AND $d_1 \ldots d_n$] ...] becomes [AND ... $d_1 \ldots d_n$ ...]

3. combine the ALL operators

[AND ... [ALL r d_1] ... [ALL r d_2] ...] becomes

[AND ... [ALL r [AND d_1 d_2]] ...]

4. combine the EXISTS operators

[AND ... [EXISTS n_1 r] ... [EXISTS n_2 r] ...] becomes

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

5. Thing concept – remove Thing, [ALL r Thing] and AND with no arguments if they appear as arguments in an AND concept

[AND ... Thing ...] becomes [AND ...]

[AND Company [ALL :Employee Thing]] becomes Company.

6. remove redundant expressions – eliminate duplicates within the same AND expression.

These six steps are applied repeatedly until no steps are applicable. The result is either Thing, an atomic concept or a concept of the following form:

[AND $a_1 \ldots a_m$
[FILLS r_1 c_1] ... [FILLS $r_{m'}$ $c_{m'}$]
[EXISTS n_1 s_1] ... [EXISTS $n_{m''}$ $s_{m''}$]
[ALL t_1 e_1] ... [ALL $t_{m'''}$ $e_{m'''}$]]

where $a_1, \ldots, a_m$ are atomic concepts (other than Thing), r_i, s_i, t_i are roles, c_i are constants, n_i are positive integers and e_i are normalized concepts.

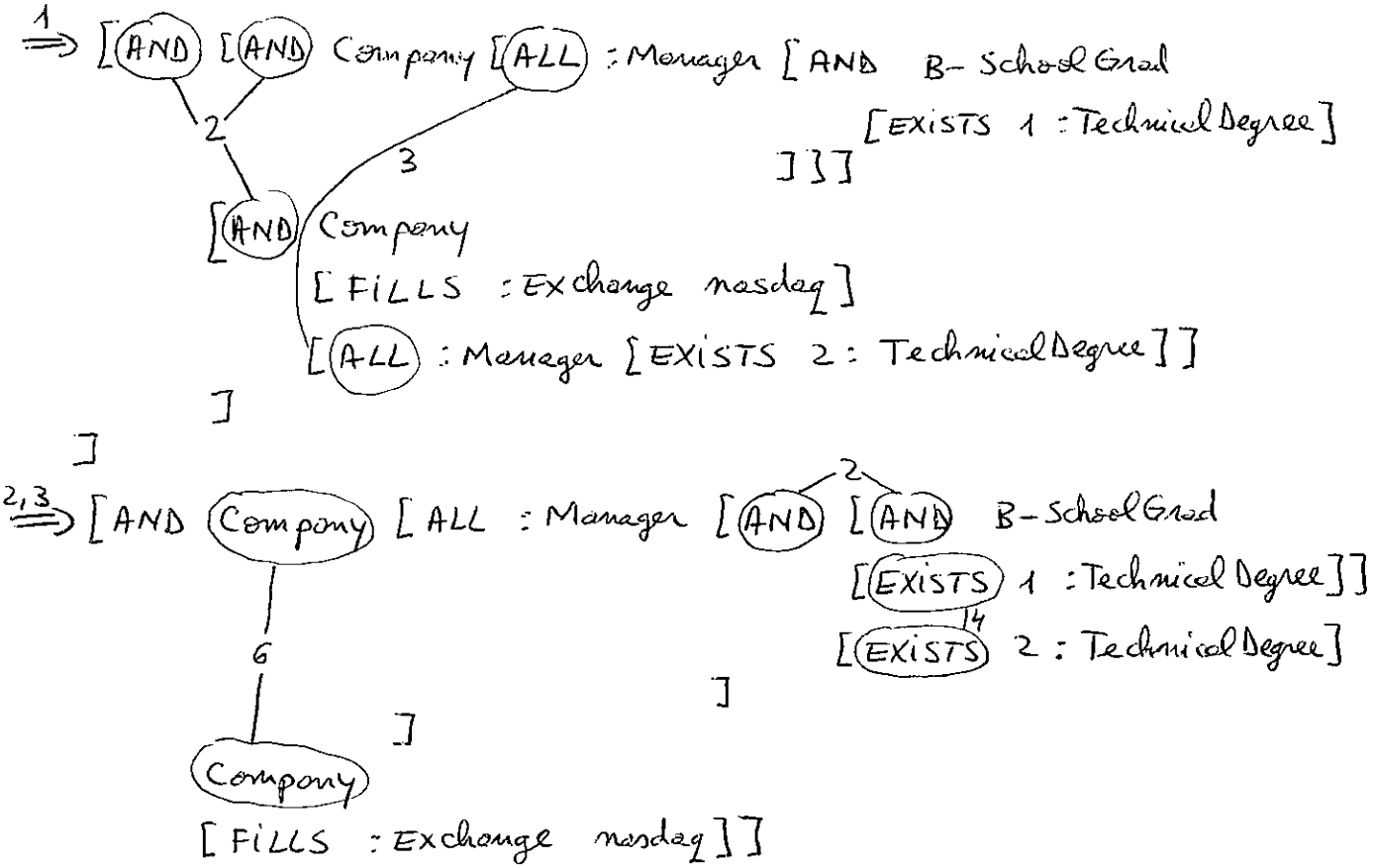
Example 1 - We have the following KB:

WellRoundedCo $\doteq$ [AND Company
[ALL :Manager [AND B-SchoolGrad
[EXISTS 1 :TechnicalDegree]
]]]

HighTechCo $\doteq$ [AND Company
[FILLS :Exchange nasdaq]
[ALL :Manager Techie]]

Techie $\doteq$ [EXISTS 2 :TechnicalDegree]

Normalize the concept [AND WellRoundedCo HighTechCo]



Structure matching procedure – Subsumption computation

Input: d and e two normalized concepts

d is [AND d_1 ... d_m]

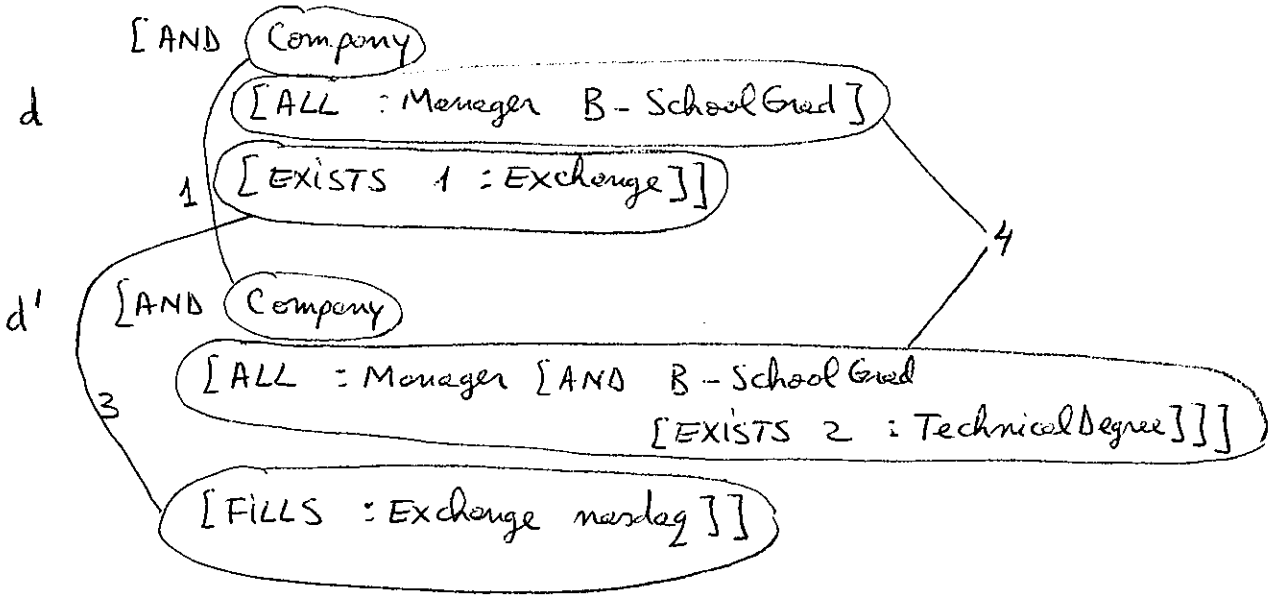
e is [AND e_1 ... $e_{m'}$]

Output: YES or NO according to whether or not $KB \models (d \sqsubseteq e)$

Return YES iff for each e_j, $j \in \overline{1,m'}$, there exists a component d_i, $i \in \overline{1,m}$ such that d_i matches e_j as follows:

1. if e_j is an atomic concept, then d_i must be identical to e_j;
2. if e_j is of the form [FILLS r c], then d_i must be identical to it;
3. if e_j is of the form [EXISTS n r], then d_i must be of the form [EXISTS n' r] for some $n' \geq n$; in the case where $n=1$, d_i can also be of the form [FILLS r c], for any constant c;
4. if e_j is of the form [ALL r e'], then d_i must be of the form [ALL r d'], where recursively $d' \sqsubseteq e'$.

Example 2



So $d' \sqsubseteq d$.

Computing satisfaction

We are interested whether $KB \models (b \rightarrow e)$, where b is a constant and e is a concept.

To find out if an individual satisfies a description, we need to propagate the information implied by what we know about other individuals before checking for subsumption. This can be done by a forward chaining procedure.

In the case where there are no EXISTS terms in any concept, the procedure is as following:

1. Construct S a list of pairs (b, d), where b is any constant mentioned in KB and d is the normalized version of the concept [AND $d_1' \ldots d_n'$] for all d_i' such that $(b \rightarrow d_i') \in KB$.
2. Find two constants b_1 and b_2 such that $(b_1, d_1) \in S$ and $(b_2, d_2) \in S$, [FILLS r b_2] and [ALL r e] are both components of d_1 but $KB \not\models (d_2 \sqsubseteq e)$.
3. If no b_1 and b_2 can be found, then exit. Otherwise, replace the pair (b_2, d_2) in S by (b_2, d_2'), where d_2' is the normalized version of [AND d_2 e] and go-to step 2.

The procedure computes for each constant b the most specific concept d such that $KB \models (b \to d)$. Now, to test whether or not $KB \models (b \to e)$, we need only to test whether or not $KB \models (d \sqsubseteq e)$.

Example 3

KB:
- joe → Person
- conCorp → [AND Company [ALL r Manager] [ALL e Canadian] [FILLS r Manager] [joe]] D

Question $KB \models (joe \to Canadian)$

$S = \{ (joe, Person), (conCorp, D) \}$ $\quad KB \not\models (Person \sqsubseteq Canadian)$

($joe = b_2$, $Person = d_2$, $conCorp = b_1$, $D = d_1$)

$\Rightarrow S = \{ (joe, [AND\ Person\ Canadian]), (conCorp, D) \}$. Now the procedure terminates because $KB \models ([AND\ Person\ Canadian] \sqsubseteq Canadian)$.

Because $KB \models ([AND\ Person\ Canadian] \sqsubseteq Canadian)$, it follows that $KB \models (joe \to Canadian)$.

In the case where there are EXISTS terms of the form [EXISTS 1 r], we will use role chains

[AND ... [ALL r_1 ... [AND ... [ALL r_k a] ...] ...] ...]

$\tau = r_1 \cdot r_2 \cdots r_k$ is called a role chain.

if b is a constant and r_1, r_2 roles, then $b \cdot r_1 \cdot r_2$ represents an individual (perhaps unnamed) that is in relation r_2 with an individual that is in relation r_1 with b.

if τ is empty, then $b \cdot \tau$ is b.

The forward chaining procedure extends, by adding two additional steps:

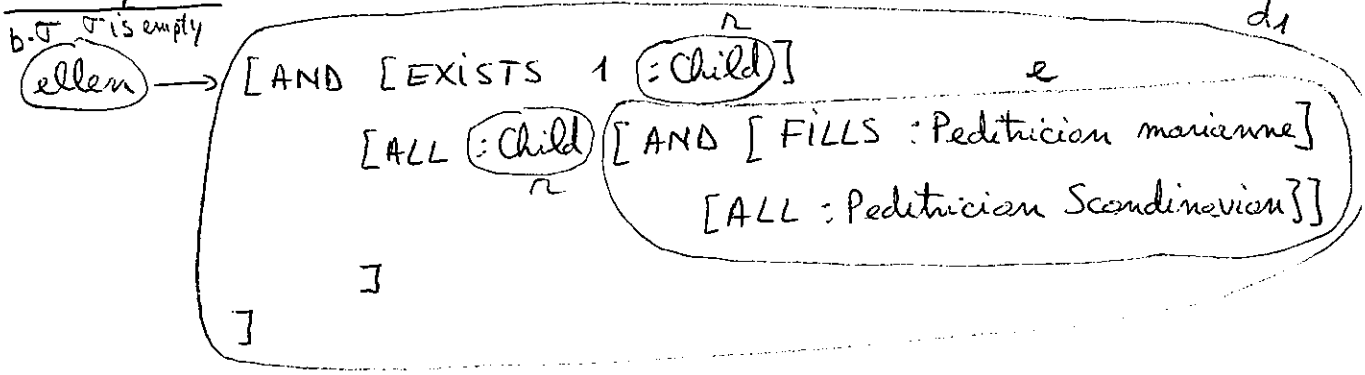
(the previous steps at page 11)

4. Find a constant b, a role chain τ (possibly empty) and a role r such that $(b\text{-}\tau, d_1) \in S$ and $(b\text{-}\tau\text{-}r, d_2) \in S$ (if no such pair exists, take d_2 to be Thing), where [EXISTS 1 r] and [ALL r e] are components of d_1, but $KB \not\models (d_2 \sqsubseteq e)$.

5. If these can be found, remove $(b\text{-}\tau\text{-}r, d_2)$ from S (if applicable) and add the pair $(b\text{-}\tau\text{-}r, d_2')$, where d_2' is the normalized version of [AND d_2 e]. Repeat.

We start with a property of the individual $b\text{-}\tau$ and conclude something new about the (unnamed) individual $b\text{-}\tau\text{-}r$. Eventually, this can lead to new information about a named individual.

Example 4 Assume that in KB we have the sentence



$S = \{(b\text{-}\tau, d_1)\}$ and $(b\text{-}\tau\text{-}r, d_2) = (ellen : Child, Thing)$

Because $KB \not\models (Thing \sqsubseteq e)$, S becomes

$S = \{(b\text{-}\tau, d_1), (\underbrace{ellen : Child}_{b\text{-}\tau\text{-}r}, [AND\ [FILLS : Pediatrician\ marianne]\ [ALL : Pediatrician\ Scandinavian]])\}$

From here, we conclude that (marianne → Scandinavian) (case with no EXISTS)

The case of terms of the form [EXISTS n r], $n > 1$ is handled the same as for $n = 1$. There is no need to create n different anonymous individuals because all of them would "produce" the same properties in the forward chaining.

Taxonomies and classification

Given a concept q, in DL it is common to ask for all of its instances, that is to find all c in KB so that $KB \models (c \rightarrow q)$.

Also, it is common to ask for all of the known categories that an individual satisfies. That is to say that given a constant c, we should find all concept a so that $KB \models (c \rightarrow a)$.

When reasoning in DL, we should exploit the hierarchical organization of the concepts, with the most general ones at the top and the more specialized ones further down.

To represent sentences in KB, we use a taxonomy (a treelike data structure) that allows answering queries efficiently (time linear with the depth of the taxonomy, not with its size).

Obs. Subsumption is a partial order.

The taxonomy have atomic concepts as nodes and edges like $\uparrow \begin{matrix} a_j \\ a_i \end{matrix}$ whenever $a_i \sqsubseteq a_j$ and there is no a_k such that $a_i \sqsubseteq a_k \sqsubseteq a_j$.

Each constant c in KB will be linked to the most specific atomic concept a_i such that $KB \models (c \rightarrow a_i)$.

Adding some new atomic concept or constant to a taxonomy corresponding to a KB is called classification. It involves creating a link from the new concept or constant to existing ones in the taxonomy.

This process exploits the structure of the taxonomy.

We start with the concept Thing and then add incrementally new atomic concepts and constants.

Computing classification

I add a sentence $(a_{new} \doteq d)$ to the taxonomy, where a_{new} is an atomic concept not appearing anywhere in the KB and d is any concept:

1. Compute S, the most specific subsumers of d

   $S = \{ a$ - concept in the taxonomy $\mid KB \models (d \sqsubseteq a)$, but $\nexists\, a' \neq a$ so that $KB \models (d \sqsubseteq a')$ and $KB \models (a' \sqsubseteq a) \}$

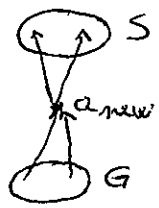
2. Compute G, the most general subsumees of d

   $G = \{ a$ - concept in the taxonomy $\mid KB \models (a \sqsubseteq d)$, but $\nexists\, a' \neq a$ so that $KB \models (a' \sqsubseteq d)$ and $KB \models (a \sqsubseteq a') \}$

3. if $\exists\, a \in S \cap G$ then a_{new} is already in the taxonomy under a different name — no action needed

4. Otherwise remove all links (if any) from concepts in G up to concepts in S.

5. Add links from a_{new} up to each concept in S and links from each concept in G up to a_{new}



6. Handling constants

   Compute $C = \{ c$ - constant in taxonomy $\mid \forall\, a \in S, KB \models (c \to a)$ and $\nexists\, a' \in G$ such that $KB \models (c \to a') \}$

   Then for each $c \in C$ we test if $KB \models (c \to d)$ and if so, we remove the links from c to S and add a single link from c to a_{new}.

II add a sentence $(a_{new} \sqsubseteq d)$ reduces to adding links from a_{new} to the most specific subsumers of d.

III add a sentence $(c_{new} \to d)$ reduces to adding links from c_{new} to the most specific subsumers of d.

Compute S - the most specific subsumers of d

start with $S = \{Thing\}$

for all $a \in S$ if $\exists a'$ so that $\uparrow_{a'}^{a}$ and $KB \models (d \sqsubseteq a')$ then

remove a from S and add all a' in S.

Repeat until no element in S has a child that subsumes d.

Compute G - the most general subsumees of d

start with $G = S$

if $\exists a \in G$ so that $KB \not\models (a \sqsubseteq d)$, then replace a with all its children (or delete it if it has no children).

Repeat until each element in G is subsumed by d.

Finally, we delete each $a \in G$ that has a parent subsumed by d.

Answering questions in DL

To find all constants c that satisfy a concept q, we should classify q and then collect all the constants at the fringe of the tree below q in the taxonomy.

To find all atomic concepts that are satisfied by a constant c, we go from c up in the taxonomy, collecting all the nodes that can be reached.

KB

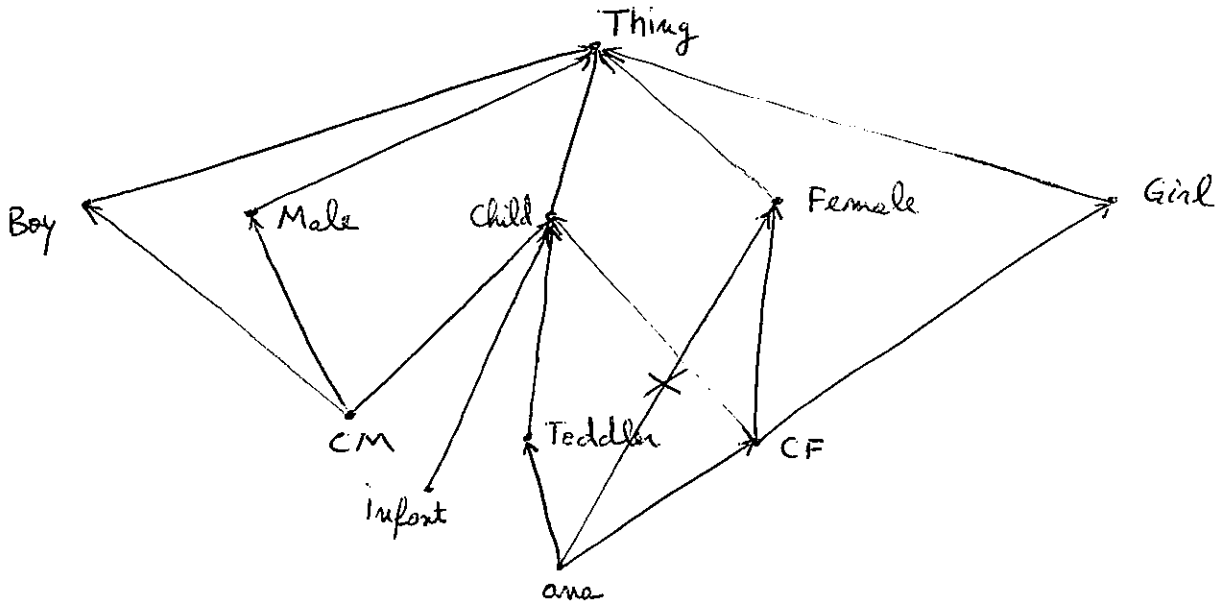
- Toddler
- Toddler ⊃ Child
- Child ∧ Male ⊃ Boy
- Infant ⊃ Child
- Child ∧ Female ⊃ Girl
- Female

Question: Girl

KB in DL

- $(Toddler \sqsubseteq Child)$
- $(CM \doteq [AND\ Child\ Male])$
- $(Infant \sqsubseteq Child)$
- $(CF \doteq [AND\ Child\ Female])$
- $(CM \sqsubseteq Boy)$
- $(CF \sqsubseteq Girl)$
- $(ana \rightarrow Toddler)$
- $(ana \rightarrow Female)$

Question: $(ana \rightarrow Girl)$



Defaults

Frames offer a simple form of default reasoning, where a slot has a certain value by inheritance unless another one is explicitly given.

Assuming that we have Dog(fido) in a KB in FOL, there are only two ways to reach the conclusion Carnivore(fido):

1. this is explicitly mentioned in KB;
2. in KB we have the universal form $\forall x.\ Dog(x) \supset Carnivore(x)$.

We are interested in expressing in FOL what we know about something in general and in particular using universals. For example, we may say that "Bikes have two wheels" but without stating that "All bikes have two wheels" because this would rule out a bike with three wheels. A possible solution would be to say

"All bikes that are not P_1 or ... P_n have two wheels",

where P_i represent the exceptional cases. The challenge here is to characterize these cases.

We would like to make a distinction between <u>universals</u> that hold for all instances and <u>generics</u> that hold in general. Much of our knowledge about the world is generic, therefore it is important to formalize it.

Default reasoning

In general, we know that dogs are carnivores. If Fido is a dog, what are the appropriate circumstances to infer that Fido is a carnivore? We can reason as following:

> Given that a P is generally a Q and knowing that P(a) is true, it is reasonable to conclude that Q(a) is true unless there is a good reason not to.

If all that we know about an individual is that it is an instance of P, then there is no reason not to conclude that it is an instance of Q.

For example, if we know that a polar bear has been playing in the mud, probably we do not want to conclude anything about its color. But if all we know is that it is a polar bear, then it is reasonable to conclude that it is white. The conclusion has no guarantee to be logically corect, it is only a reasonable default.

This form of reasoning that involves general (not universal) knowledge about a particular individual, is called <u>default reasoning</u>.

Examples of situations when we want to conclude Q(a) given P(a):

- General statements: Children love playing.
  Oranges are orange.
  People in a long queue become impatient.
- Lack of information to the contrary:
  No country has a president taller than 2 m.
  Children learn easily foreign languages.
- Conventions: The speed limit in a city.
  The closest shop is five minute walk (by default assume that it is open).
- Persistence: Marital status.
  The size of objects.

The list of examples is not exhaustive, but it suggests the great variety of sources of default information. Our focus is to describe exactly when it is appropriate to draw a default conclusion, in the absence of universals.

Nonmonotonicity

Ordinary deductive reasoning is monotonic, meaning that new facts produce only additional beliefs. If $KB_1 \models \alpha$ then $KB_2 \models \alpha$ for any KB_2 such that $KB_1 \subseteq KB_2$.

Default reasoning is nonmonotonic, meaning that sometimes new facts invalidate previous beliefs. For example, we believe by default that a bird flies, but if we find that the bird is an ostrich, we reconsider our belief.

I Closed-world reasoning

It is the simplest formalization of default reasoning. A finite vocabulary of predicate and constant symbols is used to represent facts about the world. But from all the valid atomic sentences, only a small fraction of them are expected to be true. The convention here is to explicitly represent the true atomic sentences and to assume that any unmentioned one is false.

Example

$$KB \begin{cases} \text{DirectConnect}(\text{cleveland}, \text{toronto}) \\ \text{DirectConnect}(\text{toronto}, \text{chicago}) \\ \text{DirectConnect}(\text{cleveland}, \text{vancouver}) \end{cases}$$

if a flight between two cities is not listed, then there is none. The closed-world assumption (CWA) is the following:

> Unless an atomic sentence is known to be true, it can be assumed to be false.

Obs. A sentence assumed to be false can be later determined to be true.

Def $KB^+ = KB \cup \{\neg p \mid p \text{ is atomic and } KB \not\models p\}$. A new form of entailment is defined as following:

$$KB \models_c \alpha \quad \text{iff} \quad KB^+ \models \alpha.$$

In the previous example, KB^+ would include sentences of the form $\neg DirectConnect(c_1, c_2)$.

Consistency and completeness of Knowledge

Def. A KB exhibits *consistent knowledge* iff there is no sentence α such that both α and $\neg\alpha$ are known.

Def. A KB exhibits *complete knowledge* iff for every sentence, either α or $\neg\alpha$ is known.

Knowledge can be incomplete. For example, if $KB = \{(p \vee q)\}$, neither p nor $\neg p$ can be entailed from KB.
But with the CWA, the entailment relation is complete.
For any sentence α, it holds that either $KB \models_c \alpha$ or $KB \models_c \neg\alpha$ (demonstration is by induction on the length of α).
Under CWA, whenever $KB \not\models p$ then either $KB \models \neg p$ directly or $\neg p$ is conceptually added to the KB. That means that we act as if KB represents complete knowledge.

Query evaluation

The question $KB \models_c \alpha$ reduces to questions about the literals in α:

1. $KB \models (\alpha \wedge \beta)$ iff $KB \models \alpha$ and $KB \models \beta$.
2. $KB \models \neg\neg\alpha$ iff $KB \models \alpha$.
3. $KB \models \neg(\alpha \vee \beta)$ iff $KB \models \neg\alpha$ and $KB \models \neg\beta$.
4. $KB \models_c (\alpha \vee \beta)$ iff $KB \models_c \alpha$ or $KB \models_c \beta$.
5. $KB \models_c \neg(\alpha \wedge \beta)$ iff $KB \models_c \neg\alpha$ or $KB \models_c \neg\beta$.
6. if KB^+ is consistent, then $KB \models_c \neg\alpha$ iff $KB \not\models_c \alpha$.

For example, $KB \models_C ((p \wedge q) \vee \neg(r \wedge \neg s))$ reduces to either both $KB \models_C p$ and $KB \models_C q$, or $KB \models_C \neg r$, or $KB \models_C s$.

Consistency and Generalized Closed-World Assumption (GCWA)

A consistent KB does not imply that KB^+ is also consistent. For example, if $KB = \{(p \vee q)\}$ then KB^+ contains $\{(p \vee q), \neg p, \neg q\}$ because $KB \not\models p$ and $KB \not\models q$. So, KB^+ is not consistent.

Obs. If a KB consists of just atomic sentences (e.g. DirectConnect) or conjunctions of atomic sentences (e.g. $p \wedge q$) or disjunctions of negative literals (e.g. $\neg p \vee \neg q$), then KB^+ is consistent.

One way to preserve consistency is to restrict the application of CWA only to atoms that are "uncontroversial" (not like p and q in the example above).

Def. The generalized closed-world assumption is

$KB^* = KB \cup \{\neg p \mid$ for all collections of atoms $q_1, \ldots, q_n$, if $KB \models (p \vee q_1 \vee \ldots \vee q_n)$ then $KB \models (q_1 \vee \ldots \vee q_n)\}$.

The entailment in GCWA is defined as following:

$KB \models_{GC} \alpha$ iff $KB^* \models \alpha$.

Under GCWA, we will not assume that p is false if there is an entailed disjunction of atoms including p that cannot be reduced to a smaller entailed disjunction not involving p.

For example, if $KB = \{(p \vee q)\}$ then $KB \models (p \vee q)$ but $KB \not\models q$, so $\neg p \notin KB^*$ and similarly $\neg q \notin KB^*$.

If we consider an atom r, then $\neg r \in KB^*$ because $KB \models (r \vee p \vee q)$ and $KB \models (p \vee q)$.

An example of interpretation: we know that there is a direct flight from Cleveland to Dallas or Houston. As a result, we know that there is a direct flight from Cleveland to Dallas, Houston or Austin. But because there is a flight to one of the first two cities, under GCWA we will assume that there is no flight to Austin.

$KB \models (DirectConnect(cleveland, dallas) \vee DirectConnect(cleveland, houston))$

$\Rightarrow KB \models (DirectConnect(cleveland, dallas) \vee DirectConnect(cleveland, houston) \vee DirectConnect(cleveland, austin))$

$\Rightarrow \neg DirectConnect(cleveland, austin) \in KB^*$.

Entailments in GCWA are a subset of those in CWA, that is if $\neg p \in KB^*$ then $\neg p \in KB^+$.

If KB has no disjunctive knowledge, then GCWA and CWA are in complete agreement.

if KB is consistent, then KB^* is consistent.

GCWA is a weaker version of CWA that agrees with CWA in the absence of disjunctions, but remains consistent in the presence of disjunctions.

Quantifiers and Domain Closure

Let us assume that the representation language contains the predicate DirectConnect and the constants $c_1, \ldots c_n$.

If KB contains only atomic sentences of the form $DirectConnect(c_i, c_j)$, then for any pair of constants c_i and c_j either $DirectConnect(c_i, c_j)$ or $\neg DirectConnect(c_i, c_j)$ is in KB^+.

Assuming that there is a city SmallTown that has no airport, then for every c_j, $\neg DirectConnect(c_j, smallTown)$ is in KB^+.

If we consider the query $\neg \exists x$ DirectConnect$(x, \text{smallTown})$, under CWA neither this query nor its negation is entailed. CWA excludes any of the named cities $c_1, \ldots, c_n$ flying to SmallTown, but it does not exclude other unnamed city doing so.

The easiest way to overcome this problem is to assume that the named constants are the only individuals of interest.

Def. The closed-world assumption with domain-closure is

$$KB^{\diamond} = KB^{+} \cup \{\forall x [x = c_1 \vee \ldots \vee x = c_n]\},$$

where $c_1, \ldots, c_n$ are all the constant symbols appearing in KB.

The entailment in CWA with domain-closure is defined as following:

$$KB \models_{CD} \alpha \quad \text{iff} \quad KB^{\diamond} \models \alpha.$$

The main properties are:

$KB \models_{CD} \forall x \alpha$ iff $KB \models_{CD} \alpha_c^x$ for every c appearing in KB;

$KB \models_{CD} \exists x \alpha$ iff $KB \models_{CD} \alpha_c^x$ for some c appearing in KB.

Compared to CWA, in $KB^{\diamond}$ we make the additional assumption that no other objects exist apart from the named constants.

Now, under CWA with domain-closure, our query $\neg \exists x$ DirectConnect$(x, \text{smallTown})$ is entailed.

Obs. it is the case that $KB \models_{CD} \alpha$ or $KB \models_{CD} \neg \alpha$ for any α (with or without quantifiers).

II Circumscription

One way to express exceptional cases where a default should not apply is by using a predicate Ab (from abnormal):

$$\forall x \, [\, Bird(x) \wedge \neg Ab(x) \supset Flies(x) \,].$$

Assuming that in KB we have the additional fact:

Bird (chilly)
Bird (tweety)
(tweety ≠ chilly)
¬ Flies (chilly)

we would like to conclude that Tweety flies, whereas Chilly does not.

But KB ⊭ Flies(tweety) because there are interpretations that satisfy KB where Flies(tweety) is false. In these interpretations, the denotation of Tweety is included in the interpretation of Ab.

The strategy for minimizing abnormality: we consider the interpretations of the KB where the interpretation of Ab is (a set) as small as possible. The default conclusions are true in models of the KB where as few of the individuals as possible are abnormal.

In the previous example, we know that Chilly is an abnormal bird, but we don't know anything about Tweety. The interpretation of Ab must include Chilly, but excludes Tweety (because nothing dictates otherwise). These technique is called circumscribing the predicate Ab.

In general, a family of predicates Ab_i is used to describe various aspects of individuals. Chilly may be in the interpretation of Ab_1, but not in that of Ab_2 and so on.

Minimal entailment

A new form of entailment is characterized in terms of properties of interpretations.

Let P be a fixed set of unary predicates Ab. Let $J_1 = \langle D, I_1 \rangle$ and $J_2 = \langle D, I_2 \rangle$ be interpretations over the same domain such that every constant and function is interpreted the same.

We define the relationship $\leq$ as following:

$J_1 \leq J_2$ iff for every $Pr \in P$ then $I_1[Pr] \subseteq I_2[Pr]$.

We say that $J_1 < J_2$ iff $J_1 \leq J_2$ and $J_2 \not\leq J_1$.

J_1 makes the interpretation of all Ab predicates smaller than J_2. In other words, J_1 is more normal than J_2.

Def. The minimal entailment $\models_{\leq}$ is defined as follows:

$KB \models_{\leq} \alpha$ iff for every interpretation J such that $J \models KB$, either $J \models \alpha$ or there is an J' such that $J' < J$ and $J' \models KB$.

In the previous example, $KB \not\models Flies(tweety)$ but $KB \models_{\leq} Flies(tweety)$.

If $J \models KB$ but $J \not\models Flies(tweety)$ then $J \models Ab(tweety)$.

We take J' to be exactly J, except that we remove the denotation of tweety from the interpretation of Ab. Assuming that $P = \{Ab\}$, we have that $J' < J$ and $J' \models KB$.

But $J' \not\models Ab(tweety)$, so $J' \models Flies(tweety)$

Thus, in the minimal models of KB, Tweety is a normal bird $KB \models_{\leq} \neg Ab(tweety)$ therefore $KB \models_{\leq} Flies(tweety)$.

Instead, in all the models of KB, Chilly is an abnormal bird.

In this reasoning, the only default step was to conclude that Tweety was a normal bird; the rest was ordinary deductive reasoning.

Obs. The "most normal" models of the KB may not all satisfy exactly the same sentences.

For example, suppose that the KB contains:

$$KB \begin{cases} Bird(c) \\ Bird(d) \\ \neg Flies(c) \vee \neg Flies(d) \\ \forall x\,[Bird(x) \wedge \neg Ab(x) \supset Flies(x)] \end{cases}$$

In any model of the KB, the interpretation of Ab must contain either the denotation of c or the denotation of d. Any model containing other abnormal individuals would not be minimal (including both c and d).

So, in any minimal model $J \models KB$, we have either $J \models Ab(c)$ or $J \models Ab(d)$.

if $J \models Ab(c)$ then $KB \not\models_{\leq} Flies(c)$. Similarly, if $J \models Ab(d)$ then $KB \not\models_{\leq} Flies(d)$.

We cannot conclude by default that c is a normal bird, nor that d is. But we can conclude by default that one of them is: $KB \models_{\leq} Flies(c) \vee Flies(d)$.

Obs. CWA and GCWA have a different behavior.

Because neither $KB \not\models Ab(c)$ nor $KB \not\models Ab(d)$, it results that

$$KB^+ \supset KB \cup \{\neg Ab(c), \neg Ab(d)\}.$$

So, $KB \models_c (Flies(c) \wedge Flies(d))$, that is KB^+ is not consistent.

On the other hand, under GCWA $\neg Ab(c) \notin KB^*$ and $\neg Ab(d) \notin KB^*$.

$[KB \models Ab(c) \vee \neg Bird(c) \vee Flies(c)$ but $KB \not\models \neg Bird(c) \vee Flies(c)]$

So, under GCWA we cannot conclude anything about Flies(c) or Flies(d) (or their disjunction).

In the circumscription case, one model of the KB is preferred to another one if it has less abnormal individuals. Assuming that we have the statements: Richard Nixon was both quaker (thus implicitly pacifist) and republican (thus implicitly not pacifist), we have the following KB:

$$KB \left\{ \begin{array}{l} Republican(nixon) \wedge Quaker(nixon) \\ \forall x \, [\, Republican(x) \wedge \neg Ab_2(x) \supset \neg Pacifist(x)\,] \\ \forall x \, [\, Quaker(x) \wedge \neg Ab_3(x) \supset Pacifist(x)\,] \end{array} \right.$$

If we circumscribe the predicates Ab_2 and Ab_3, we have two minimal models

$J_1 \models Ab_2(nixon)$ and $J_1 \models Pacifist(nixon)$

$J_2 \models Ab_3(nixon)$ and $J_2 \models \neg Pacifist(nixon)$

So, $KB \not\models_{\leq} Pacifist(nixon)$ and $KB \not\models_{\leq} \neg Pacifist(nixon)$.

If, for example, we give priority to religious convictions rather than to political ones, we can express it by prioritized circumscription, where we prefer the (minimal) model that minimizes Ab_3.

III Default logic

It provides a mechanism that explicitly specifies which sentences should be added to the KB, maintaining consistency. For example, if $Bird(t)$ is entailed by the KB, we might want to add the default assumption $Flies(t)$, if it is consistent to do so.

In default logic, a KB consists of two parts: a set F of first-order sentences and a set D of default rules, which are specifications of what assumptions can be made and when.

The role of the default logic is to specify the following:
- the appropriate set of implicit beliefs that incorporate the facts in $\mathcal{F}$;
- as many default assumptions as possible, given the default rules in $\mathcal{D}$;
- the logical entailments inferred from the implicit beliefs and the default assumptions.

Default rules

A default rule is written in the form $<\alpha : \beta / \sigma>$, where α is the prerequisite, β is the justification and σ is the conclusion. σ is considered to be true if α is true and it is consistent to believe β (that is $\neg\beta$ is not true).

$$<\text{Bird}(\text{tweety}) : \text{Flies}(\text{tweety}) / \text{Flies}(\text{tweety})>$$

A rule where the justification and conclusion are the same is called a normal default rule and it is written as $\text{Bird}(\text{tweety}) \Rightarrow \text{Flies}(\text{tweety})$.

A rule can be formulated using free variables:

$<\text{Bird}(x) : \text{Flies}(x) / \text{Flies}(x)>$ represents the set of all its instances, formed by replacing x by a ground term.

Default extensions

Given a default theory $KB = (\mathcal{F}, \mathcal{D})$, what are the sentences that should be believed?

We define an extension of the theory as a reasonable set of beliefs given a default theory.

Def. A set of sentences $\mathcal{E}$ is an extension of a default theory $(\mathcal{F}, \mathcal{D})$ if for every sentence π we have:

$$\pi \in \mathcal{E} \text{ iff } \mathcal{F} \cup \{\sigma \mid <\alpha : \beta / \sigma> \in \mathcal{D}, \alpha \in \mathcal{E}, \neg\beta \notin \mathcal{E}\} \models \pi.$$

Thus, an extension is the set of all entailments of $\mathcal{F} \cup \Delta$, where Δ is a suitable set of assumptions.

Obs. The definition of $\mathcal{E}$ does not say how to find an $\mathcal{E}$, but $\mathcal{E}$ is completely characterized by its set of applicable assumptions Δ.

Example

$$\mathcal{F} = \{ Bird(tweety),\ Bird(chilly),\ \neg Flies(chilly) \}$$
$$\mathcal{D} = \{ Bird(x) \Rightarrow Flies(x) \}$$

Let $\mathcal{E} = \mathcal{F} \cup \{ Flies(tweety) \}$.

$Flies(tweety)$ is the only assumption applicable to $\mathcal{E}$.

$\left.\begin{array}{l} Bird(tweety) \in \mathcal{E} \\ \neg Flies(tweety) \notin \mathcal{E} \end{array}\right\} \Rightarrow Flies(tweety)$ is applicable.

$Flies(t)$ is not applicable for any other t (in our example t could be only chilly).

Thus, $Flies(tweety)$ is the only applicable assumption, so $\mathcal{E}$ is an extension (it can be proven that it is the only one).

Obs. An extension $\mathcal{E}$ of a default theory $(\mathcal{F}, \mathcal{D})$ is inconsistent iff $\mathcal{F}$ is inconsistent.

Multiple extensions

Consider the following default theory:

$$\mathcal{F} = \{ Republican(nixon),\ Quaker(nixon) \}$$
$$\mathcal{D} = \{ Republican(x) \Rightarrow \neg Pacifist(x),\ Quaker(x) \Rightarrow Pacifist(x) \}.$$

Let $\mathcal{E}_1$ be the extension characterized by the assumption $Pacifist(nixon)$ and $\mathcal{E}_2$ characterized by the assumption $\neg Pacifist(nixon)$.

$\mathcal{E}_1$ and $\mathcal{E}_2$ are extensions because their assumptions are applicable and there are no other applicable ones (for $t \neq nixon$).

The empty set of assumptions does not give an extension, because both Pacifist(nixon) and ¬Pacifist(nixon) would be applicable. For any other extensions, assumptions would be of the form Pacifist(t) or ¬Pacifist(t), but none are applicable for $t \neq$ nixon.
Thus, $\mathcal{E}_1$ and $\mathcal{E}_2$ are the only extensions possible.

On the basis of what we know, either Nixon is a pacifist or he is not a pacifist are reasonable beliefs. There are two options:

1. a skeptical reasoner will only believe those sentences that are common to all extensions of the default theory;
2. a credulous reasoner will simply choose an extension as a set of sentences to believe.

In some cases, the existance of multiple extensions is an indication that we have not said enough to make a reasonable decision.

In the previous example, we may want to say that the default about Quakers should apply only to individuals that are not politically active. If we add in ∃ the fact

$$\forall x\,[\text{Republican}(x) \supset \text{Political}(x)],$$

we can replace the rule $\text{Quaker}(x) \Rightarrow \text{Pacifist}(x)$ by a non-normal one:

$$\frac{\text{Quaker}(x) : [\text{Pacifist}(x) \wedge \neg \text{Political}(x)]}{\text{Pacifist}(x)}.$$

For ordinary Quakers, the assumption is that they are pacifists. But for Quaker Republicans like Nixon, we assume that they are not pacifists.
If we replace $\forall x\,[\text{Republican}(x) \supset \text{Political}(x)]$ by the default rule $\text{Republican}(x) \Rightarrow \text{Political}(x)$, then we have two extensions:
- one characterized by the assumptions {¬Pacifist(nixon), Political(nixon)};
- one characterized by the assumption {Pacifist(nixon)}.

Resolving conflicts among default rules is crucial when we deal with concept hierarchies.

For example, for the following KB:

$$\mathcal{F} = \{\ \forall x\,[\,Penguin(x) \supset Bird(x)\,],\ Penguin(chilly)\ \}$$

$$\mathcal{D} = \{\ Bird(x) \Rightarrow Flies(x),\ Penguin(x) \Rightarrow \neg Flies(x)\ \}$$

we have two extensions: one where Chilly is assumed to fly and one where Chilly is assumed not to fly.

The default that penguins do not fly should preempt the more general default that birds fly.

$$\frac{Bird(x) : [\,Flies(x) \wedge \neg Penguin(x)\,]}{Flies(x)}$$

Unlike defaults in an inheritance mechanism, the default logic do not automatically prefer the most specific defaults.

Vagueness, uncertainty and degrees of belief

In some situations, it is not suitable to express general statements using logical universals. We have seen different ways to represent defaults, but sometimes other extensions for knowledge representation are necessary.
Besides the intrinsec imprecision of statements like "someone is somewhat tall", the way the conclusions are formulated may also be imprecise (e.g. in medical field, a rule may not be applicable in 100% of cases).

Noncategorical reasoning

We distinguish three ways to "relax" the categorical nature of classical logic, in order to make the universal $\forall x\, P(x)$ more flexible:

1. Relaxation of the strength of the quantifier - instead of "for all x" we say "for % of x".
   "95% of the persons in this group are master students"
   This is a statistical interpretation. The use of probability in these sentences is objective (it is an assertion about the frequency of an event, it is not a subjective interpretation or a degree of confidence).
2. Relaxation of the applicability of a predicate - instead of a statement like "Everyone in my group is (absolutely) tall", we say "Everyone in my group is (moderately) tall".
   The predicate "tall" applies to an individual to a greater or lesser extent. These predicates are called vague.
   A person may be considered both tall (moderately) and short (weakly) at the same time.

3. Relaxation of the degree of belief in a sentence – instead of having the statement "Everyone in this group is a master student" we say "I believe that everyone in this group is a master student, but I am not sure". This is uncertain knowledge and it can be quantified by using the concept of subjective probabilities.

Objective probability

It refers to the frequency of a single event happening and it does not depend on who is assessing the probability. It is best applied to situations like "coin flipping" or "card drawing".

Subjective probability

The degree of confidence (also called subjective probability) in a sentence is separable from the content of the sentence. The degree of belief in a sentence can vary, regardless of how vague or categorical the sentence may be. For example, we may be absolutely certain that Bill is quite tall, while we may only suspect that he is married.

With subjective beliefs, we express degrees of confidence rather than black-and-white conclusions.

Subjective probabilities can be mechanically computed like the objective ones, but they are used in a different way. We are interested in how evidences combine to change our degree of confidence in a belief, rather than simply deriving new conclusions.

Def The prior probability of a sentence α involves the prior state of information (or background knowledge) β.
We write it $Pr(\alpha|\beta)$.

For example, suppose that we know that 0.2% of the population has hepatitis. Based on just that, our degree of belief that John (a randomly chosen individual) has hepatitis is 0.002.

Def A posterior probability is derived when new evidence is considered: $P(\alpha \mid \beta \wedge \gamma)$ where γ is the new evidence.
For example, if John is yellowish, given the symptoms and the prior probability, we may conclude that the posterior probability of John having hepatitis is 0.65.

The key problem is how to combine evidences from different sources to reevaluate our beliefs.

A basic Bayesian approach

Suppose that we have a number of atomic sentences of interest $p_1, \ldots, p_n$ (e.g. Eric is tall, Ane is married, George is a teacher and so on). In different interpretations, different combinations of these sentences will be true. Let J be an interpretation that specifies which sentences are true/false.

Def. The joint probability distribution $\mathcal{J}$ is the specification of the degree of belief for each of the 2^n truth assignments

$$\sum_J \mathcal{J}(J) = 1 \quad \text{and} \quad \mathcal{J}(J) \in [0,1].$$

The degree of belief in any sentence α is defined as

$$P_{\mathcal{J}}(\alpha) = \sum_{J \models \alpha} \mathcal{J}(J).$$

Knowing that $P_{\mathcal{J}}(\alpha \mid \beta) = \dfrac{P_{\mathcal{J}}(\alpha \wedge \beta)}{P_{\mathcal{J}}(\beta)}$, the degree of belief that John is tall given that he is male from California is

$$\frac{P_{\mathcal{J}}(\text{John is tall, John is male, John is from California})}{P_{\mathcal{J}}(\text{John is male, John is from California})}.$$

For n atomic sentences, we need to specify $2^n - 1$ values. This is unachievable for any practical applications.

Belief (or Bayesian) networks

Suppose that we have the atomic sentences $p_1, \ldots, p_n$. We can specify an interpretation using $\langle P_1, \ldots, P_n \rangle$, where each P_i is p_i (when the sentence is true) or $\neg p_i$ (when the sentence is false). We have that

$$J(\langle P_1, \ldots, P_n \rangle) = P_r(P_1 \wedge P_2 \ldots \wedge P_n)$$

because there is only one interpretation that satisfies $P_1 \wedge \ldots \wedge P_n$.

We represent all the variables p_i in a directed acyclic graph, called a belief (or Bayesian) network.

There is an arc from p_i to p_j if the truth of p_i directly affects the truth of p_j (the former is a parent of the letter). We assume that the variables are numbered such that the parents of any p_j appear earlier in the sequence than p_j (we can do that because the graph is acyclic).

According to the chain rule in probabilities, we have:

$$J(\langle P_1, \ldots, P_n \rangle) = P_r(P_1) \cdot P_r(P_2 \mid P_1) \ldots P_r(P_n \mid P_1 \wedge \ldots \wedge P_{n-1}).$$

To compute the joint probability distribution, we still need $2^n - 1$ values because for each term $P_r(P_{j+1} \mid P_1 \wedge \ldots \wedge P_j)$ there are 2^j conditional probabilities to specify and $\sum_{j=0}^{n-1} 2^j = 2^n - 1$.

In order to reason about subjective probabilities, some simplifying assumptions are necessary.

We will assume that each propositional variable in the belief network is conditionally independent from the nonparent variables, given the parent variables.

$$Pr(P_{j+1} \mid P_1 \wedge \ldots \wedge P_j) = Pr(P_{j+1} \mid \text{parents}(P_{j+1}))$$

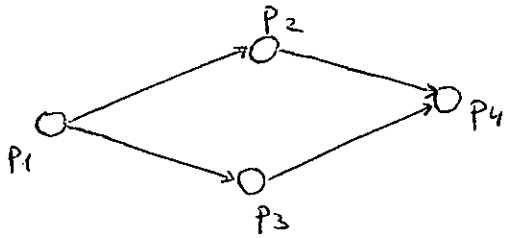
With these independence assumptions, it follows that

$$J(\langle P_1, \ldots, P_n \rangle) = Pr(P_1 \mid \text{parents}(P_1)) \cdot \ldots \cdot Pr(P_n \mid \text{parents}(P_n)).$$

To fully specify J, we need to know $Pr(P \mid \text{parents}(P))$ for each variable p.

If K is the maximum number of parents for any node, then we have no more than $n \cdot 2^K$ values to specify.

For the belief network in the figure below



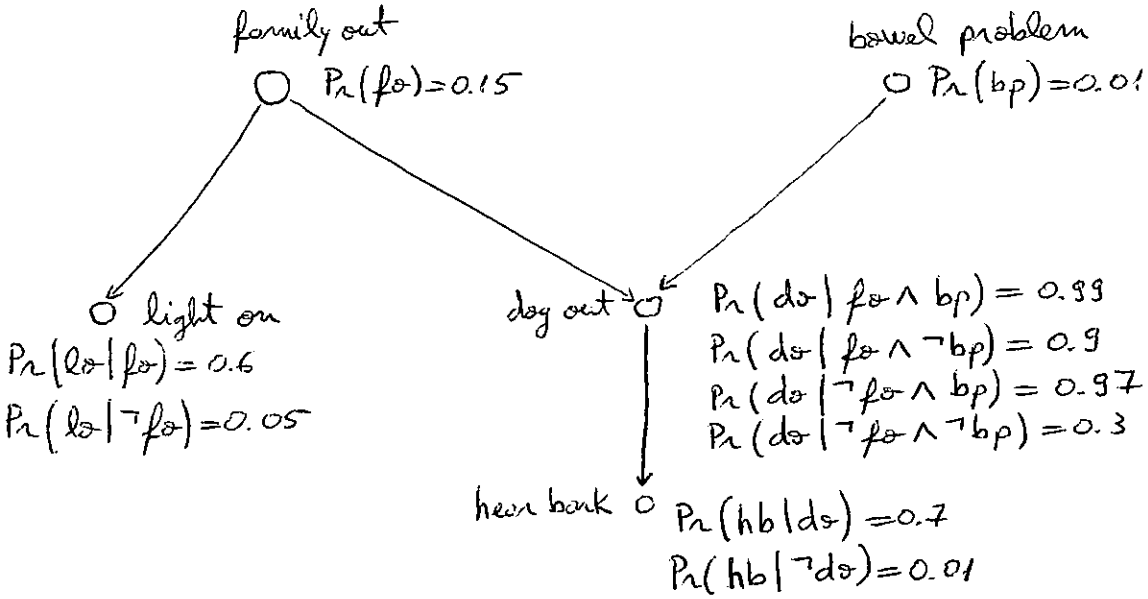
we have $J(\langle P_1, P_2, P_3, P_4 \rangle) = Pr(P_1) \cdot Pr(P_2 \mid P_1) \cdot Pr(P_3 \mid P_1) \cdot Pr(P_4 \mid P_2 \wedge P_3)$.

We now need $(1+2+2+4) = 9$ values rather than 15 (without the independence assumption) to compute J.

An example (due to Eugene Charniak)

We have a family with a dog. We usually put the dog out (do) when the family is out (fo). We also put the dog out when it has bowel problem (bp). A reasonable proportion of time when the dog is out, you can hear it barking (hb) when you approach the house. We usually leave the light on (lo) outside the house when the family is out.

Using these facts, we can construct the following belief network:



We assume the following about the joint probability distribution:

$$J(\langle FO, LO, BP, DO, HB\rangle) = Pr(FO) \cdot Pr(LO|FO) \cdot Pr(BP) \cdot Pr(DO|FO \wedge BP) \cdot Pr(HB|DO).$$

We need $1+2+1+4+2 = 10$ values to specify the joint probability distribution.

Using this belief network, we want to calculate the probability that the family is out, given that the light is on and we don't hear barking.

$$Pr(fo|lo \wedge \neg hb) = \frac{Pr(fo \wedge lo \wedge \neg hb)}{Pr(lo \wedge \neg hb)} = \frac{\sum_{BP,DO} J(\langle fo, lo, BP, DO, \neg hb\rangle)}{\sum_{FO,BP,DO} J(\langle FO, lo, BP, DO, \neg hb\rangle)}$$

1. $J(\langle fo, lo, bp, do, \neg hb\rangle) = Pr(fo) \cdot Pr(lo|fo) \cdot Pr(bp) \cdot Pr(do|fo \wedge bp) \cdot (1 - Pr(hb|do))$
$= 0.15 \cdot 0.6 \cdot 0.01 \cdot 0.99 \cdot 0.3$
2. $J(\langle fo, lo, bp, \neg do, \neg hb\rangle) = 0.15 \cdot 0.6 \cdot 0.01 \cdot 0.01 \cdot 0.99$
3. $J(\langle fo, lo, \neg bp, do, \neg hb\rangle) = 0.15 \cdot 0.6 \cdot 0.99 \cdot 0.9 \cdot 0.3$
4. $J(\langle fo, lo, \neg bp, \neg do, \neg hb\rangle) = 0.15 \cdot 0.6 \cdot 0.99 \cdot 0.1 \cdot 0.99$

5. $J(\langle \neg fo, lo, bp, do, \neg hb \rangle) = 0.85 \cdot 0.05 \cdot 0.01 \cdot 0.97 \cdot 0.3$

6. $J(\langle \neg fo, lo, bp, \neg do, \neg hb \rangle) = 0.85 \cdot 0.05 \cdot 0.01 \cdot 0.03 \cdot 0.99$

7. $J(\langle \neg fo, lo, \neg bp, do, \neg hb \rangle) = 0.85 \cdot 0.05 \cdot 0.99 \cdot 0.3 \cdot 0.3$

8. $J(\langle \neg fo, lo, \neg bp, \neg do, \neg hb \rangle) = 0.85 \cdot 0.05 \cdot 0.99 \cdot 0.7 \cdot 0.99$

So, $Pr(fo \mid lo \wedge \neg hb) = \dfrac{1. + 2. + 3. + 4.}{1. + 2. + 3. + 4. + 5. + 6. + 7. + 8.}$

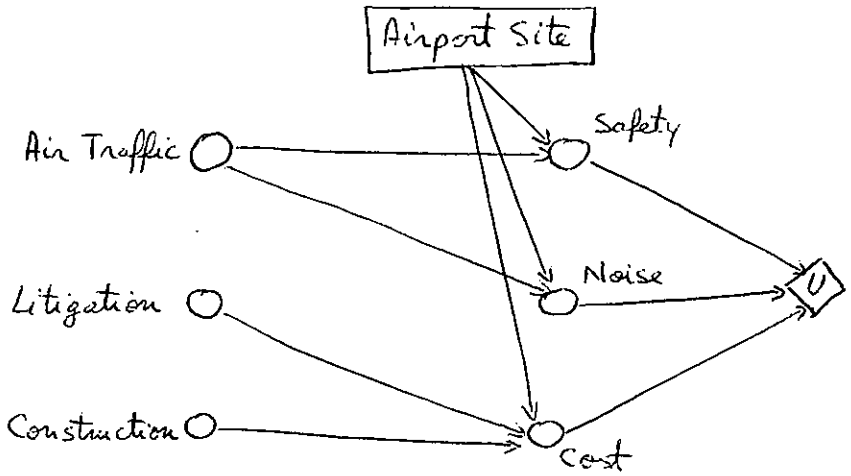
Decision networks (influence diagrams)

They are general decision mechanisms that combine Bayesian networks with additional node types for actions and utilities.

A decision network represents information about an agent's current state, its possible actions, the resulting state from the agent's action and the utility (value) of that state.

There are three types of nodes:

- Chance nodes (as circles) represent probabilistic variables, just like they do in Bayesian networks.
  In decision networks, the parent nodes can include chance nodes as well as decision nodes.
- Decision nodes (as rectangles) represent decisions made by the agent. In the following figure, the AirportSite action can take on a different value for each site under consideration.
  The choice influences the cost, safety and noise that will result.
- Utility (value) node (as diamond) – represent the agent's utility function – there is only one such a node.
  It has parents all variables describing the outcome that directly affect utility. The utility is expressed as a function of the parents attributes.



Evaluating decision networks

For each possible value of the decision node, the resulting utility is calculated. The action (decision) with the highest utility (value) will be chosen.
The algorithm is the following:

1. set the evidence variables for the current state.
2. for each possible value of the decision node:
   - calculate the posterior probabilities for the parent nodes of the utility node (using, for example, a Bayesian network).
   - calculate the resulting utility for that action.
3. return the action with the highest utility.

Representing ignorance – Dempster-Shafer Theory

It is designed to deal with the distinction between uncertainty and ignorance. Rather than computing the probability of a proposition, it computes a lower and an upper bound on the probability of a proposition.

if we have an unbiased coin, the degree of belief that we get heads if we flip it would be 0.5.
if we have a biased coin, due to lack of information, we may want to say only that the degree of belief lies between some limits within $[0, 1]$.

These limits are called belief and plausibility.
For an unbiased coin, we have 0.5 belief and 0.5 plausibility that the result is heads. For an unknown coin, we have 0 belief that we get heads and 1 plausibility.

Thus, the value of a propositional variable is represented by a range, called the possibility distribution of the variable.

Example - suppose that we have a database with names of people and their believed ages. In the case of complete knowledge, the ages would be values. But if we do not know the exact age, we may specify it by a range.

Mary	[18, 22]
Ana	[20, 24]
George	[35, 40]
David	[27, 33]
Cris	[20, 23]

Given an interval Q, rather than asking if $age(x) \in Q$, it is more natural to ask about the possibility of $age(x) \in Q$.
For example, if $Q = [19, 24]$ then it is possible that $age(Mary) \in Q$; it is not possible that $age(George) \in Q$; and it is certain that $age(Cris) \in Q$.

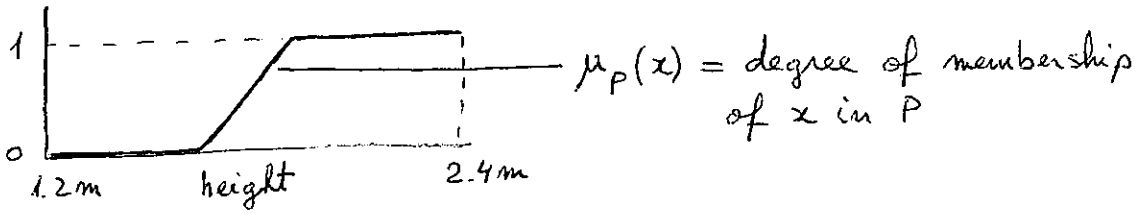
Consider now that we ask what is the probability that the age of a randomly selected individual is in Q?
The belief in this proposition is 2/5 (because of Ana and Cris) and the plausibility is 3/5 (Mary, Ana and Cris). So, the answer is [0.4, 0.6].

The Dempster - Shafer rule combines multiple sources of information with varying levels of knowledge and confidence.

Vagueness

It refers to the degree to which certain predicates are satisfied. For each vague predicate, there is a corresponding base function in terms of which the predicate is understood. (for "tall" the base function is "height").

Def. The degree curve is a function that capture the relationship between a vague predicate and its base function.



An important thing is that an object's degree of satisfaction can be nonzero for multiple predicates over the same base function (e.g. "short" and "tall").

Negation, conjunction and disjunction of vague predicates:

$$\mu_{\neg P} = 1 - \mu_P$$

$$\mu_{P \wedge Q} = \min(\mu_P, \mu_Q)$$

$$\mu_{P \vee Q} = \max(\mu_P, \mu_Q)$$

In a typical application, called fuzzy control, vague predicates are used in production rules.

Unlike standard production systems where a rule either applies or not, here the antecedent of a rule will apply to some degree and the action will be affected to a proportional degree. Such a system enables inferences even when the antecedent conditions are only partially satisfied.

Example – We are given the following rules:

1. If the service is (poor) or the food is (rancid) then the tip is (stingy).
2. If the service is (good) then the tip is (normal).
3. If the service is (excellent) or the food is (delicious) then the tip is (generous).

Assume that service and food quality are described by numbers on a linear scale (e.g. a number from 0 to 10). The amount of tip is represented as a percentage of the cost of the meal (e.g. 10%).

For each of the eight predicates in the example, we are given a degree curve. The base functions are: service, food quality or tip.

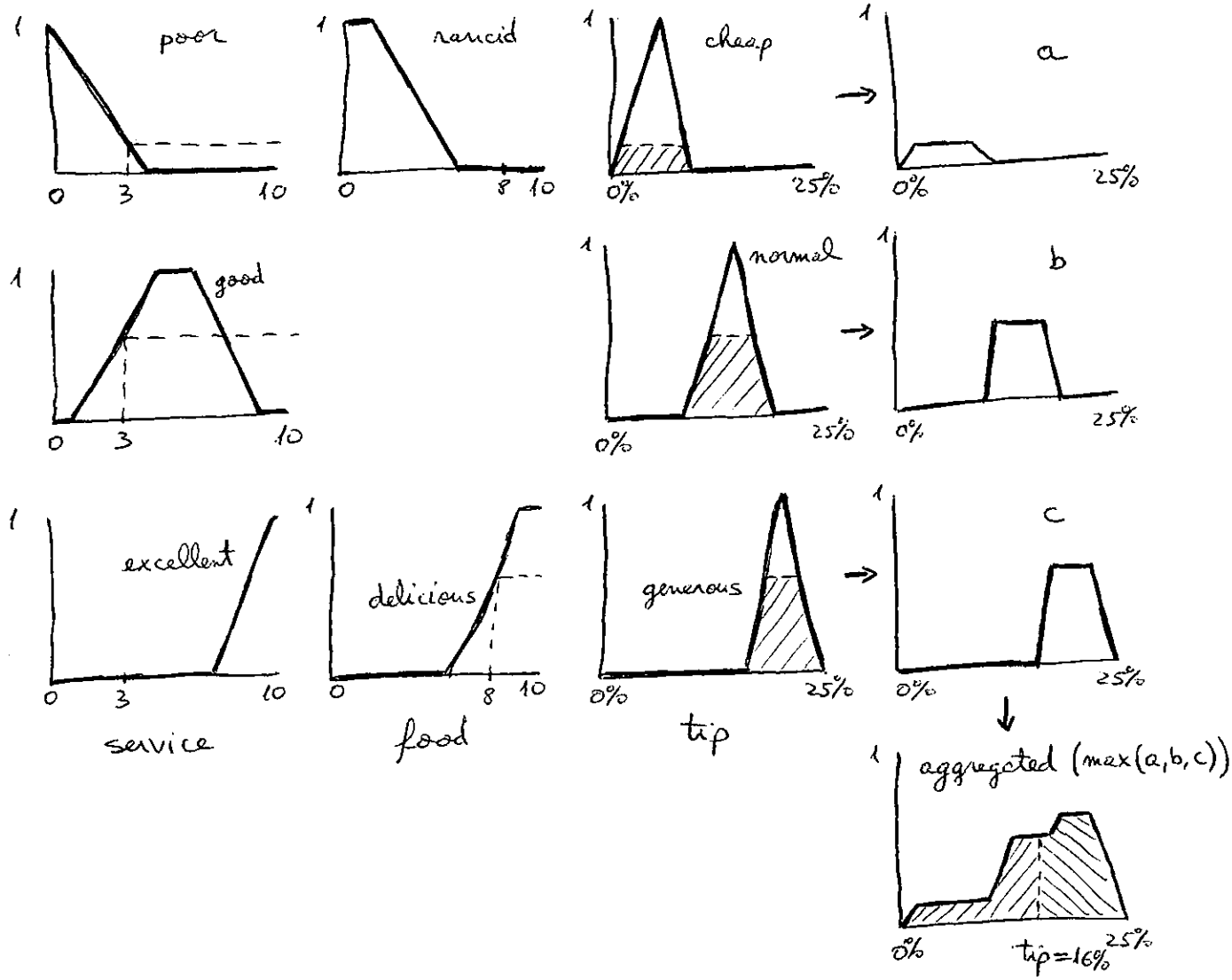
Problem: Given the ratings for the service and for the food, calculate the tip, subject to the rules above.

e.g. service = 3, food = 8, tip = ?

Algorithm

1. transform the inputs into the degrees to which each of the vague predicates used in the antecedents hold.
2. evaluate the antecedents – combine the degrees of applicability of all the predicates in the antecedent of a rule.
3. evaluate the consequents by determining the degrees to which the predicates of each consequent side should be satisfied.
   An intuitive way is that the consequent should hold only to the degree that the rule is applicable.
4. aggregate the consequents - obtain a single degree curve for the "tip" base function.

5. defuzzify the output – generate a value for the tip from the aggregated degree curve at step 4.
One way to do that is to take the center of the area under the curve.



A Bayesian reconstruction

Much of the reasoning with vague predicates can be formulated in terms of subjective probability.
The vague predicates are now treated as ordinary ones, true in some interpretations, false in others.

For a predicate (e.g. Tall), we associate a base measure (e.g. height). We have sentences like Tall(bill) and height(bill) = n, where n is a number.

$$\sum_{n=1.2}^{2.4} Pr(\text{height}(\text{bill}) = n) = 1$$

We reinterpret the "degree of tallness for height of x" as "degree of belief in tallness given the height of x"

$$Pr(\text{Tall}(x) \mid \text{height}(x) = n).$$

If α and β are not independent, we assume that

$$Pr(\alpha \wedge \beta \mid \gamma) = \min\{Pr(\alpha \mid \gamma), Pr(\beta \mid \gamma)\}$$
$$Pr(\alpha \vee \beta \mid \gamma) = \max\{Pr(\alpha \mid \gamma), Pr(\beta \mid \gamma)\}$$

For the example about tips, in subjective terms we are interested to calculate

$$\text{AveragedTip} = \sum_z z \cdot Pr((\text{tip} = z) \mid (\text{food} = x) \wedge (\text{service} = y)).$$

We have that $Pr((\text{tip} = z) \mid (\text{food} = x) \wedge (\text{service} = y)) =$

$$\sum_{G,N,S} Pr((\text{tip} = z) \wedge G \wedge N \wedge S \mid (\text{food} = x) \wedge (\text{service} = y)) =$$

$$\sum_{G,N,S} Pr((\text{tip} = z) \mid G \wedge N \wedge S \wedge (\text{food} = x) \wedge (\text{service} = y)) \cdot$$
$$Pr(G \wedge N \wedge S \mid (\text{food} = x) \wedge (\text{service} = y)),$$

where G is Generous or its negation, N is Normal or its negation, and S is Stingy or its negation.

We assume that the tip is completely determined given G, N and S, therefore

$$Pr((\text{tip} = z) \mid G \wedge N \wedge S \wedge (\text{food} = x) \wedge (\text{service} = y)) =$$
$$Pr((\text{tip} = z) \mid G \wedge N \wedge S).$$

From the Bayes' rule, we have that

$$Pr((\text{tip} = z) \mid G \wedge N \wedge S) = \frac{Pr(G \wedge N \wedge S \mid (\text{tip} = z)) \cdot Pr((\text{tip} = z))}{Pr(G \wedge N \wedge S)} =$$

$$\frac{Pr(G \wedge N \wedge S \mid (\text{tip} = z)) \cdot Pr((\text{tip} = z))}{\sum_u Pr(G \wedge N \wedge S \wedge (\text{tip} = u))} =$$

$$\frac{Pr(G \wedge N \wedge S \mid (\text{tip} = z)) \cdot Pr((\text{tip} = z))}{\sum_u Pr(G \wedge N \wedge S \mid (\text{tip} = u)) \cdot Pr((\text{tip} = u))}.$$

Assuming that all tips are a priori equally likely

$$Pr((tip = z)) = Pr((tip = u)),$$

we obtain that $Pr((tip = z) \mid G \wedge N \wedge S) = \frac{Pr(G \wedge N \wedge S \mid (tip = z))}{\sum_u Pr(G \wedge N \wedge S \mid (tip = u))}$.

$$Pr(G \wedge N \wedge S \mid (tip = u)) = \min \{ Pr(G \mid (tip = u)), Pr(N \mid (tip = u)), Pr(S \mid (tip = u)) \}$$

can be calculated from the given degree curves for the predicates Stingy, Generous and Normal.

$$Pr(G \wedge N \wedge S \mid (food = x) \wedge (service = y)) = \min \{ Pr(G \mid (food = x) \wedge (service = y)), Pr(N \mid (food = x) \wedge (service = y)), Pr(S \mid (food = x) \wedge (service = y)) \}.$$

From the production rules, we assume that

$$Pr(G \mid (food = x) \wedge (service = y)) = \max \{ Pr(Excellent \mid (food = x) \wedge (service = y)), Pr(Delicious \mid (food = x) \wedge (service = y)) \}$$

Considering the food quality to be independent of the service quality, we obtain that

$$Pr(G \mid (food = x) \wedge (service = y)) = \max \{ Pr(Excellent \mid (service = y)), Pr(Delicious \mid (food = x)) \}$$

that can be calculated from the degree curves for Excellent and Delicious.

Similarly,

$$Pr(N \mid (food = x) \wedge (service = y)) = Pr(Good \mid (service = y))$$

and

$$Pr(S \mid (food = x) \wedge (service = y)) = \max \{ Pr(Poor \mid (service = y)), Pr(Rancid \mid (food = x)) \}$$

that can be calculated from the degree curves for Good, Poor and Rancid.