Lecture 07

Knowledge Representation & Reasoning

In Logic and Prolog

Jeremy Pitt
Department of Electrical & Electronic Engineering
Imperial College London, UK
Email: j.pitt _at_ imperial _dot_ ac _dot_ uk
Phone: (int) 46318
Office: 1010 EEE

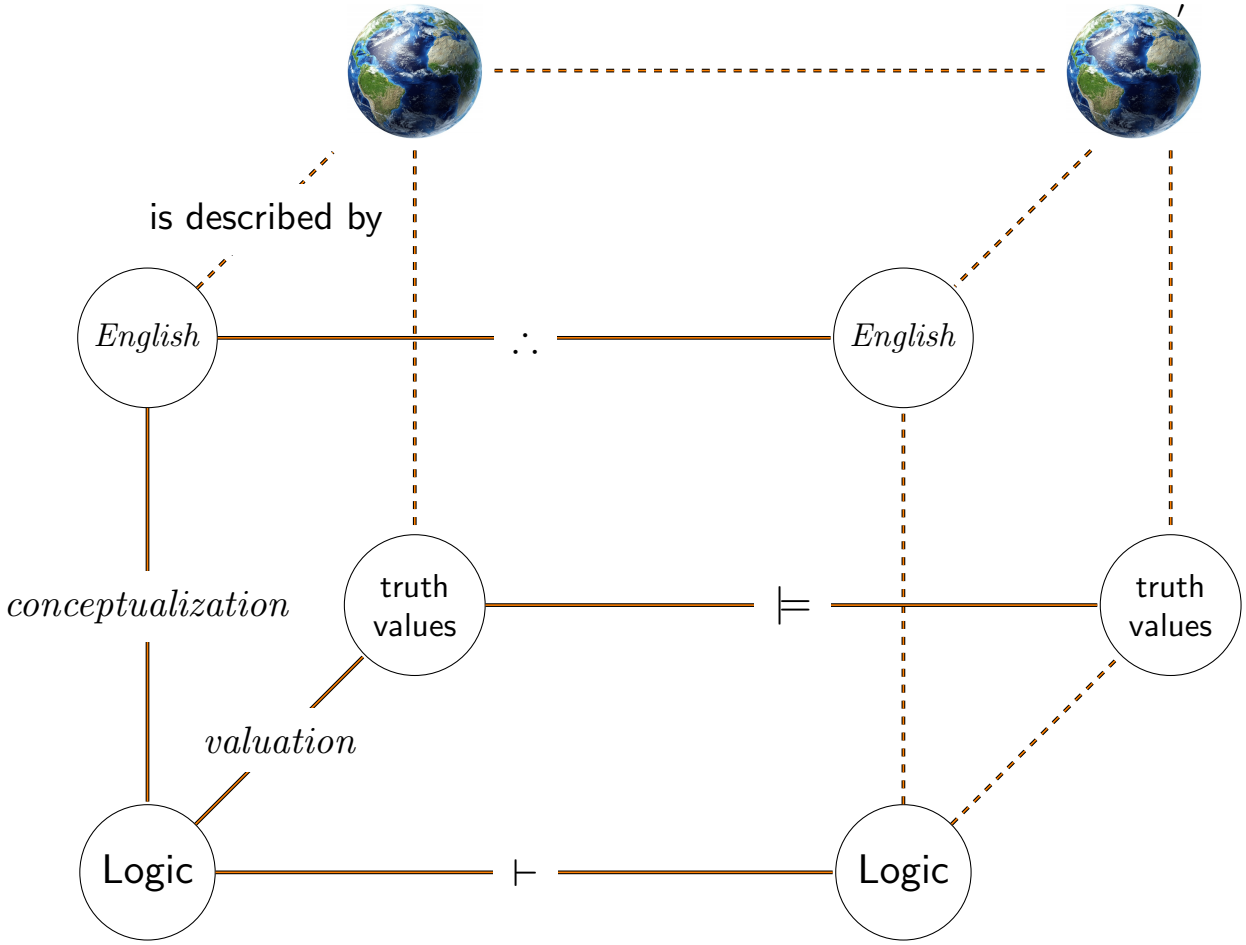
Aims and Objectives

- Aims
  - To show how formal logic can be used to represent 'real world' knowledge
  - To show how knowledge so represented can be reasoned with
  - To expose some of the logical foundations of Prolog programming language

- Objectives
  - Understand the role and use of logic for both representation and reasoning

Logic for Knowledge Representation

- The world is a set of facts
- The facts are expressed in English
- Arguments or reasoning are used to deduce new facts, and a new state of the world (world′)
- Conceptualization is the process of transforming ontological commitments about objects and relations in English into logical statements
- Logical statements are assigned a meaning (truth value) by a valuation
- The semantic and syntactic relationships between sets of formulas (premises) and one formula (conclusion) are given by the $\models$ (entails) and $\vdash$ (proves) relations



Conceptualisation

- Conceptualisation in propositional logic

- Conceptualisation in predicate logic
  - Conceptualisation $= \langle$ objects, functional basis, relational basis $\rangle$
  - Objects: make up the domain of discourse
    - May be concrete or abstract, real or fictional, ...
  - Functions: the functional basis of the conceptualisation
    - Maps objects to other objects
  - Relations: the relational basis of the conceptualisation
    - Relations hold between any number of n objects
  - Issues
    - Granularity (similar to problem space formulation)
    - Reification
    - Uniqueness
      - For a domain of d individuals, there d^n n-tuples
      - 2^{b^n} possible sets for each relation

Reasoning: The Issues

- Given an initial set of statements (assertions, or premises), we want to carry out deduction, inference, argument or reasoning with it
  - An argument derives a new statement (conclusion) from the initial set
  - Syllogistic reasoning

All As are Bs	All men are mortal
Some C is an A	Socrates is a man
$\therefore$ Some C is a B	$\therefore$ Socrates is mortal

- Consistency: the initial set cannot contain any contradictions

- Soundness: If the premises are all true, then so should the conclusion be
  - Note it is asserted that the premises are true: we are concerned with the soundness of the argument not the truth of the assertions
  - See example arguments in Appendix A

Reasoning

- We want to know if a formula (the conclusion) is a logical consequence of a (possibly empty) set of formulas (the premises)
  - In propositional logic, if every valuation that satisfies the premises, satisfies the conclusion
  - In predicate logic, if every interpretation that satisfies the premises, satisfies the conclusion

- In propositional logic, **valid** arguments can be defined semantically using truth tables (compute membership of the entailment relation)

- However truth tables cannot be used to provide an account of validity in predicate logic because there are no truth tables for quantified expressions

- We have to use another technique – compute membership of the proves relation

Proof Systems

- We want to demonstrate logical consequence (entailment)
- Semantic proof checker for propositional logic
  - Use a truth table to check all possible values
  - For formulas with n propositional symbols, this requires checking 2^n valuations
  - This is already exponential
  - And anyway won't work with predicate and modal logics – we can't check all the models
- Have to do something else: use syntax to show $\mathcal{S} \vdash P$
  $\mathcal{S}$ **proves** P
- How to think of the proves relation $\vdash$
- How does this relate to $\models$?

Requirements of Proof Systems

- Soundness: If we prove a theorem, we want to be sure it is valid
  - $\vdash \rightarrow \models$
  - Soundness is pretty much essential
- Completeness: If we proposition is valid, we want to be able to prove it
  - $\models \rightarrow \vdash$
  - Completeness is desirable
- Decidable: if we try to prove a theorem or a non-theorem, will we ever stop
  - Propositional logic: yes
  - Predicate logic: semi-decidable (if a proof exists, we will find it; it one does not exist, we cannot be guaranteed to stop)
- Efficiency: we are looking at exponential complexity

Normal Forms

- Propositional Logic only needs negation and one other connective
  - All other connectives can be defined in terms of $\neg$ and the one connective by applying (semantic) **equivalences**
  - By repeatedly applying equivalences, a formula can be converted into its **normal form**
  - A typical normal form
    - Eliminates some connectives
    - Uses others in a restricted manner
    - Makes the formula easier to process (but harder to read)
    - May be exponentially larger than the original formula

Some Definitions

- Define
  - A **literal** is an atomic formula or its negation
  - A **maxterm** is a literal or a disjunction of literals
  - A **mintem** is a literal or a conjunction of literals

- A formula is in **negation normal form** (NNF) if
  - The only connectives used are $\wedge$, $\vee$ and $\neg$
  - $\neg$ only applies to atomic formulas (i.e. literals)

- $P_1 \wedge P_2 \wedge \ldots \wedge P_n$ is in **conjunctive normal form** (CNF) if
  - Each $P_i \, (1 \leq i \leq n)$ is a maxterm (CNF: conjunction of disjunctions)

- $P_1 \vee P_2 \vee \ldots \vee P_n$ is in **disjunctive normal form** (DNF) if
  - Each $P_i \, (1 \leq i \leq n)$ is a minterm (DNF: disjunction of conjunctions)

Conversion into Normal Form

- Every formula can be converted into its equivalent in NNF, CNF, or DNF
  - Step 1: Eliminate $\rightarrow$ and $\leftrightarrow$
    - $P \leftrightarrow Q \equiv P \rightarrow Q \wedge Q \rightarrow P$
    - $P \rightarrow Q \equiv \neg P \vee Q$
  - Step 2: 'push negation in' until it applies only to literals (NNF)
    - $\neg\neg P \equiv P$
    - $\neg(P \wedge Q) \equiv \neg P \vee \neg Q$
    - $\neg(P \vee Q) \equiv \neg P \wedge \neg Q$
  - Step 3: 'push disjunction in' until it applies only to literals (CNF)
    - $P \vee (Q \wedge R) \equiv (P \vee Q) \wedge (P \vee R)$
    - $(Q \wedge R) \vee P \equiv (Q \vee P) \wedge (R \vee P)$
  - Step 4: simplify
    - Delete: $P \vee \neg P \equiv \mathbf{true}$
    - Reduce: $P \wedge (P \vee Q) \equiv P$
    - Replace: $(P \vee Q) \wedge (P \vee \neg Q) \equiv P$

Tautology Checking Using Normal Forms

- Simplify the formula to 'simplest' form of CNF
  - Each simplification preserves semantic equivalence
  - Either the simplified formula reduce to true, e.g.

$$
\begin{aligned}
& ((p \to q) \to p) \to p \\
\equiv\ & \neg((p \to q) \to p) \vee p \\
\equiv\ & \neg(\neg(p \to q) \vee p) \vee p \\
\equiv\ & \neg(\neg(\neg p \vee q) \vee p) \vee p \\
\equiv\ & (\neg\neg(\neg p \vee q) \wedge \neg p) \vee p \\
\equiv\ & ((\neg p \vee q) \wedge \neg p) \vee p \\
\equiv\ & (\neg p \vee q \vee p) \wedge (\neg p \vee p) \\
\equiv\ & \mathbf{true} \wedge \mathbf{true} \\
\equiv\ & \mathbf{true}
\end{aligned}
$$

- Or we have a valuation that falsifies the formula, e.g.

$$\begin{aligned} & (p \vee q) \rightarrow (q \vee r) \equiv \neg(p \vee q) \vee (q \vee r) \\ \equiv \quad & (\neg p \wedge \neg q) \vee (q \vee r) \equiv (\neg p \vee q \vee r) \wedge (\neg q \vee q \vee r) \\ \equiv \quad & (\neg p \vee q \vee r) \wedge \mathbf{true} \equiv (\neg p \vee q \vee r) \end{aligned}$$

- To see why:
  - If the CNF is of the form $P_1 \wedge P_2 \wedge \ldots \wedge P_m$
  - The each P_i is of the form $l_1 \vee l_2 \vee \ldots \vee l_n$
  - We can specify an interpretation τ that each falsifies each l_j
    - If l_j is $\neg p$, then $\tau(p) = \mathbf{true}$
    - If l_j is p, then $\tau(p) = \mathbf{false}$

- Either way we have another way of checking if $(\mathcal{S}, P) \in \models$

Translating English into Logical Form

- Rough guidelines

statement1	p
statement2	q
statement1 and statement2	$p \wedge q$
statement1 or statement2	$p \vee q$
if statement1 then statement2	$p \rightarrow q$
statement1, if statement2	$p \leftarrow q$
statement1 only if statement2	$p \rightarrow q$
statement1 if and only if statement2	$p \leftrightarrow q$
statement1 unless statement2	$\neg q \rightarrow p$
every, all, for all, any	$\forall$
there exists, there is, for some, a	$\exists$
every noun predicate	$\forall x.noun(x) \rightarrow predicate(x)$
some noun predicate	$\exists x.noun(x) \wedge predicate(x)$

- Common sense, intuition, experience, ...

Example

- The domain of discourse
  - Every horse is faster than every dog
  - There is a greyhound that is faster than every rabbit
  - Every greyhound is a dog
  - Faster is transitive
  - Harry is a horse
  - Ralph is a rabbit

- The translation
  - $\forall x.horse(x) \rightarrow \forall y.dog(y) \rightarrow faster(x, y)$
  - $\exists x.greyhound(x) \wedge \forall y.rabbit(y) \rightarrow faster(x, y)$
  - $\forall x.greyhound(x) \rightarrow dog(x)$
  - $\forall x.\forall y.\forall z.(faster(x, y) \wedge faster(y, z)) \rightarrow faster(x, z)$
  - $horse(harry)$
  - $rabbit(ralph)$

Proof (1)

- Now we can prove (and we will, over and over) that
  - $Harry\ is\ faster\ than\ Ralph$
- This is a consistent set of assertions. We are only interested in those interpretations in which all these assertions are true. We want to show that in all those interpretations, the conclusion is also true.
- To do this, we will need some inference rules

Some Inference Rules

- Rules

$$\textit{modus ponens}\ \frac{P \to Q \quad P}{Q} \qquad \wedge\text{-elimination}\ \frac{P \wedge Q}{P \quad Q} \qquad \wedge\text{-introduction}\ \frac{P \quad Q}{P \wedge Q}$$

$$\forall\text{-instantiation}\ \frac{\forall x.P}{P_{\{c/x\}}} \qquad \exists\text{-instantiation}\ \frac{\exists x.P}{P_{\{f(v_1,\ldots,v_n)/x\}}}$$

- Notes
  - $\forall$-instantiation
    - c is a constant (any domain constant) substituted for x in P
  - $\exists$-instantiation
    - f is a *new* function constant
    - $v_1, \ldots, v_n$ are all the free (unquantified) variables in P
    - The function is substituted for x in P
    - If there are no free variables, f is a new object constant (e.g. c)

Therefore ...

	$\forall x.horse(x) \rightarrow \forall y.dog(y) \rightarrow faster(x, y)$	p1
	$\exists x.greyhound(x) \wedge \forall y.rabbit(y) \rightarrow faster(x, y)$	p2
	$\forall x.greyhound(x) \rightarrow dog(x)$	p3
	$\forall x.\forall y.\forall z.(faster(x, y) \wedge faster(y, z)) \rightarrow faster(x, z)$	p4
	$horse(harry)$	p5
	$rabbit(ralph)$	p6
1	$greyhound(c) \wedge \forall y.rabbit(y) \rightarrow faster(c, y)$	$\exists$-instantiation (p2)
2	$greyhound(c)$	$\wedge$-elimination (1)
3	$\forall y.rabbit(y) \rightarrow faster(c, y)$	〃
4	$rabbit(ralph) \rightarrow faster(c, ralph)$	$\forall$-instantiation (3)
5	$faster(c, ralph)$	*modus ponens* (4, p6)
6	$horse(harry) \rightarrow \forall y.dog(y) \rightarrow faster(harry, y)$	$\forall$-instantiation (p1)
7	$\forall y.dog(y) \rightarrow faster(harry, y)$	*modus ponens* (6, p5)
8	$dog(c) \rightarrow faster(harry, c)$	$\forall$-instantiation (7)
9	$greyhound(c) \rightarrow dog(c)$	$\forall$-instantiation (p3)
10	$dog(c)$	*modus ponens* (9, 2)
11	$faster(harry, c)$	*modus ponens* (8, 10)
12	$faster(harry, c) \wedge faster(c, ralph)$	$\wedge$-introduction (10,11)
13	$\forall y.\forall z.(faster(harry, y) \wedge faster(y, z)) \rightarrow faster(harry, z)$	$\forall$-instantiation (p4)
14	$\forall z.(faster(harry, c) \wedge faster(c, z)) \rightarrow faster(harry, z)$	$\forall$-instantiation (13)
15	$(faster(harry, c) \wedge faster(c, ralph)) \rightarrow faster(harry, ralph)$	$\forall$-instantiation (14)
16	$faster(harry, ralph)$	*modus ponens* (15, 12)

Clausal Form

- Are we brilliant, or did we just get lucky?
- If we want to automate this process, we need a restricted form of the language
  - Eliminate some connectives
  - Uses others in a restricted manner
  - Reduce scope of negation
  - Eliminate quantifiers
- We will convert to **clausal form**
- We need to a procedure to do this conversion so that the set of converted formulas are semantically equivalent to the set of original formulas

Conversion into Clausal Form (1)

- Eliminate implication
  - Use the equivalence: $(p \rightarrow q) \cong (\neg p \vee q)$
- Reduce scope of negation
  - Use the equivalences
    - $\neg\neg p \cong p$
    - $\neg\exists x.p(x) \cong \forall x.\neg p(x)$
    - $\neg\forall x.p(x) \cong \exists x.\neg p(x)$
    - $\neg(p \wedge q) \cong (\neg p \vee \neg q)$ [de Morgan's]
- Rename variables
  - $(\forall x.p(x) \wedge \forall x.q(x)) \cong (\forall x.p(x) \wedge \forall y.q(y))$
- Move quantifiers left without changing order
  - The formula is now is **prenex normal form**

Conversion into Clausal Form (2)

- Eliminate existential quantifiers by **skolemisation**
  - $\exists x.\forall y.faster(x, y)$ gives $\forall y.faster(c, y)$
  - An interpretation that satisfies the original formula true also satisfies the skolemised formula, and vice versa
  - If the order of quantifiers is reversed, then skolem functions (rather than skolem constants) are required
- Drop all universal quantifiers
- Convert to conjunctive normal form
- Make each conjunction a separate clause
- Rename variables
  - Give variables with the same name but in different clauses different names

Example

- From the original *harry-ralph* example: 4th formula
  - $\forall x.\forall y.\forall z.(faster(x,y) \wedge faster(y,z)) \rightarrow faster(x,z)$
  - $\forall x.\forall y.\forall z.\neg(faster(x,y) \wedge faster(y,z)) \vee faster(x,z)$
  - $\forall x.\forall y.\forall z.\neg faster(x,y) \vee \neg faster(y,z) \vee faster(x,z)$
  - $\neg faster(x,y) \vee \neg faster(y,z) \vee faster(x,z)$

- Second formula
  - $\exists x.greyhound(x) \wedge \forall y.rabbit(y) \rightarrow faster(x,y)$
  - $\exists x.greyhound(x) \wedge \forall y.\neg rabbit(y) \vee faster(x,y)$
  - $greyhound(c) \wedge \forall y.\neg rabbit(y) \vee faster(c,y)$
  - $greyhound(c)$
    $\neg rabbit(y) \vee faster(c,y)$
  - $greyhound(c)$
    $\neg rabbit(y2) \vee faster(c,y2)$

"Here's One I Made Earlier"

- Clausal form of the *harry-ralph* example

$$\neg horse(x1) \vee \neg dog(y1) \vee faster(x1, y1)$$
$$greyhound(c)$$
$$\neg rabbit(y2) \vee faster(c, y2)$$
$$\neg greyhound(x3) \vee dog(x3)$$
$$\neg faster(x4, y4) \vee \neg faster(y4, z4) \vee faster(x4, z4)$$
$$horse(harry)$$
$$rabbit(ralph)$$

- This is a set of **horn clauses**
  - A horn clause is a disjunction of literals with only one positive literal
- We use a set of horn clauses to do **proof by negation**
  - If adding the negation of a fact to a consistent set of assertions makes it inconsistent, then the original fact must be true
  - So, start from the negation of what we want to prove and derive a contradiction ($\perp$)
  - We'll need an inference rule

Resolution

- In essence

$$\frac{p \vee q \quad \neg p \vee r}{q \vee r}$$

- The general case (of a horn clause)

$$\frac{\neg q_1 \vee \neg q_2 \vee \ldots \vee \neg q_m \vee p_i \quad \neg p_1 \vee \neg p_2 \vee \ldots \vee \neg p_{i-1} \vee \neg p_i \vee \neg p_{i+1} \vee \ldots \vee \neg p_m}{\neg p_1 \vee \neg p_2 \vee \ldots \vee \neg p_{i-1} \vee (\neg q_1 \vee \neg q_2 \vee \ldots \vee \neg q_m) \vee \neg p_{i+1} \vee \ldots \vee \neg p_m}$$

- The base case

$$\frac{p \quad \neg p}{\bot}$$

- If only we can work out what to do resolving formulas with variables

Unification

- What is the resolution of $cat(X) \vee mouse(X)$ and $\neg cat(jerry)$?
  - The answer is $mouse(X)$, as we'd expect; but only if $X = jerry$
- Unification
  - Unification is a process of attempting to identify two symbolic expressions by the matching of terms and the replacement of certain sub-expressions (variables) by other expressions
    - The **unification problem** for two terms is finding a way of replacing their variables with other terms such that the two resulting terms are (syntactically) equal
    - Different occurrences of the same variable must always be replaced by the same term
      - $p(a, X)$ and $p(X, b)$ are not going to unify

- Unification as decision problem
  - Not just a matter of saying "yes" or "no"
  - If two terms are unifiable, constructing a solution is also desirable
  - The solution to a unification problem is a **substitution** that identifies the two terms
  - In general, there may be infinitely many unifiers, but in practice it is sufficient to consider the **most general unifier**

- A unification algorithm, in general
  - Should decided the solvability of a unification problem
  - If it is solvable, it should also compute the most general unifier

- It turns out that there are efficient algorithms for this

A Unification Algorithm

- An algorithm for syntactic unification for two terms S and T
  - If S and T are constants, then they unify if and only if they are identical
  - if S is a variable and T is any term, then they unify with the substitution $\{S \mapsto T\}$
    - This includes the case that T is also a variable
  - if both S and T are compound terms, then they unify if and only if:
    - They have the same functor name
    - They are of the same arity
    - All of their pairwise corresponding arguments unify

- The **occurs check**
  - What is the unifier of X and $p(X)$?

Proof (2): By Contradiction

- To prove $faster(harry, ralph)$, assume the negation, show inconsistency, infer the original

	$\neg horse(x1) \vee \neg dog(y1) \vee faster(x1, y1)$		p1
	$greyhound(c)$		p2
	$\neg rabbit(y2) \vee faster(c, y2)$		p3
	$\neg greyhound(x3) \vee dog(x3)$		p4
	$\neg faster(x4, y4) \vee \neg faster(y4, z4) \vee faster(x4, z4)$		p5
	$horse(harry)$		p6
	$rabbit(ralph)$		p7
	$\neg faster(harry, ralph)$		¬conc
1	$\neg faster(harry, y4) \vee \neg faster(y4, ralph)$	$\{x4 \mapsto harry\ z4 \mapsto ralph\}$	(¬conc, p5)
2	$\neg horse(harry) \vee \neg dog(y1) \vee \neg faster(y1, ralph)$	$\{x1 \mapsto harry, y1 \mapsto y4\}$	(1, p1)
3	$\neg dog(y1) \vee \neg faster(y1, ralph)$		(2, p6)
4	$\neg greyhound(x3) \vee \neg faster(x3, ralph)$	$\{x3 \mapsto y1\}$	(3, p4)
5	$\neg faster(c, ralph)$	$\{x3 \mapsto c\}$	(4, p2)
6	$\neg rabbit(ralph)$	$\{y2 \mapsto ralph\}$	(5, p3)
7	$\bot$		(6, p7)

Proof by Negation, How To

- Are we still brilliant, or did we just get lucky again?
  - There were multiple rules that could have been used in any resolution
  - How do we know which one to use?
  - We don't ...
  - ... but we need to try them all
  - So we need to be systematic about it
  - In fact, we need an algorithm

Order of Computation

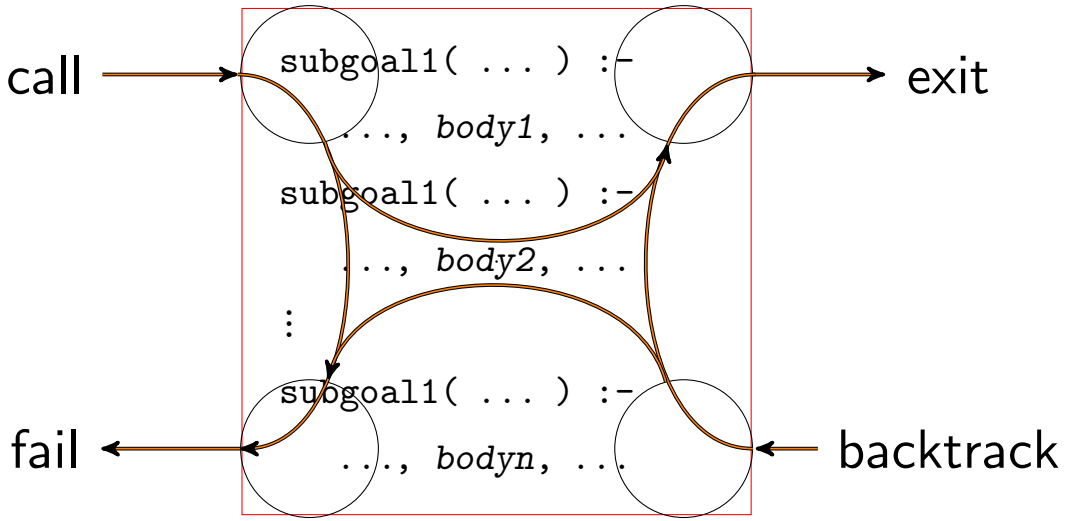
- The query (goal) is a disjunction of negated literals, each disjunct being a sub-goal
- Try to satisfy the sub-goals, from left to right, returning yes with any variable instantiation
- To satisfy a subgoal, check through the clauses looking for one with a positive literal with the same functor and arity
  - If none are found, fail the sub-goal and start backtracking
  - If one is found, try to unify the sub-goal with the literal
    - Any values assigned to variable by substitution computed by the unification process are transmitted to sub-goals in the rest of the clause, and other sub-goals in the query
    - The rest of the clause replace the original disjunct in the query
  - If the unification fails, re-start the search through the clauses
  - If no unification succeeds, fail the sub-goal and start backtracking
- If the query is empty, then the goal succeeds

So Here's a Thing

- The following are equivalent
  - $\neg b_1 \vee \neg b_2 \vee \ldots \vee \neg b_n \vee h$
  - $\neg(b_1 \wedge b_2 \wedge \ldots \wedge b_n) \vee h$
  - $(b_1 \wedge b_2 \wedge \ldots \wedge b_n) \rightarrow h$
- So we could write it
  - `h :- b_1, b_2, ..., b_n.`
- A Prolog program is a set of horn clauses

Order of Computation (2)

- Suppose the query is
  `?- subgoal1(...), subgoal2(...), ..., subgoalm(...).`
  - Call subgoal1(...)
  - If the call unifies with the head of the ith clause ($1 \leq i \leq n$), exit
  - If a call in the body, or some subsequent call, fails, backtrack and resume looking for matches at clause $i + 1$
  - If no unifier (or no more unifiers) with a head are found, fail



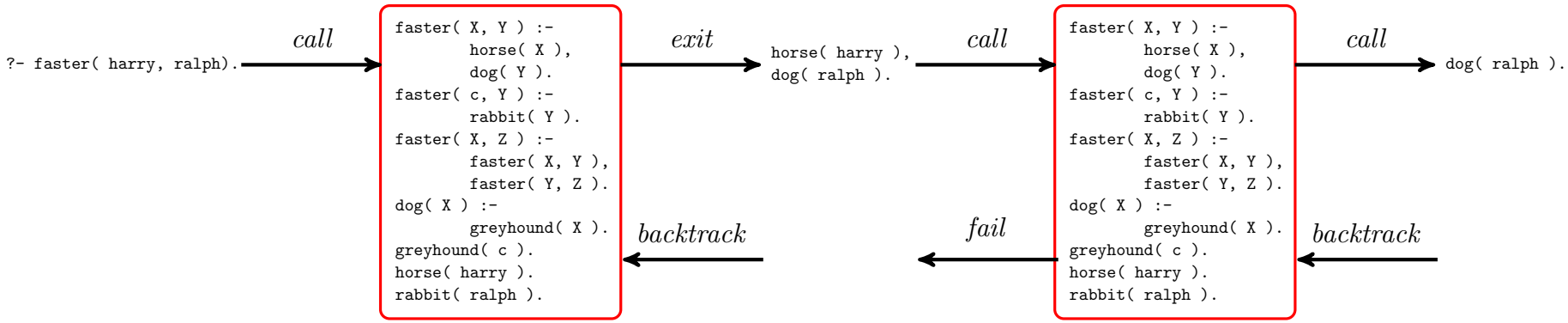
Proof (3): By Prolog Program

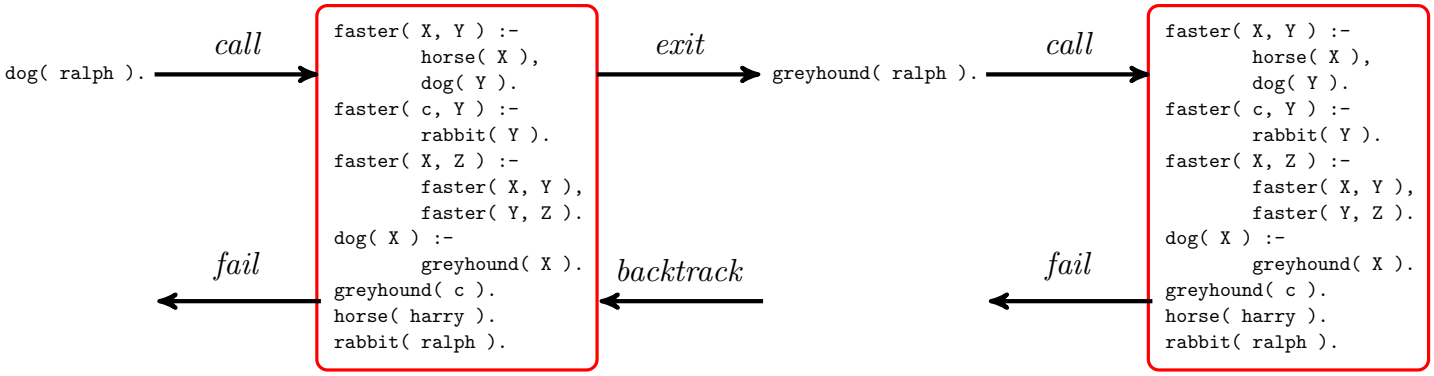
- Converting all the clauses to Prolog, gives:

```
– faster( X, Y ) :-
        horse( X ),
        dog( Y ).

  faster( c, Y ) :-
        rabbit( Y ).

  faster( X, Z ) :-
        faster( X, Y ),
        faster( Y, Z ).

  dog( X ) :-
        greyhound( X ).

  greyhound( c ).
  horse( harry ).
  rabbit( ralph ).
```

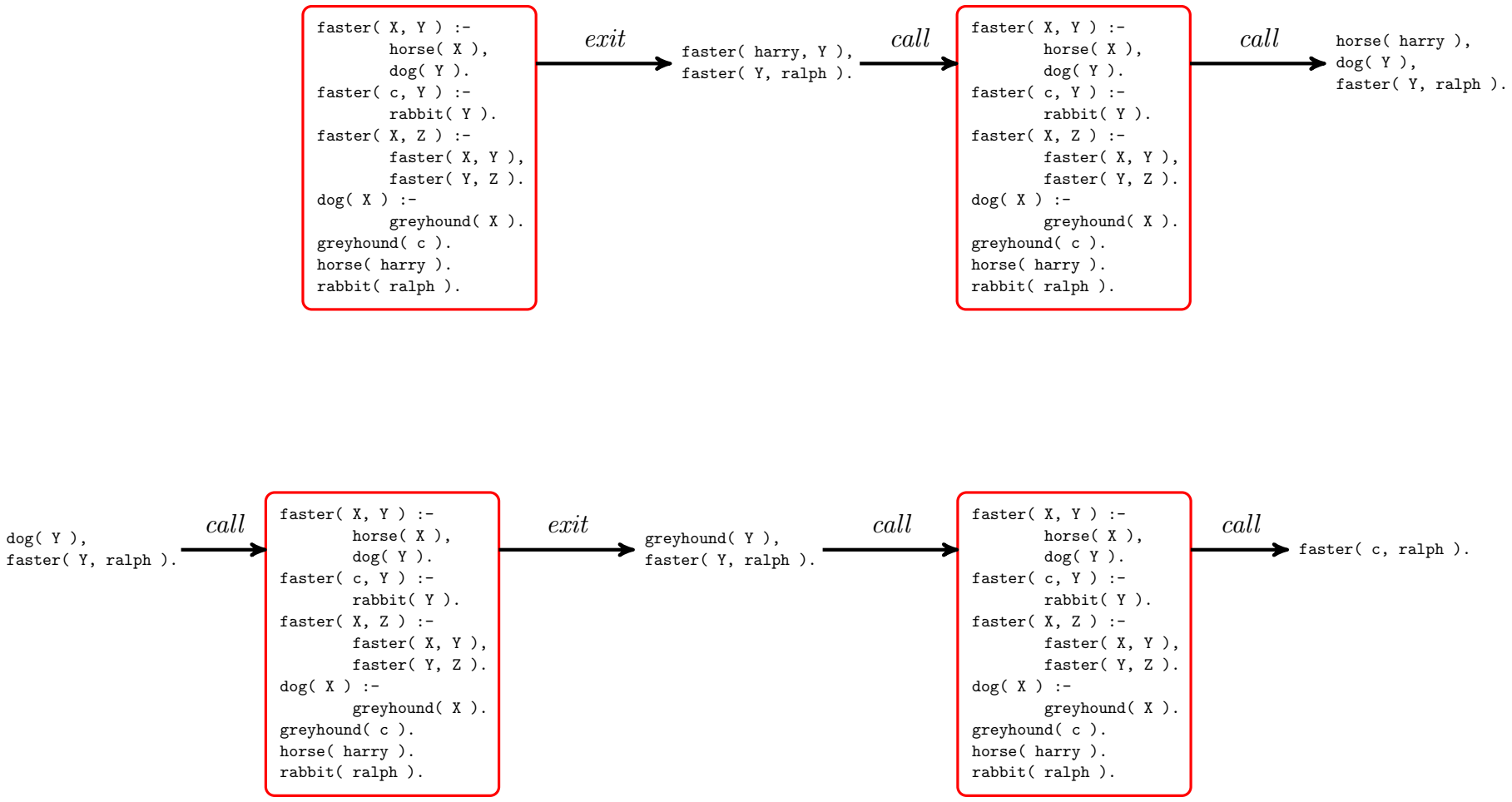
- Query: `?- faster( harry, ralph ).`

What Happens Next

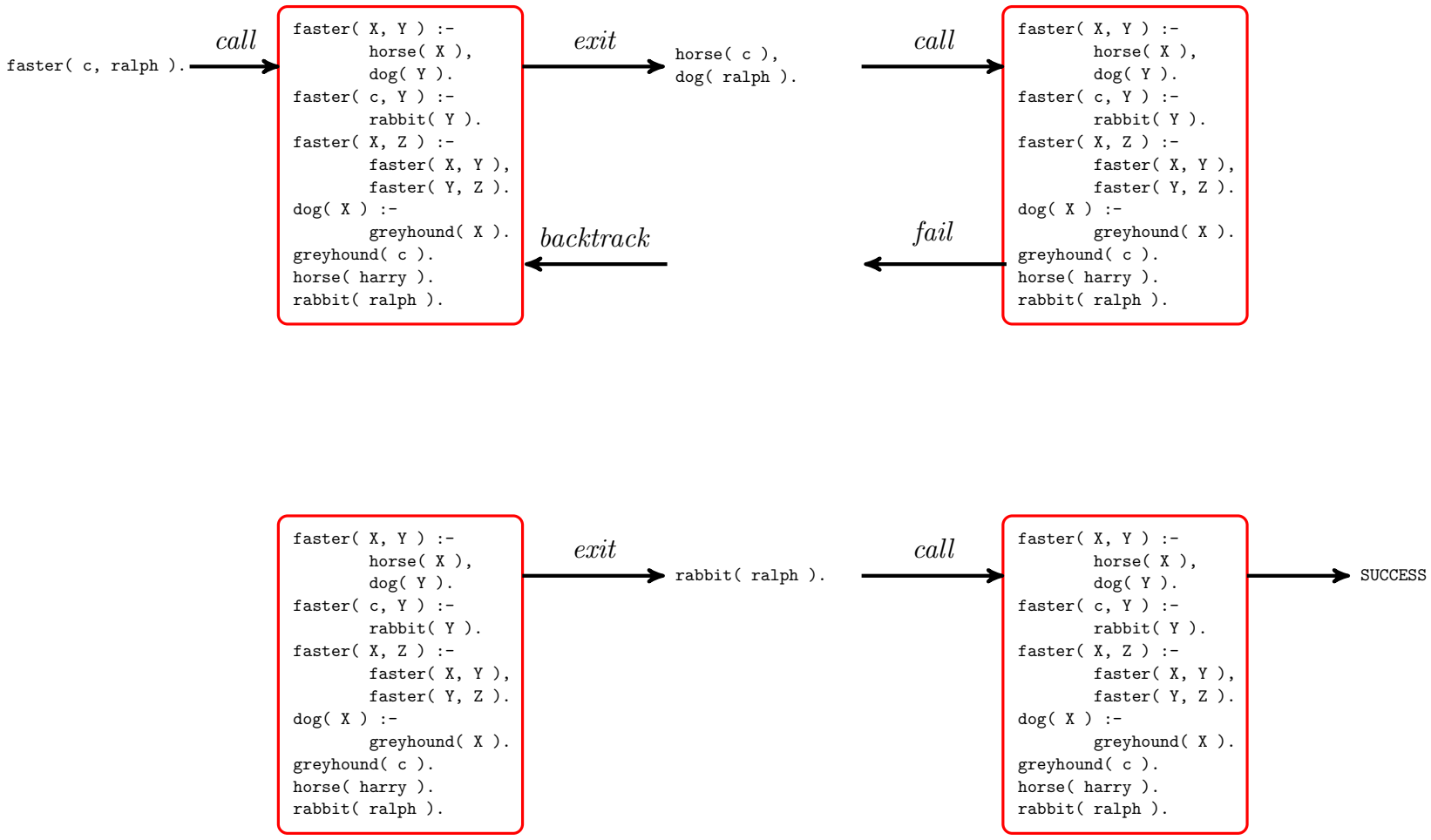




What Happens Next



What Happens Next



Comments

- Top-down (matching clauses), left-to-right (satisfying sub-goals) is depth-first search

- What happens if we put the third clause of faster, first

  – Prolog's proof procedure is **sound**, if Prolog proves something, then it is an entailment $\vdash \rightarrow \models$
  – It's just not **complete**, if there is a valid entailment, Prolog might not prove it $\models \rightarrow \vdash$

- Sometimes we want sound and complete automated reasoning

  – So we'll use Prolog to implement a sound and complete proof system

Summary

- The process of knowledge engineering seeks to acquire, represent and reason with knowledge about a domain of discourse. It is difficult, time-consuming and pains-taking activity, but if not done well and done right, all the logical inference in the world will not get you an 'intelligent' or 'expert' system

- There are many different knowledge representation languages, most of them reducible in expressiveness to either predicate or propositional logic. Logics are particularly useful for directly representing knowledge because of the possibility of automation

- Representing knowledge declaratively affords a reasonably straightforward Prolog implementation

Appendix A: Some Arguments

- Try these
  - Leaded petrol is bad for the environment because of lead emissions
  - Unleaded petrol produces less lead emissions
  - Less lead emissions is better for the environment
  - $\therefore$ Unleaded petrol is better for the environment

  - If Britain does not sell land mines, another country would
  - British land mines are better made, so are safer than other countries'
  - Other countries have not agreed to stop selling land mines
  - $\therefore$ It is safer for Britain to sell land mines

  - If I am paranoid, and they are out to get me, then I win
  - If I am paranoid, and they are not out to get me, then I don't lose
  - If I am not paranoid, and they are not out to get me, then I don't win
  - If I am not paranoid, and they are out to get me, then I lose
  - $\therefore$ It is better to be paranoid [with apologies to Pascal]