Reasoning in Propositional Logic

Prof. Mauro Gaspari
Dipartimento di Informatica Scienze e Ingegneria (DISI)
mauro.gaspari@unibo.it

Reasoning in PL

- **Semantic methods (model checking)** for checking logical consequences of a KB have the merit of being conceptually simple; they directly manipulate assignments.
- Unfortunately, the number of assignments grows exponentially with the number of atomic formulas in a KB.
- If a KB is large, the number of assignments may be impossible to manipulate.
- **PSAT** (Propositional Satisfiability) is (co)NP-complete.
- No existing worst-case-polynomial algorithm.

The aim of this lesson is to emphatise PL as a reasoning tool for KR. We want to check if a given formula is a logical consequence of a KB. There's the semantic method but too costly. By "reasoning sys" we mean a method to do inference which exploits the SYNTAX.

We will see that syntax based methods are more efficient than semantic methods.

We study logic mainly because we want to use it for:
- planning
- derive new knowledge from a KB
- use it as a tool for problem solving
-...

All these things can be done thanks to the concept of LOGICAL CONSEQUENCE (This regards both Prop. L. and FOL).
The logical consequence can be of two types:

1) semantic : KB |= F, which means that in all possible assignments in which KB is true, also F is true.
(the symbol |= is called "entailment", remember that when you see it you must deal with assignments -> truth tables are involved.
(about this we've seen the deduction theorem and the refutation theorem to solve the same problem in a slightly different way)

2) syntactic: KB |- F, which means that F is DERIVABLE from KB thanks to inference rules which exploits only the syntax. (the symbol |- refers to "derivation"/"inference", remember that when you see it you must use methods like Natural Deduction, Resolution....

There's a theorem which actually tells us that semantic and syntactic logical consequence are equivalent! This theorem is called the "completeness theorem". So thanks to this we can choose in which way to find if F is logical consequence of KB. In general these methods are called "inference ENGINES". But we'll see that semantic methods are less efficient (in FOL actually are almost useless because you have infinite possible interpretations of a formula, so syntactic methods are for sure more useful).

Notation

KB= knowledge base = program (in logic programming languages) = set of formulas

KR = knowledge representation

engine= method to find the logical consequence of a formula F from the KB, (implementable as an automatic computation).

ND= natural deduction. (which is an example of engine, as Resolution).

Don’t be confused: there are also engines which are based on FOL. Here he's saying in a confusing way something obvious: engines for Prop. logic are based on Prop. logic syntax.

Reasoning in PL

- Reasoning is the formal manipulation of the wffs in the KB to produce new ones.
- Reasoning is a syntax based mechanical process, if one or more wffs in the KB matches a certain syntax an inference can be activated.
- Several engines are available for propositional logic all of them use the syntax of wffs in propositional logic.
- We write KB $\vdash^{E}$ F

  to denote that the formula F can be deduced by the KB using engine E.

Reasoning in PL

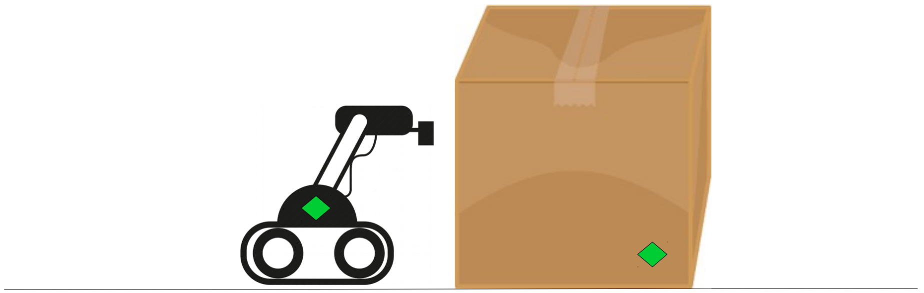
Knowledge Base (KB)

$batteryOK$

$moves$

$moves \land batteryOK \rightarrow LightW$

We have robot with a battery and the box has a sensor, to know whether it moves.



Natural deduction engine

- Rules for introducing or eliminating connectives, for example:

$$\frac{A \quad B}{A \wedge B}\ (\wedge i) \qquad \frac{A \wedge B}{A}\ (\wedge e1) \qquad \frac{A \wedge B}{B}\ (\wedge e2)$$

Modus Ponens: $$\frac{A \rightarrow B \quad A}{B}\ (\rightarrow e)$$

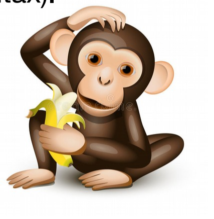
Example: $$\cfrac{\cfrac{[A \wedge B]}{A}\ (\wedge e1)}{A \wedge B \rightarrow A}\ (\rightarrow i)$$

$$\frac{\begin{array}{c}[A]\\ \vdots \\ B\end{array}}{A \rightarrow B}\ (\rightarrow i)$$

With Gabrielli we've seen ND only for prop. logic and Resolution only for FOL. Actually ND and Resolution are methods which belong to both prop. logic and FOL (with some differences, because in FOL we need to do unification)

Natural Deduction (ND) is an example of really basic engine (method to do reasoning = inferences using syntax).
Let's see an application of it. the formula we want to DERIVE from the KB is "LightW".
(aim: KB |- LightW with Natural Deduction)

Natural Deduction



BatteryOK (S)
Moves   (T)
Moves ∧ BatteryOK → LightW

Thanks to the "and introduction" inference rule you can add to the KB the formula " moves and BatteryOK"

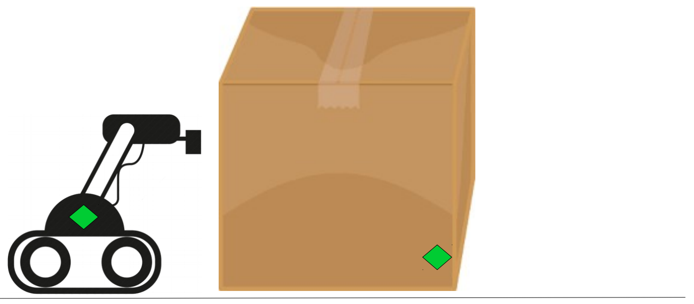
$\xrightarrow{\wedge i}$

BatteryOK
Moves
Moves ∧ BatteryOK → LightW (S → T)
Moves ∧ BatteryOK   (S)

Thanks to the "implication elimination" rule we can add to the KB the formula LightW.

$\xrightarrow{\to e}$

BatteryOK
Moves
Moves ∧ BatteryOK → LightW
Moves ∧ BatteryOK
LightW ⚙



There exist also more rules, also for the other connectives, but we don't go so deep because Prop. Logic is not much used actually. This is just an example of how an engine can work.

Note that there are two choice points:
- which rule should we apply at each step?
- to which formulas of the KB to apply it? (= which formula should we use as premises)

So we can have different deductions/"paths"

This is useful for what we'll talk later:
Imagine to have a KB (= set of formulas).
Imagine that it is possible to apply not only one inference rule, but many (and also that for the inference rules appliable you could choose many formulas as premises). So which one do you apply?
There's not a general rule. This problem can be seen as a tree search, where each inference rule (applied to some formulas as premises) defines a branch. You could obtain the formula you are looking for with different sequences of inference rules. Indeed each path can be thought as a derivation. There are derivations which are succesful (the leaf node is a KB which contains the formula you were looking for) or not succesful (the leaf node is a KB to which you can't apply any inference rule to obtain new formulas). So as we know for tree searches we can use heuristics. This will be discussed later.

Note that also in this case if you applied a semantic method (which tries all the assignments) it would take more time, instead with natural deduction (so a reasoning method) it's more efficient.

Natural Deduction

- Soundness: IF $KB \vdash^{ND} F$ THEN $KB \models F$
- Completeness: IF $KB \models F$ THEN $KB \vdash^{ND} F$
- Decidable.
- Efficient: NO
- Expressive Power: POOR

Natural deduction is **sound** and also **complete**.

Efficiency IS THE MAIN PROBLEM

Is not clear which formulas must be applied to the rules, so the branching factor is big, and there are not good heuristics which tell us which rules to apply and/or to which formulas.

Some precisations:

1) Natural Deduction is complete only if you use all the inference rules. For example with Gabrielli we used only a few inference rules (among all the ones belonging to ND) to solve exercises, and I am not sure that only with them completeness is guaranteed. I asked it to Gaspari and he said that if you don't use the inference rule "modus ponens" then it isn't a complete engine.

2) In literature there's a little bit of confusion about the terms "soundness" and "completeness". This is the resume of what I've understood:
- "soundness" = "correctness", is used as an attribute for engines. I means that if the engine gives you an answer, for sure that's correct.
How do you find if the answer is correct? If we're talking about a semantic based engine, then the correctness of the answer is checked in the other world: with a syntactic engine. And viceversa.
The confusing point is that there's a theorem called "soundness theorem" which in particular refers to this case:
if you're SYNTACTIC based engine gives you an answer, then it also must be correct in the world of semantic (which in other words means: the engine is sound if what you derive can also be entailed).
MANY PEOPLE WHEN TALK ABOUT SOUNDNESS REFER TO THE SOUNDNESS THEOREM (which simply means that they're implicitly talking about a syntactic based engine).

- there's exactly the same problem with the term "completeness", plus these two precisations:
    - the completeness theorem tells us that something which is a semantic logical consequence must be also a syntactic logical consequence, AND VICEVERSA (so anyway I'm sure that what is written in the slide is wrong/incomplete). So this theorem gives us a powerful notion, but is also used as an attribute for engines as in the case of soundness. So the point is: ideally anything is semantic log. cons. must be also syntactic log. cons. and viceversa, and an engine which is able to guarantee that is called complete.

How to improve efficiency?

- Problems of natural deduction based engines:
  - Many inference rules.

    Which means high number of choices -> high branching factor in the tree search.
  - Lack of effective heuristics for selecting them.
  - The connection between the reasoning process and the GOAL is not clear.

    Don't worry, the meaning of this sentence is more clear when you see Resolution ("+ refutation"). (slide 15)
- How we can approach these problems:
  - Making the syntax more simple ==> this reduces the number of inference rules.

    More syntax => more inference rules needed. So we could use less connectives.
  - Develloping effective heuristics.

    You can think about the reasoning process as a tree search, and we want to obtain a leave which is out goal.
    So what we want to try to do is to use the goal to do the search.
  - Use the GOAL to guide search.

Making syntax more simple

Here we're listing some examples of how syntax can be done more simple -> to reduce the inference rules needed -> to improve the Efficiency of ND.

- Remove implications: B→A ≡ ¬BvA
- Bring negation close to atomic formulas (DE MORGAN).
- Exploit distribution.

- A **literal** is an atomic formula or a negated atomic formula.
- A **clause** is a set of literals which is assumed (implicitly) to be a disjunction of those literals:
- **Unit clause**: clause with only $\neg p \vee q \vee \neg r \quad \leftrightsquigarrow \quad \{\neg p, q, \neg r\}$
- **Conjunctive Normal Form (CNF)** of a formula: Implicit conjunction of clauses:

$$p \wedge (\neg p \vee q \vee \neg r) \wedge (\neg q \vee q \vee \neg r) \wedge (\neg q \vee p)$$
$$\Updownarrow$$
$$\{\{p\}, \{\neg p, q, \neg r\}, \{\neg q, q, \neg r\}, \{\neg q, p\}\}$$

Note that when we define a KB it is a list of formulas: actually you should consider as if between them there's an implicit "and".

Note that "," is replaced by "and" outside the cause. Instead within the clause it is replaced with "or".

- A KB is a set of clauses.

Don't be confused: Actually in prolog the "," always means "and". So I don't understand why Gaspari introduced this notation which is also not used in his next slides.

Example

- Transform the formula
  - $(p \rightarrow q) \rightarrow (\neg q \rightarrow \neg p)$

  into an equivalent formula in CNF
- Solution:

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

The general steps are always in this order:

1) remove implications and equivalences
2) apply DeMorgan
3) apply distributivity until you find the form you want.

Resolution Inference Rule

This is another "engine". It has only one inference rule, called "Resolution Rule".

"l^c" means complement literal (= the negation) of the literal "l".

- Suppose C_1, C_2 are clauses such that l in C_1, l^c in C_2. The clauses C_1 and C_2 are said to be **clashing clauses** and they clash on the complementary literals l, l^c
- C, the resolvent of C_1, C_2 is the clause

$$Res(C_1, C_2) = (C_1 - \{l\}) \cup (C_2 - \{l^c\})$$

- C_1 and C_2 are called the **parent clauses** of C.

Don't worry: if you don't understand this slide is just because it's explained in the worst way. Looking the next two slides will be clear clear. Just remember the meaning of clashing clauses because it will be used (in the examples he indicates with C1 and C2 the formulas of the KB used as premises of the resolution inference rule at each step).

$$\begin{tikzcd}
C_1 \arrow[dr, dash] & & C_2 \arrow[dl, dash] \\
& C = (C_1 - \{l\}) \cup (C_2 - \{l^c\}) &
\end{tikzcd}$$

Precisations:

RESOLUTION is a term used for two different things:
- the Resolution (inference) RULE, which is the rule with the premises and the conclusion.

- the Resolution procedure /method/engine, which only uses one rule: the Resolution (inference) rule.

So you could think that they're the same thing, in fact they're not:
the resolution procedure has a specific way of working: if you want to derive a formula F, then you must put in your KB the negation of F; then apply the inference rule, each time adding the conclusion to the KB, until you find the empty clause/a contraddiction/ the formula false.

this is what you find in any book. But Gaspari decided to be different from all the others:
He doesn't make a difference between the resolution rule and the resolution procedure. And he calles (in the next slides) Refutation+Resolution what is actually normally known as resolution procedure (or just Resolution)

Resolution

$$\frac{\{B, A_1, \ldots, A_m\} \quad \{\neg B, C_1, \ldots, C_n\}}{\{A_1, \ldots, A_m, C_1, \ldots, C_n\}}$$

$$\frac{B \vee A \quad \neg B \vee C}{A \vee C}$$

[curiosity]
Resolution is actually a generalization of the modus ponens rule (of ND): to see it remove A form the left premise and transform "not(B) or C" in "B->C".

- Resolution it is sound: note that B must be either false or true.
  - if B is false, then $B \vee A$ is equivalent to A, so we get $A \vee C$
  - if B is true, then $\neg B \vee C$ is equivalent to C, so we get $A \vee C$

This is the semantic EXPLANATION of the resolution rule. So this is just to understand what does it mean BUT WHEN IT IS APPLIED IT DOESN'T MATTER THE MEANING, IT IS USED ONLY AS SYNTACTIC RULE.
(Is the same of ND, when you do the exercises you can apply blindly the inference rules you withouth thinking about their semantic meaning).

Again the goal is to derive "LightW".
This time applying the resolution rule.

The clashing clauses here are the second and third clauses, because of the literal "moves".
Applying the resolution rule we obtain "not(BatteryOK..". And so on.

Resolution

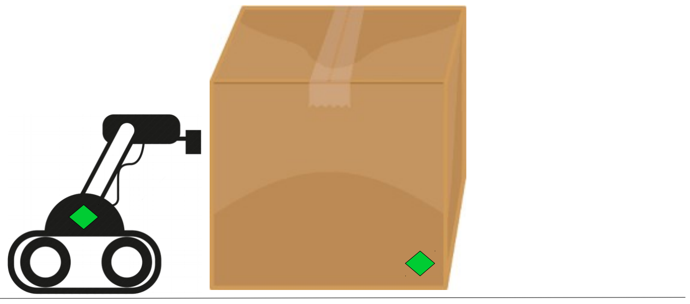
BatteryOK
Moves (C1)
¬Moves v ¬BatteryOK v LightW (C2)

→

BatteryOK (C1)
Moves
¬Moves v ¬BatteryOK v LightW
¬BatteryOK v LightW (C2)

→

BatteryOK
Moves
¬Moves v ¬BatteryOK v LightW
¬BatteryOK v LightW
LightW ⚙



Resolution

It is sound BUT it is not complete:
for example also if (P or Q) is semantic consequence of (P and Q) , we have no rule to obtain it with Resolution.

- Soundness: IF $KB \vdash^R F$ THEN $KB \models F$
- Completeness: NO ($P \wedge Q \models P \vee Q$ BUT $P \wedge Q \nvdash^R P \vee Q$)
- Decidable.
- Efficient: NO

  It is better than ND about efficiency (because we use only one inference rule, so we reduce the coiche points: now we only have to choose to which formulas to apply it.), **but we loose completeness**.
- Expressive Power: POOR

Refutation & Resolution

Thanks to refutation we can obtain completeness. The idea of the refutation method is this:
the goal **NEGATED** is put in the KB, therefore the derivation is succesful if we obtain a contraddiction/ false/empty clause as conclusion.
(the little empty square means "empty clause")

- Use the GOAL for search.

Note that in this way we're exploiting the goal for the search, which was much more difficult in ND.
Now is more clear the meaning of the fourth line in the slide 8.

- $\square$ ==> Unsatisfiable
- Refutation idea: $KB \cup \{\neg G\} \models \square$ IFF $KB \models G$

$$\frac{\neg F \quad F}{\square}$$

- Ground resolution theorem
- Resolution: perform refutations until the empy clause is produced.

Refutation & Resolution

The goal is LightW. So we put it negated in the KB. (this is what the professor means with (U not(Goal)).
Note that we only have to apply the resolution inference rule, so this reduces the number of choice points wrt to ND. With C1 and C2 are indicated the clauses ("clashing clauses") to which is applied the rule at each step.

¬LightW (∪ ¬GOAL) (C1)
BatteryOK
Moves
¬Moves v ¬BatteryOK v LightW (C2)

¬LightW
BatteryOK (C1)
Moves
¬Moves v ¬BatteryOK v LightW
¬Moves v ¬BatteryOK (C2)

¬LightW
BatteryOK
Moves (C1)
¬Moves v ¬BatteryOK v LightW
¬Moves v ¬BatteryOK
¬Moves (C2)

¬LightW
BatteryOK
Moves
¬Moves v ¬BatteryOK v LightW
¬BatteryOK v LightW
¬Moves
□

There's an error: "not(moves) or not(BatteryOK)" has been wrongly replaced by "not(BatteryOK) or LightW"

Refutation & Resolution

- Soundness: IF $KB \cup \{\neg F\} \vdash^{RR} \Box$ THEN $KB \models F$

We call Refutation+ Resolution = "RR". RR is sound and complete.

- Completeness: IF $KB \models F$ THEN $KB \cup \{\neg F\} \vdash^{RR} \Box$
- Decidable.
- Efficient: NO

We still have not a good efficientcy at this point (also if it's better than ND).

But we can improve it with some heuristic. Let's see the most common ones.

- Expressive Power: POOR

The resolution tree is the tree generated by all the possible application of the resolution rule to the KB.
We need some strategies to efficiently search on it:

Ordering strategies: estabilish which clauses should be used to apply the rule first.

- breadth first: consider first all the possibilities at a certain tree level.
(given a KB the number of clauses is finite, so this method is complete, but it takes too much memory).

In both case the resolution tree is finite in PL.

- depth first: another possible strategy, going in depth: you use the conclusion as premise in the next step.

- Unit preference: we choose first as premises the unit clauses. In the example showed before we applied this. when you have no unit clauses you can have a second strategy to be applied.

Here during the lesson he was refering to the tree search of resolution. I think this refers to enignes in general. But with natural deduction we would need not only these heuristic which refers to which formula use as premises, but also which inference rule to apply (with resolution we don't have this problem because the rule is only one).

Develloping heuristics

- Ordering strategies:
  - Breadth first
  - Depth first (deep bound + backtracking)
  - Unit preference: prefers resolutions which involves a unit clause.

From this book: "Knowledge Rep. and Reasoning"(R.J BRACHMAN, H.J. LEVESQUE) :
"The best strategy found to date (but not the only one) is unit preference, that is, to use unit clauses first. This is because using a unit clause together with a clause of length k always produces a clause of lengthk−1. By going for shorter and shorter clauses, the hope is to arrive at the empty clause more quickly. "

There's a difference with ordering strategies:
with ordering strategies all the possible resolutions are complete INSTEAD WITH THESE HEURISTIC ONLY SOME KIND OF RESOLUIONS ARE ALLOWED, SO SOME OF THESE HEURISTIC MAKE THE ENGINE NOT COMPLETE.

I also asked him about what he meant by "kinds of resolutions", but he replied using the same words, so it was useless. I think that what he means is this:

WITH ORDERING STRATEGIES WE ONLY POSTPONE THE EXPLORATION OF SOME PATHS. WITH REFINEMENT STRATEGIES WE PRUNE THE TREE, SO IN SOME OF THIS KIND OF STRATEGIES IT COULD BE THAT WE REMOVE ALL THE PATHS IN WHICH WE COULD HAVE FOUND SUCCESFUL DERIVATIONS -> THE ENGINE IS NOT COMPLETE ANYMORE. (THE ADVANTAGE IS THAT THEY ARE FASTER).

I also asked him about what he meant by "kinds of resolutions", but he replied using the same words, so it was useless. I think that what he means is this:

WITH ORDERING STRATEGIES WE ONLY POSTPONE THE EXPLORATION OF SOME PATHS. WITH REFINEMENT STRATEGIES WE PRUNE THE TREE, SO IN SOME OF THIS KIND OF STRATEGIES IT COULD BE THAT WE REMOVE ALL THE PATHS IN WHICH WE COULD HAVE FOUND SUCCESFUL DERIVATIONS -> THE ENGINE IS NOT COMPLETE ANYMORE. (THE ADVANTAGE IS THAT THEY ARE FASTER)

Develloping heuristics

He only read this part, adding almost nothing, and did only the example for the set of supports, under request. I think it's not cristal clear so at the beginning of the next lesson I'll ask him some examples.

- Refinement strategies: they permit only certain kinds of resolutions (some if them are complete):
  - **Set of support:** clauses that comes from the negation of the goal G or descendents of those clauses. [COMPLETE]
  - **Linear input:** at least one of the clauses being resolved is a member of the original set of clauses, including those originating from the goal G. [NOT COMPLETE]
  - **Ancestry filtering:** one of the clauses is a member of the original set or is an ancestor of the other clause being resolved. [COMPLETE]

What does this **linear impnout definition** means? It means that at each resolution step we have to take only one of the clauses of the initial set. So, more precisely, to do resolution we need just two causes, and in doing resolution at least one of the two clauses should belong to the initial set. The initial set is the initial knowledge base + the negated goal. So in the resolution we are not be able to use two clauses that we will generate in the next steps, **one clause must belong to the initial set**.

This sentence means that the ancestor was considered in resolving one of the clause we are using. So one clause should be known from the original set and with the other clauses we have generated during deduction we can do something more: during the deduction we can use one of the clauses the is an ancestor of the clause we are going to resolve.

Discussion

- Resolution and resolution strategies have a positive impact on the reasoning process:
    - Just one inference rule.
    - Using the goal for reasoning.
    - Heuristics for selecting clauses to resolve.
- Unfortunately this is not enough for reducing complexity: the KR system based on Propositional Logic remains NP-complete in the worst case.

The sequence of rules used in the derivation process is a kind of explanation.

Discussion

If we have to choose between semantic based sys or syntactic based, the syntactic methods are more efficient. Then between ND and Resolution Resolution is more efficient. ("Resolution" + refutation is also complete). Resolution used with some heuristic is more efficient.

- However, the size of the resolution tree is in practice orders of magnitudes smaller than corresponding truth-tables, and for finding the empty clause not the whole tree needs to be traversed.
- Only one path of the tree needs to be kept in the memory at a time, which reduces the memory requirements substantially.
- This enables the use of resolution in solving important problems in AI.

Refutation + resolution is not the on,y method we can use to prove satisfability. We can use aslo tree search methods.

- Additionally, **the provided proof can be used as explanation**.

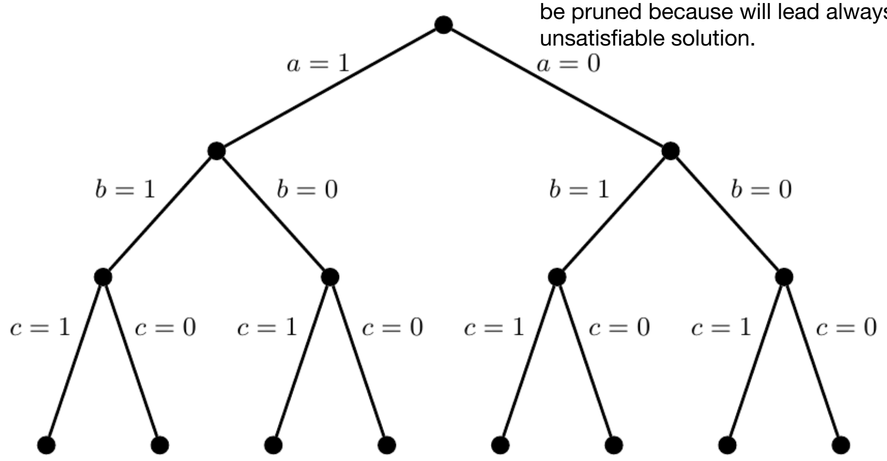
“The sequence of rules used in the derivation process is a kind of explanation" : It can be simply thought as an explanation of why F is logical consequence of KB, or more usefully it can be used to do planning, as we did in fundamentals of AI.

He stopped here then he said: next time I'll finish this part, then we'll do a short introduction to LISP.

Other methods and tools

- **Tree-Search Algorithms** are semantic based methods which do not explicitly go through all possible valuations of atomic formulas and do not store all of them in a big table.
  - The simplest algorithm does a **depth first traversal of a binary tree** that represents all valuations (still exponential).

If **a** is an atomic formula, the branch with a = 0 will be pruned because will lead always to unsatisfiable solution.



Other methods and tools

- The improvements are based on the observation that many of the partial valuations in the tree make the KB false and large parts of the search three can be pruned.
  - DPLL (Davis, Putnam, Logemann, Loveland) based on CNF (1960-1962).
  - + heuristics for selecting the decision variables: Conflict driven clause learning procedure Marques-Silva and Sakallah (1996).
  - Heuristic search in model space (sound but incomplete) e.g., min-conflicts-like (WalkSAT) or hill-climbing based algorithms (Russel and Norvig). These algorithms are not adapt to detect unsatisfiability.

So these other methods basically methods for exploring the tree search trying to describe branches where is possible and in many cases we are actually reach good performance. However these algorithms does not work well when the system **is not satisfiable** .

Question: in practical applications, why should we prefer a method based on semantic instead of syntax?
The advantage of the resolution based methods is that they tell you also the sequence of reasoning step you have done for reach the conclusion. And this is really useful for many applications like medical diagnosis or also decision support systems that suggest you to take a specific decision and explain also why the system suggested that decision.
On the other hand, the semantic based methods just try to find an assignment that works and do not tell you which is the reasoning process that brings to the conclusion, they just say that for this set of values your original KB is true and therefore your goal is true as well.

Discussion

- Again, the size of these trees is in practice orders of magnitudes smaller than corresponding truth-tables, and for finding one satisfying valuation not the whole tree needs to be traversed.
- Only one path of the tree, from the root node to the current leaf node, needs to be kept in the memory at a time, which reduces the memory requirements substantially.
- This enables the use of these algorithms for KBs of the size encountered in solving important problems in AI and computer science in general, with up to hundreds of thousands of atomic propositions and millions of clauses.

PL application

So thanks to that, we can actually use PL effectively for several applications in AI like planning, problem solving, intelligent control, diagnosis and so on.

- The use of PL has increased since the development of powerful algorithms and implementation methods in last two-three decades.
- Although checking satisfiability in the worst case is exponential, using the above algorithm, PL can be exploited in many practical applications.
- For example in several areas of Artificial Intelligence:
    - Planning
    - Problem-solving
    - Intelligent control
    - Diagnosis

LP applications

- And computer science in general:
    - In **hardware design**, propositional logic has long been used to minimize the number of gates in a circuit, and to show the equivalence of combinational circuits.
    - **State-space search:** the problem of testing whether a state in a transition system is reachable from one initial state.

KR example in PL

- **Chemical synthesis:** under suitable conditions, the following chemical reactions are possible:

$$\mathrm{HCl} + \mathrm{NaOH} \rightarrow \mathrm{NaCl} + \mathrm{H_2O}$$
$$\mathrm{C} + \mathrm{O_2} \rightarrow \mathrm{CO_2}$$
$$\mathrm{CO_2} + \mathrm{H_2O} \rightarrow \mathrm{H_2CO_3}$$

- (Chang and Lee) formalize each compound as a propositional symbol and express the reactions as implications:

$$\mathrm{HCl} \land \mathrm{NaOH} \rightarrow \mathrm{NaCl} \land \mathrm{H_2O}$$
$$\mathrm{C} \land \mathrm{O_2} \rightarrow \mathrm{CO_2}$$
$$\mathrm{CO_2} \land \mathrm{H_2O} \rightarrow \mathrm{H_2CO_3}$$

Tools

- **Chaff solver** (DPLL+heuristics): https://www.princeton.edu/~chaff/software.html
- **Z3 Theorem prover**, lisp like syntax for defining formulas, API for Java, Python and other languages, not just propositional logic:
    - https://rise4fun.com/z3/tutorial
    - https://github.com/z3prover
- **Minimalist-Propositional-Logic-Resolution-Theorem-Prover** based on Python:

  https://github.com/eglinker/Minimalist-Propositional-Logic-Resolution-Theorem-Prover

The key of these different tools is to understand which of these are more suitable for your specific problem since one works better for a specific problem while others with other problem.

Anyway

- The KR system based on Propositional Logic remains NP-complete in the worst case.
- Is it possible to approach this problem?

But there is a problem.
The problem is that the knowledge representation system we have studied so far based on propositional logic is still **not efficient in the worst case.**
So, is this possible to approach this problem in a way to make it efficient?
We can do it by **reducing** the expressive power of the language. Note that this is **not** a property of the engine (remember that the properties of the engine are **soundness**, **completeness**, **decidability**, **efficiency**), but we have to work on the **syntax** of the representation formalism and try to make it simpler.

What can we do?

- **==> Reducing expressive power**

These class of horn clauses are useful especially in refutation.

- **Horn clauses**: having at most one positive literal.
- There are three types of horn clauses:
  - **Facts**: an atomic formula.
  - **Rules**: (definite clauses) clauses having the form ¬A∨¬B∨¬C∨D also written as implications (A ∧ B ∧ C)→D.

$P \to Q \equiv \neg P \vee Q$
+
De Morgan

    The body is set of preconditions.

    - (A ∧ B ∧ C): BODY of the rule.
    - D: HEAD of thee rule Is the positive literal that we have in the rule.
  - **Goals**: negative literals that represent goals (negated) to be proved having the form ¬P∨¬Q ≡ ¬(P ∧ Q)

So if I want to prove that (P∧Q) holds in my knowledge base, I should negate this goal and result would be a horn clauses without positive literals.

SLD resolution

- KBs including facts and rules:

P→Q
L∧M→P
B∧L→M
A∧P→L
A∧B→L
A
B

BatteryOK

Moves

Moves∧BatteryOK → LightW

The important point is that we can improve the resolution strategies if we apply the SLD algorithm.

SLD resolution

- SLD – resolution: Linear resolution with Selection function for Definite clauses.

"Definite clauses" are actually the horn clauses.

  - C1∧ ...∧C` is the current goal.
  - Select one of the Ci for expansion.
  - look for a matching rule or fact Ci ← A1∧ ...∧An or Ci in the KB
  - Set C1∧ ...∧Ci−1∧A1∧ ...∧An∧Ci +1∧...∧C` or C1∧ ...∧Ci−1∧Ci +1∧...∧C` as the new goal.
- Repeat this process until the current goal is □

(Empty

C1… C' is the current goal which actually should be negated as we stated before, but here we are not representing the negation just because is more useful to see it in this form (but remember that actually is the negated goal).

The what we have do is to select one of the Ci for the expansion (this is the **selection** rule). The SLD resolution does not specify which one.

Then we have to look if our Ci matches a rule or is in the knowledge base (fact).

SLD Resolution example

The beginning negated goal is LightW. Then we look for all the clauses that math, and we have that third actually matches with LightW. So our new goal becomes Moves   BatteryOk. Then we find BatteryOK in the knowledge base so our new goal becomes Moves. Repeat the process and we have that Moves holds since is in the knowledge base and now we have the empty clause.
So we start from the negated goal until we reach the empty clause.

$$\begin{tikzcd}
 & \text{resolvence} & \\
 & \text{LightW} \arrow[d] & \text{(negated GOAL)} \\
 & \text{Moves} \land \text{BatteryOK} \arrow[d] & \\
\text{BatteryOK} \arrow[ur] & \text{Moves} \arrow[d] & \\
\text{Moves} \arrow[ur] & \square & \\
\text{Moves} \land \text{BatteryOK} \rightarrow \text{LightW} \arrow[uuuur] & &
\end{tikzcd}$$

An **and–or tree** is a graphical representation of the reduction of problems (or goals) to conjunctions and disjunctions of subproblems (or subgoals).

Given an initial problem P0 and set of problem solving methods of the form:

P if P1 and … and Pn
the associated and-or tree is a set of labelled nodes such that:

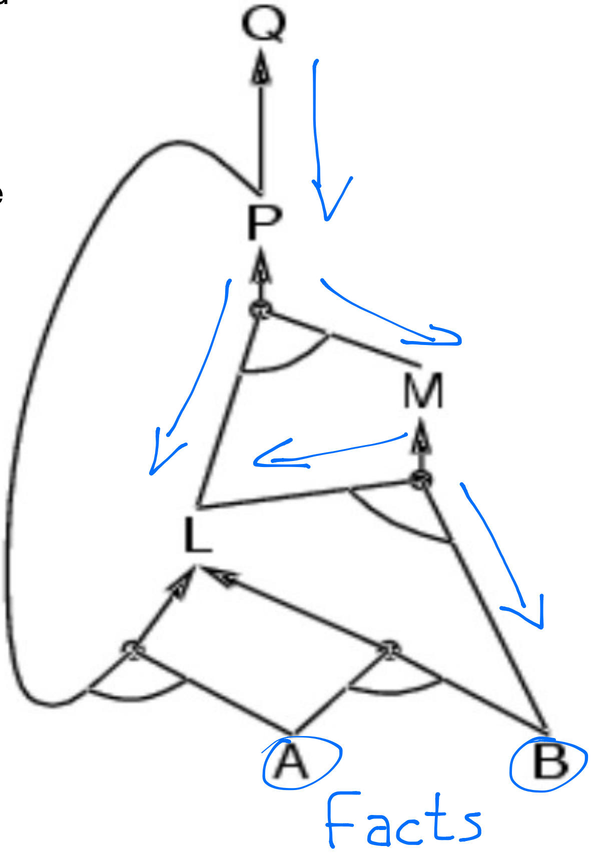
The root of the tree is a node labelled by P0.
For every node N labelled by a problem or sub-problem P and for every method of the form P if P1 and … and Pn, there exists a set of children nodes N1, …, Nn of the node N, such that each node Ni is labelled by Pi. The nodes are conjoined by an arc, to distinguish them from children of N that might be associated with other methods.
A node N, labelled by a problem P, is a success node if there is a method of the form P if nothing (i.e., P is a "fact"). The node is a failure node if there is no method for solving P.

If all of the children of a node N, conjoined by the same arc, are success nodes, then the node N is also a success node. Otherwise the node is a failure node.

AND-OR tree

P→Q
L∧M→P
B∧L→M
A∧P→L
A∧B→L
A
B

From a KB you can always draw a And-Or tree (which is basically a representation of the KB), and do resolution means exploring this tree. However when we write the KB we do not have to care about it, is the engine that generates this tree.



- Goal Q
- Each proposition is associated to an **OR node**. L has two possible solutions, just one have to be solved.
- The BODY of a clause is an **AND node**. All the branches should be solved.

RR with horn clauses

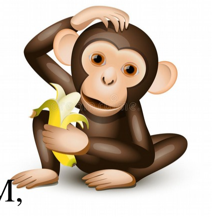
- Soundness: IF $KB^{H} \cup \{\neg F\} \vdash^{SLD} \square$ THEN $KB \models F$
- Completeness:IF $KB^{H} \models F$ THEN $KB \cup \{\neg F\} \vdash^{SLD} \square$
- Decidable.
- Efficient: LINEAR with respect to the dimension of the KB
- Expressive Power: POOR

Tools

- Prolog implements SLD resolution with depth-first search + backtraking.
- **SWI Prolog** can be used with propositional Horn Clauses.

  https://www.swi-prolog.org/

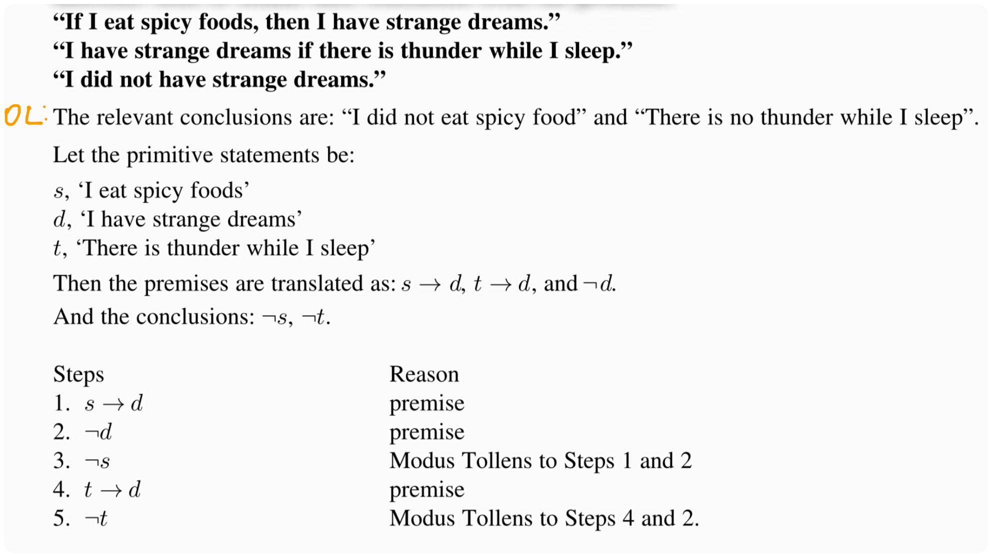
References



- M. Davis and H. Putnam. A computing procedure for quantification theory. Journal of the ACM, 7(3):201–215, 1960.
- M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. Communications of the ACM, 5:394–397, 1962.
- Jerry R. Burch, Edmund M. Clarke, David E. Long, Kenneth L. MacMillan, and David L. Dill. Symbolic model checking for sequential circuit verification. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 13(4):401–424, 1994.
- Tom Bylander. The computational complexity of propositional STRIPS planning. Artificial Intelligence, 69(1-2):165–204, 1994.
- Henry Kautz and Bart Selman. Planning as satisfiability. In Bernd Neumann, editor, Proceedings of the 10th European Conference on Artificial Intelligence, pages 359–363. John Wiley & Sons, 1992.
- João P. Marques-Silva and Karem A. Sakallah. GRASP: A new search algorithm for satisfiability. In Computer-Aided Design, 1996. ICCAD-96. Digest of Technical Papers., 1996 IEEE/ACM International Conference on, pages 220–227, 1996.
- Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: engineering an efficient SAT solver. In Proceedings of the 38th ACM/IEEE Design Automation Conference (DAC'01), pages 530–535. ACM Press, 2001.
- Benjamin Chambers, Panagiotis Manolios, and Daron Vroon. Faster SAT solving with better CNF generation. In Proceedings of the Conference on Design, Automation and Test in Europe, pages 1590–1595, 2009.

EX.1

(He showed these exercises but the slides are not yet online, but he said he will upload).



"If I eat spicy foods, then I have strange dreams."
"I have strange dreams if there is thunder while I sleep."
"I did not have strange dreams."

SOL: The relevant conclusions are: "I did not eat spicy food" and "There is no thunder while I sleep".

Let the primitive statements be:

s, 'I eat spicy foods'
d, 'I have strange dreams'
t, 'There is thunder while I sleep'

Then the premises are translated as: $s \to d$, $t \to d$, and $\neg d$.
And the conclusions: $\neg s$, $\neg t$.

Steps	Reason
1. $s \to d$	premise
2. $\neg d$	premise
3. $\neg s$	Modus Tollens to Steps 1 and 2
4. $t \to d$	premise
5. $\neg t$	Modus Tollens to Steps 4 and 2.

EX.2

" I am dreaming or hallucinating."
" I am not dreaming."
" If I am hallucinating, I see elephants running down the road."

EX.3

" If I work, it is either sunny or partly sunny."
" I worked last Monday or I worked last Friday."
" It was not sunny on Tuesday."
" It was not partly sunny on."