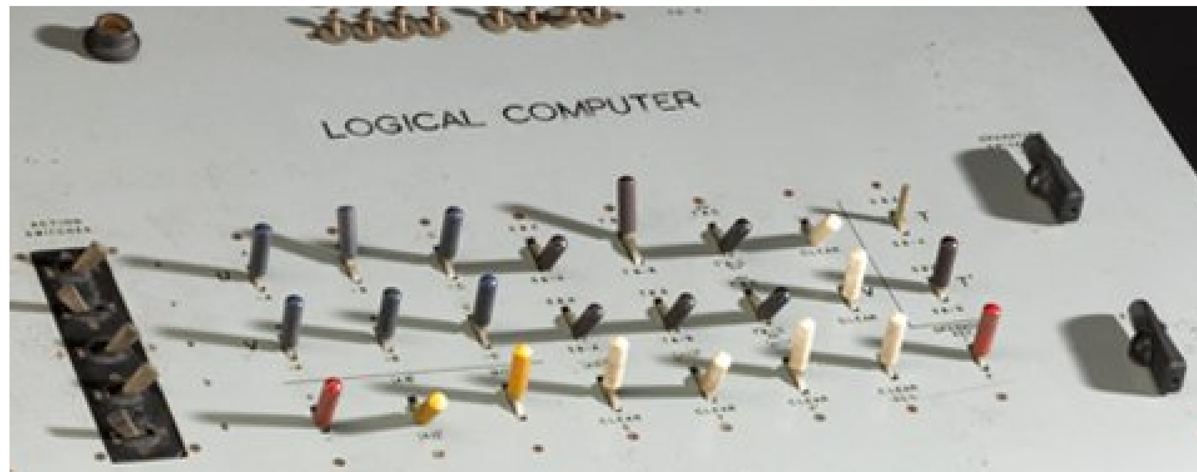
INF551

Computational Logic:

Artificial Intelligence in Mathematical Reasoning

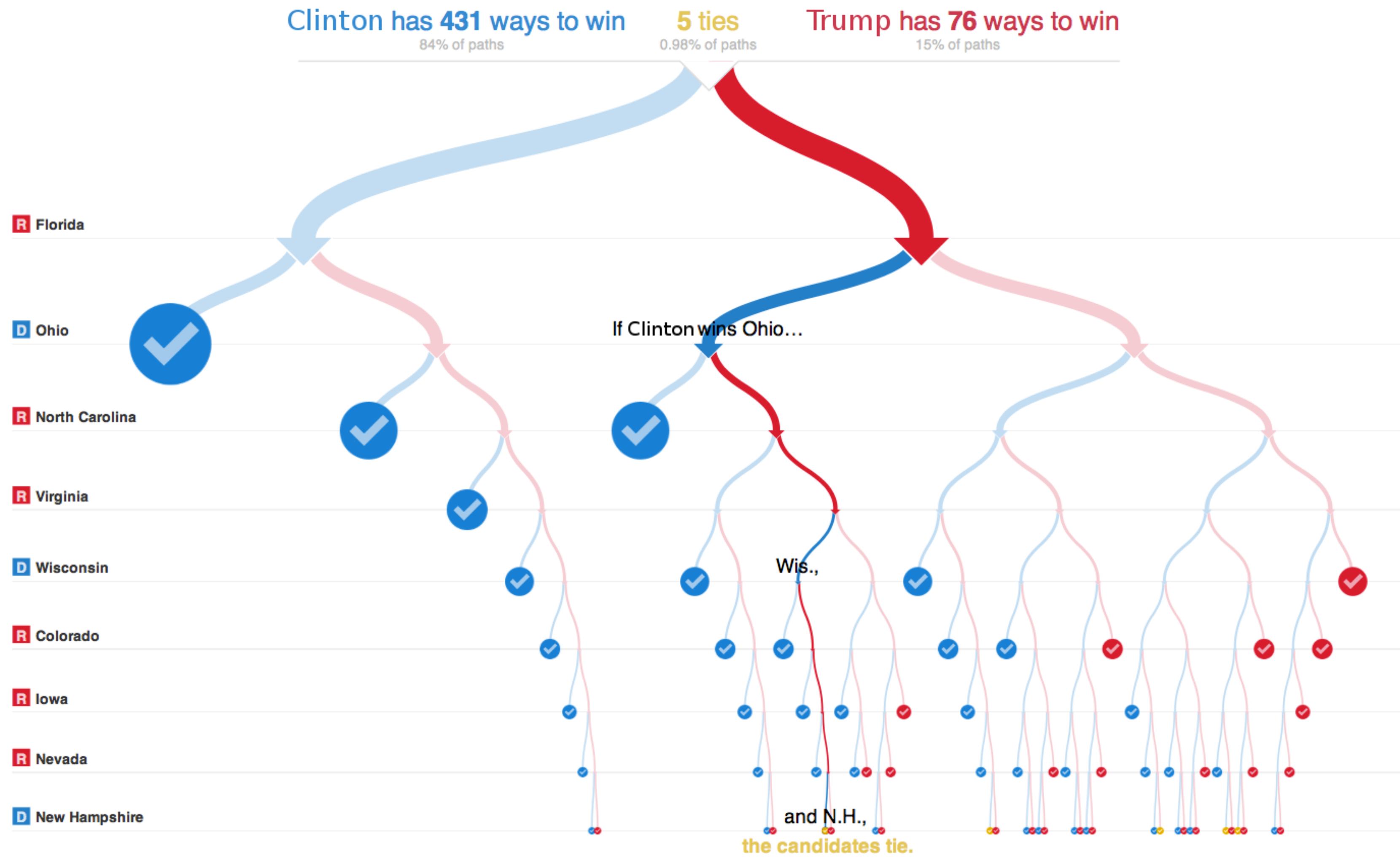


Stéphane Graham-Lengrand

Stephane.Lengrand@Polytechnique.edu

Lecture II

Automated reasoning mechanisms: propositional logic



The semantical view

Remember:

- $\mathcal{T} \vdash A$ if and only if every model of $\mathcal{T}$ is a model of A (soundness & completeness)
- a model structure is either a model of A or a model of $\neg A$ ($[\![A]\!] \in \{0,1\}$)

Provability (of A in theory $\mathcal{T}$)

YES	Do we have $\mathcal{T} \vdash A$?	NO
	$\Updownarrow$	
YES	Is every model of $\mathcal{T}$ a model of A?	NO
NO	Is there a model of $\mathcal{T}$ that is a model of $\neg A$?	YES
	$\Updownarrow$	
NO	Is there a model of $\mathcal{T}, \neg A$?	YES

Satisfiability (of $\neg A$ with respect to theory $\mathcal{T}$)

I. Propositional logic: solving SAT by DPLL

Propositional logic

$\mathcal{T} := \emptyset$

class of formulae := those built from the following signatures:

$$\Sigma := \emptyset$$

$$\Psi := \{p_1/0, p_2/0, \ldots, p_i/0, \ldots\}$$ "Propositional variables"

No predicate symbol of arity ≥ 1 $\Rightarrow$ formulae contain no terms

Formulae contain no terms $\Rightarrow$ $\{{}^{t}\!/\!_{x}\} A$ is A

$$(\forall x A) \Leftrightarrow A$$

$$(\exists x A) \Leftrightarrow A$$

General methodology

To determine whether, in propositional logic, $\vdash A$,

we simply determine whether there is a model of $\neg A$

(i.e. a model structure *satisfying* $\neg A$, i.e. a model structure where $[\![\neg A]\!] = 1$)

What is a model structure for propositional logic? Applying definition of model structure:

A mapping from Ψ (the set of propositional variables) to $\{0, 1\}$

A formula is a finite object, using finitely many propositional variables

To determine whether there is a model of A, using propositional variables $p_1, \ldots, p_n$

it suffices to look at the model structures for $p_1, \ldots, p_n$

How many? $\ldots 2^n$

Problem is **decidable**

(enumerate every structure, and for each of them compute $[\![\neg A]\!]$ with truth tables)

Problem is in fact NP-complete

This can take time

Are there easy cases?

1. If A of the form $l_1 \wedge \cdots \wedge l_m$,

   where the l_i are *literals* (i.e. prop. variables or their negations),

   it's **easy**. 2 cases:

- If A does not feature both p and $\neg p$ for any p,

  A is satisfiable: let the model structure map every l_i to 1
- if A features both p and $\neg p$ for some p,

  A is unsatisfiable

2. If A is a *disjunctive normal form* (DNF): a disjunction of conjunctions of literals,

   it's **easy**:

   Notice there is a model of $A_1 \vee A_2$ iff there is a model of A_1 or a model of A_2,
   so take each of the formulae in the disjunction, then check whether

   one of them is satisfiable ...then A is satisfiable

   or all of them are unsatisfiable ...then A is unsatisfiable

   In both 1 and 2, question can be answered in polynomial time (in the size of formula)

An example of 2.

propositional variables = American swing states

(Florida, Ohio, North Carolina, Virginia, Wisconsin, Colorado, Iowa, Nevada, New Hampshire)

mapped to 1 if won by Clinton, to 0 if won by Trump

A: “there is a tie in the number of electors”

Clinton has **431** ways to win
84% of paths

5 ties
0.98% of paths

Trump has **76** ways to win
15% of paths

R Florida

D Ohio

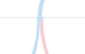
If Clinton wins Ohio…

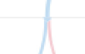
R North Carolina

R Virginia

D Wisconsin

Wis.,





R Colorado

R Iowa

R Nevada

D New Hampshire

and N.H.,
the candidates tie.

If A already expressed as

$$
\begin{aligned}
& (\neg\text{Florida} \wedge \text{Ohio} \wedge \neg\text{NCarolina} \wedge \neg\text{Virginia} \wedge \text{Wisconsin} \wedge \neg\text{Colorado} \wedge \neg\text{Iowa} \wedge \neg\text{Nevada} \wedge \text{NHampshire}) \\
\vee\; & (\neg\text{Florida} \wedge \neg\text{Ohio} \wedge \text{NCarolina} \wedge \text{Virginia} \wedge \neg\text{Wisconsin} \wedge \neg\text{Colorado} \wedge \neg\text{Iowa} \wedge \neg\text{Nevada} \wedge \text{NHampshire}) \\
\vee\; & (\neg\text{Florida} \wedge \neg\text{Ohio} \wedge \neg\text{NCarolina} \wedge \text{Virginia} \wedge \text{Wisconsin} \wedge \text{Colorado} \wedge \neg\text{Iowa} \wedge \neg\text{Nevada} \wedge \neg\text{NHampshire}) \\
\vee\; & (\neg\text{Florida} \wedge \neg\text{Ohio} \wedge \neg\text{NCarolina} \wedge \text{Virginia} \wedge \neg\text{Wisconsin} \wedge \text{Colorado} \wedge \text{Iowa} \wedge \neg\text{Nevada} \wedge \text{NHampshire}) \\
\vee\; & (\neg\text{Florida} \wedge \neg\text{Ohio} \wedge \neg\text{NCarolina} \wedge \text{Virginia} \wedge \neg\text{Wisconsin} \wedge \text{Colorado} \wedge \neg\text{Iowa} \wedge \text{Nevada} \wedge \text{NHampshire})
\end{aligned}
$$

then it’s easy to check satisfiability:

each item of the big disjunction gives rise to a model so long as there is no inconsistency in it (p and $\neg p$ in it)

The worst case...

...is on the contrary when A is a *conjunctive normal form* (CNF):

a conjunction of disjunctions of literals

In that case we can

- transform A into a propositionally equivalent DNF by distributing $\wedge$ over $\vee$:

$$A \wedge (B \vee C) \quad \longrightarrow \quad (A \wedge B) \vee (A \wedge C)$$

Exponential complexity!

- Or try to enumerate the 2^n model structures

Exponential complexity!

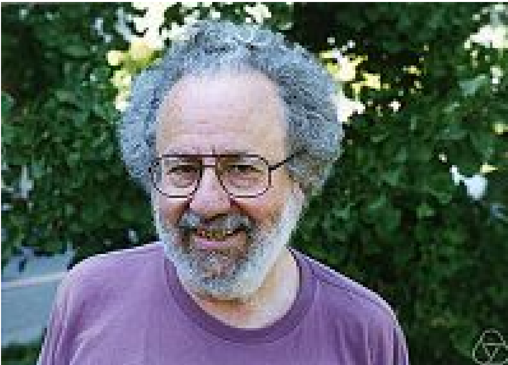
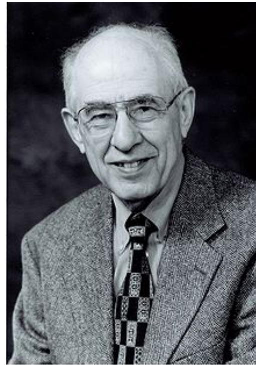
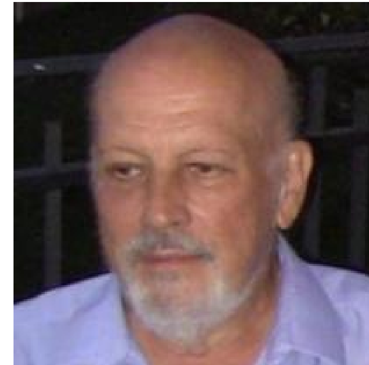
We will choose a smart version of the latter:

the **incremental construction** of a model structure that has a chance of working

DPLL

DPLL

(stands for Davis–Putnam–Logemann–Loveland, 1962)

The idea of the incremental construction is to avoid investigating all 2^n structures

by efficiently cutting branches in the tree of possibilities

Working on worst-case scenario: is a CNF satisfiable?

Terminology

- A *literal* l is a prop. variable p or its negation $\neg p$

  we define $\bar{l}$ by: $\overline{p} := \neg p$ and $\overline{\neg p} := p$
- A *clause* C is a set of literals (to be understood as a disjunction)

  The input is a set ϕ of clauses (to be understood as a conjunction of clauses)

  (in other words, we do not care about associativity and commutativity of $\wedge$ and $\vee$)

  The output is SAT (with a model) or UNSAT

Ingredients and notations

We now organise the incremental construction of a model

The DPLL process will make transitions between *states* that are either the state UNSAT or a state of the form

$$\Delta \| \phi$$

where Δ is a sequence of possibly *tagged* literals,

i.e. literals, like l, and tagged literals, like l^d (also called *decision literals*)

Intuition:

- To each state $\Delta \| \phi$ where Δ is consistent (does not contain l and $\bar{l}$) corresponds a partial model: the partial map assigning 1 to the literals in Δ (regardless of tags)
- When we reach $\Delta \| \phi$, all the possible model structures extending (that represented by) Δ remain to be checked as potential models of ϕ

. . . but not only!

if Δ contains tagged literal l^d, i.e. $\Delta = \Delta_1, l^d, \Delta_2$

it means we have made the arbitrary decision of mapping l to 1, and all the structures extending (that represented by) $\Delta_1, \bar{l}$ also remain to be checked

Outcome of DPLL

We start from $()\|\phi$

We apply the transition rules until we no longer can

If we reach UNSAT, then ϕ is unsatisfiable

If we reach $\Delta\|\phi$ and no rule applies, the structure represented by Δ is a model of ϕ

Transition rules

lit(ϕ) denotes the set of literals appearing in ϕ

We write $\Delta \models \neg C$ if, for every $l \in C$, we have $\bar{l}$ appearing in Δ (tagged or untagged)

i.e. there is no extension of (the structure represented by) Δ that is a model of C

- **Decide**:

  $\Delta \| \phi \Rightarrow \Delta, l^d \| \phi$ where $l \notin \Delta, \bar{l} \notin \Delta, l \in$ lit(ϕ)

- **Backtrack**:

  $\Delta_1, l^d, \Delta_2 \| \phi, C \Rightarrow \Delta_1, \bar{l} \| \phi, C$ if $\Delta_1, l, \Delta_2 \models \neg C$ and no decision literal is in Δ_2

- **Fail**:

  $\Delta \| \phi, C \Rightarrow$ UNSAT if $\Delta \models \neg C$ and there is no decision literal in Δ

Short-cutting the exploration of 2^n possibilities

Above rules are complete:

they describe the depth-first exploration of the tree of possibilities (of height n)

apply Decide eagerly and you get the full exploration of 2^n possibilities

apply Backtrack and Fail eagerly and you stop investigating a sub-tree

as soon as it becomes clear that no model will be found there

To even avoid branching on literal that is a consequence of previous ones, DPLL offers:

Unit propagation: $\Delta \| \phi, C \vee l \Rightarrow \Delta, l \| \phi, C \vee l$ where $\Delta \models \neg C, l \notin \Delta, \bar{l} \notin \Delta$

Pure literal: $\Delta \| \phi \Rightarrow \Delta, l \| \phi$ where $l \notin \Delta, \bar{l} \notin \Delta, l \in \text{lit}(\phi)$ and $\bar{l} \notin \text{lit}(\phi)$

Termination and determinism

System is not deterministic: you are still free to choose how you apply transition rules

Theorem: No matter the strategy applying the rules, DPLL always terminates.

Proof: To each state $\Delta || \phi$, let

- a be the number of model structures that remain to be checked (2^n at first)
- b be the number of literals in lit(ϕ) not determined in Δ

(a, b) strictly decreases with each transition (lexicographically)

$\square$

Which strategy to use?

In your interest: apply Decide as a last resort to factorize the gains of branch-cutting

But: applying other rules first requires checking side-conditions (look at all clauses?)

computational cost

Efficient strategy needs to implement appropriate data structures and algorithms

to perform eager application of other rules with efficient checks of side-conditions

(e.g. technique of “2-watched literals”)

Example

$$
\begin{array}{ll}
() & \| \overline{1} \vee 2, \overline{3} \vee 4, \overline{5} \vee \overline{6}, 6 \vee \overline{5} \vee \overline{2} \\
1^d & \| \overline{1} \vee 2, \overline{3} \vee 4, \overline{5} \vee \overline{6}, 6 \vee \overline{5} \vee \overline{2} \\
1^d 2 & \| \overline{1} \vee 2, \overline{3} \vee 4, \overline{5} \vee \overline{6}, 6 \vee \overline{5} \vee \overline{2} \\
1^d 2 3^d & \| \overline{1} \vee 2, \overline{3} \vee 4, \overline{5} \vee \overline{6}, 6 \vee \overline{5} \vee \overline{2} \\
1^d 2 3^d 4 & \| \overline{1} \vee 2, \overline{3} \vee 4, \overline{5} \vee \overline{6}, 6 \vee \overline{5} \vee \overline{2} \\
1^d 2 3^d 4 5^d & \| \overline{1} \vee 2, \overline{3} \vee 4, \overline{5} \vee \overline{6}, 6 \vee \overline{5} \vee \overline{2} \\
1^d 2 3^d 4 5^d \overline{6} & \| \overline{1} \vee 2, \overline{3} \vee 4, \overline{5} \vee \overline{6}, 6 \vee \overline{5} \vee \overline{2} \\
1^d 2 3^d 4 \overline{5} & \| \overline{1} \vee 2, \overline{3} \vee 4, \overline{5} \vee \overline{6}, 6 \vee \overline{5} \vee \overline{2}
\end{array}
$$

Decision 1^d incompatible with decision 5^d

Decision 3^d has nothing to do with it:

finding that $\overline{5}$ must be in model is lost information for future branch $\overline{3}$

Would like to directly go to $1^d 2 \overline{5} \| \overline{1} \vee 2, \overline{3} \vee 4, \overline{5} \vee \overline{6}, 6 \vee \overline{5} \vee \overline{2}$

Improving on backtrack

Backjump: $\Delta_1, l^d, \Delta_2 \| \phi, C \Rightarrow \Delta_1, l_{bj} \| \phi, C$ when

1. $\Delta_1, l^d, \Delta_2 \models \neg C$
2. $\phi, C \vdash C_0 \vee l_{bj}$
3. $\Delta_1 \models \neg C_0$
4. $l_{bj} \notin \Delta_1, \bar{l}_{bj} \notin \Delta_1$ and $l_{bj} \in$ lit$(\phi, \Delta_1, l^d, \Delta_2)$

for some clause C_0 such that lit$(C_0) \subseteq$ lit(ϕ, C)

Here we apply it with

$\Delta_1 = 1^d 2$

$l^d = 3^d$

$\Delta_2 = 45^d\bar{6}$

$C = 6 \vee \bar{5} \vee \bar{2}$

$l_{bj} = \bar{5}$

$C_0 = \bar{2}$

…and we get a transition from

$$1^d 2 3^d 4 5^d \bar{6} \| \bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$$

to

$$1^d 2 \bar{5} \| \bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$$

Clearly, difficulty (i.e. computational cost) is in inventing clause $\bar{2} \vee \bar{5}$ such that $\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2} \vdash \bar{2} \vee \bar{5}$

Can come from the analysis of conflict

$$1^d 2 3^d 4 5^d \bar{6} \models \neg(6 \vee \bar{5} \vee \bar{2})$$

But the work is not wasted!

We have learnt a clause (called the *Backjump clause*) that we can reuse later

Learn: $\Delta \| \phi \Rightarrow \Delta \| \phi, C$ if $\mathsf{lit}(C) \subseteq \mathsf{lit}(\phi)$ and $\phi \vdash C$

Typical application of this is after a Backjump

Careful in eagerly adding such clauses: having too many (redundant) clauses can also slow down the process

Sometimes it can be useful to remove a redundant clause:

Forget: $\Delta \| \phi, C \Rightarrow \Delta \| \phi$ if $\phi \vdash C$

Yet another potentially useful rule:

Restart: $\Delta \| \phi \Rightarrow () \| \phi$

only useful if ϕ is different from original set of clauses (clauses have been learnt)

Conclusions

New rules, new possibilities

$\Longrightarrow$ more chances of speeding up process

$\Longrightarrow$ more strategy to control how to apply them

Not in the scope of this course to investigate them.

Back to the original problem: deciding provability in propositional logic

We wonder whether

$$\vdash A$$

Assume $\neg A$ to be, or reduce $\neg A$ to, a CNF

Start DPLL with $() \| \neg A$

If UNSAT: then we have $\vdash A$

If SAT: then we have $\not\vdash A$

Of course, reducing $\neg A$ to a CNF is also computational costly (imagine if it is a DNF!), so there are ways to adapt DPLL to formulae that are not CNF

Questions?