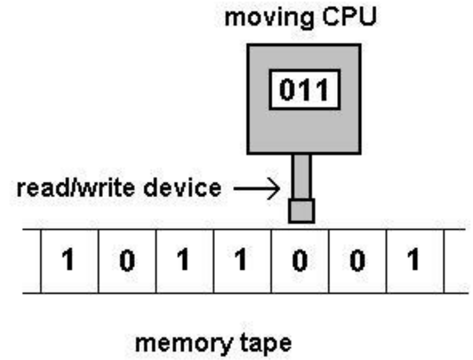
Entailment and Algorithms

Marco Piastra

Computational Complexity in a Quick Ride

Turing Machine (A. Turing, 1937)



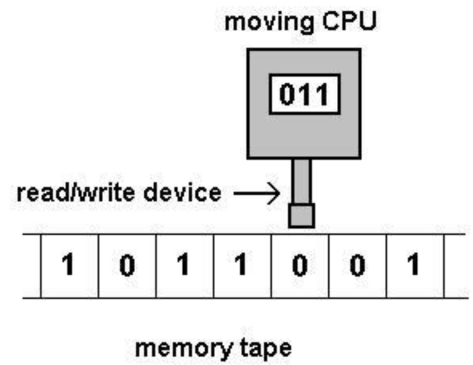
- A more precise definition
  - A non-empty and finite set of *states* S
    - At each instant the machine is in a state $s \in S$
  - A non-empty and finite alphabet of *symbols* Q
    - The alphabet Q includes a *blank*, default symbol b
    - Each cell in the tape contains a symbol $q \in Q$
  - A partial *transition* function

$$\tau : S \times Q \rightarrow S \times Q \times \{\text{Left}, \text{None}, \text{Right}\}$$

current state — *input symbol* — *next state* — *output symbol* — *head move*

    - *It is partial in the sense it needs not be defined on any input tuple*
  - A subset of *terminal* states $T \subseteq S$
  - An initial state $s_0 \in S$

Turing Machine (A. Turing, 1937)



- A *busy beaver* example (3 states)

$$S = \{A, B, C, \text{HALT}\}$$

$$s_0 = A \qquad F = \{\text{HALT}\}$$

$$Q = \{0, 1\} \qquad b = 0$$

$$\begin{aligned}
\tau = & \\
& < A, 0 > \to < B, 1, \text{Right} > \\
& < A, 1 > \to < C, 1, \text{Left} > \\
& < B, 0 > \to < A, 1, \text{Left} > \\
& < B, 1 > \to < B, 1, \text{Right} > \\
& < C, 0 > \to < B, 1, \text{Left} > \\
& < C, 1 > \to < \text{HALT}, 1, \text{Right} >
\end{aligned}$$

Assume that the tape is infinite and plenty of blank symbols 0

What does this machine do?

Decisions and decidability (automation)

- What is a <u>*problem*</u>?

  A *problem* is an association, i.e. a **relation** between *inputs* and *outputs* (i.e. *solutions*)

  $K : \langle \mathrm{I} , \mathrm{S} \rangle$

- *Search* problem

  Typically, K associates *one* input to *many* solutions

  Optimization problems

  A *search problem* plus an *objective* or *cost* function

  $c : \mathrm{S} \rightarrow \mathbb{R}$ (i.e. from S to the set of real numbers)

  In general, the task is finding the solution(s) having maximal or minimal cost

- ***Decision*** problem

  The solution space S is $\{0, 1\}$
  and K associates each input to a <u>*unique*</u> solution: $K : \mathrm{I} \rightarrow \{0, 1\}$

  Example: $\Gamma \models \varphi$?

  The input space I contains all possible combinations of set Γ of wffs with individual wffs φ
  The solution is uniquely defined for any instance of such problems in I

Decisions and decidability (automation)

- ***Decidable*** problem

  A decision problem K for which there exists an algorithm, i.e a *Turing machine*,

  (there are other ways of defining an algorithm or an *effective procedure*: they are all equivalent)

  that ***always terminates*** and produces the right answer in ***finite time***.

Example of an *undecidable* problem: The *Halting Problem*

Given the formal description of a particular Turing machine and a specific input,
is it possible to tell if whether it will either halt eventually or run forever?

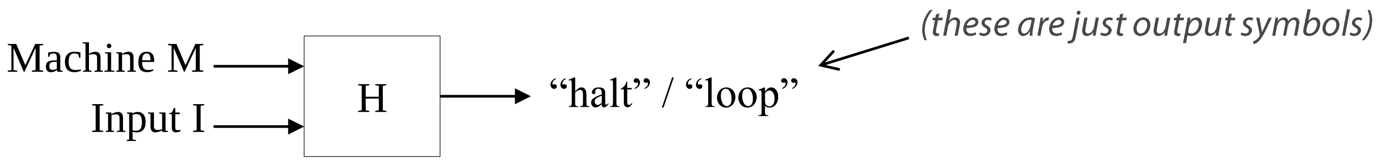
In other words, does it exist a Turing machine that, given in input the description of *another* Turing machine, will always produce the answer desired?

The answer is **no** (such a Turing machine *cannot* exist)

An aside: The *Halting Problem*

- Intuitive ideas behind the proof (i.e. of the *undecidability* of this problem)

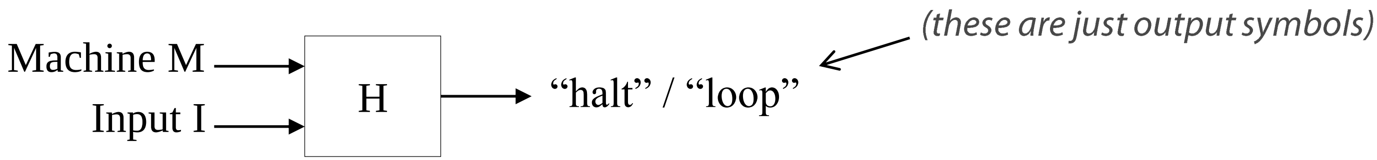
  Let's assume there exists a Turing machine H that, given the description of a Turing machine M with input I always terminates producing an output "halt" or "loop" depending on whether M with input I will terminate or not



An aside: The *Halting Problem*

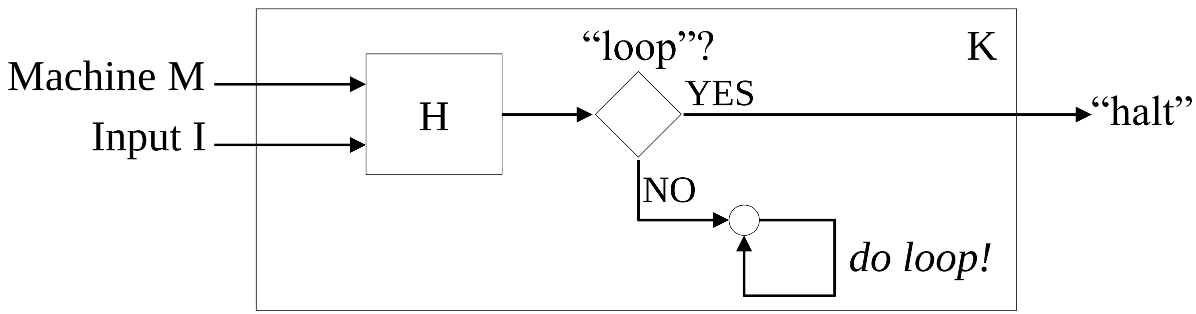
- Intuitive ideas behind the proof (i.e. of the *undecidability* of this problem)

  Let's assume there exists a Turing machine H that, given the description of a Turing machine M with input I always terminates producing an output "halt" or "loop" depending on whether M with input I will terminate or not



  <u>Assume H existed</u>

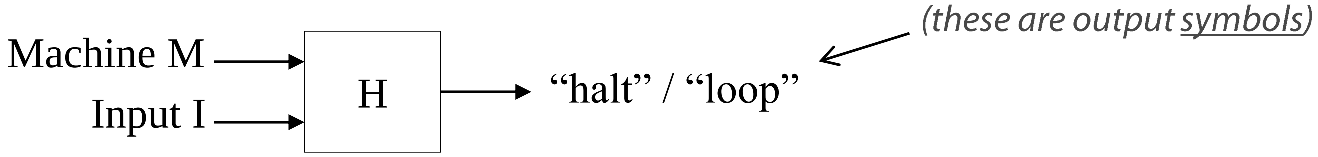
  We could build another Turing machine K that enters an infinite loop whenever the output of H is "halt" and that terminates, with output "halt", when H outputs "loop"



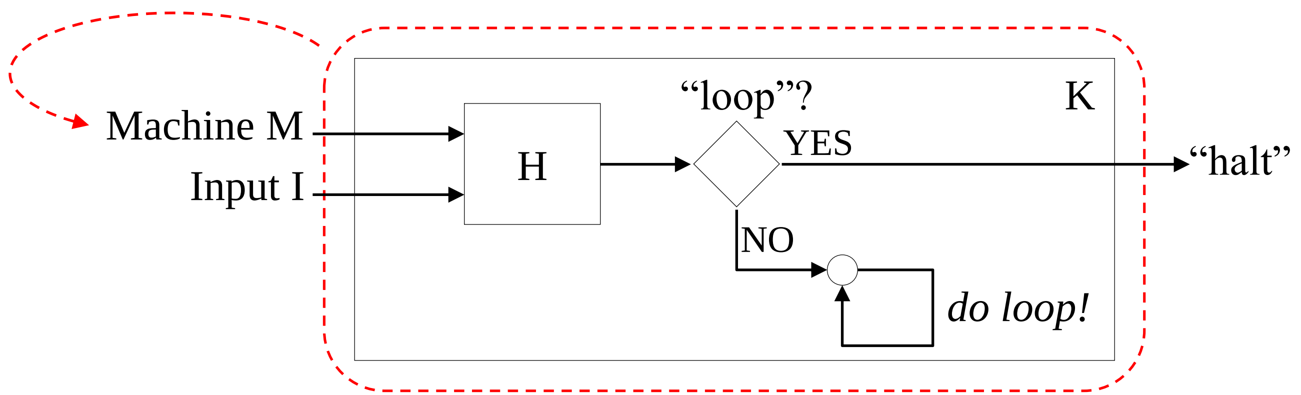
An aside: The *Halting Problem*

- Intuitive ideas behind the proof (i.e. of the *undecidability* of this problem)

  Let's assume there exists a Turing machine H that, given the description of a Turing machine M with input I always terminates producing an output "halt" or "loop" depending on whether M with input I will terminate or not



  Assume H *existed*
  We could build another Turing machine K that enters an infinite loop whenever the output of H is "halt" and that terminates, with output "halt", when H outputs "loop"



  What will be the output of K when given K *itself* as the input?
  K should *diverge* when K *terminates* and vice-versa: i.e. we have an absurdity

Computational complexity,

These notions apply to ***decidable problems*** only

It is based on the performances of a (known) Turing machine that gives the answer with respect to the *worst case* (i.e. the less favorable input)

- Time complexity

  The number of *steps* that the Turing machine requires for computing the answer, as a function of some numerical dimension of the input (e.g. the number of atoms in a wff)

- Memory complexity

  The number of tape *cells* that the Turing machine requires for computing the answer, as a function of some numerical dimension of the input

- Big-O notation

$$f(x) = O(g(x))$$

means that

$$\exists M > 0,\ \exists x_0 > 0 \quad \text{such that} \quad |f(x)| \leq M|g(x)|, \quad \forall x > x_0$$

Classes P, NP and NP-complete - The SAT problem

- Class P

  The class of problems for which there is a Turing machine that requires $O(\mathrm{P}(n))$ time

  where P( ) is a polynomial of finite degree and n is the dimension of the (*worst-case*) input

- Class NP

  The class of all problems:

  a) A method for *enumerating* all possible answers (i.e. *recursive enumerability*)

  b) An algorithm in class P that *verifies* if a possible answer is also a *solution*

  It includes all problems in class P (that is, $\mathrm{P} \subseteq \mathrm{NP}$)

Classes P, NP and NP-complete – The SAT problem

- Class NP-complete

  It is a subclass of NP (NP-complete $\subseteq$ NP)

  A problem K is NP-complete if every problem in class NP is *reducible* to K

- Reducibility

  For class NP-complete

  Consider a problem K for which a decision algorithm $M(K)$ is known

  A problem J is *reducible* to K if there exist a decision algorithm $M(J)$ such that:

  a) algorithm $M(K)$ is called just once, as a “subroutine”, at the end of $M(J)$

  b) apart from $M(K)$, $M(J)$ has polynomial complexity

- The problem SAT

  Is NP-complete (*historically, it is the first one to be known*)

  Moral: if we had a polynomial decision algorithm for SAT, we would also have that

  $P = NP$

  This fact is not known, it is believed that: $P \neq NP$

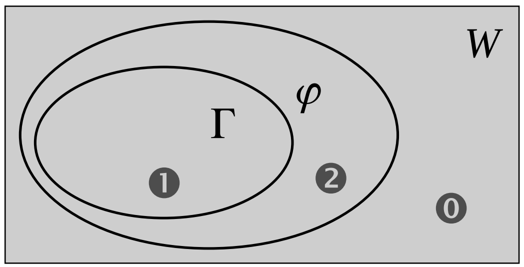
  (*and a lot will change in the digital world, if this proves to be false*)

Entailment as a Decision Problem

Transforming problems: entailment as satisfiability

- Step 1: the decision problem " $\Gamma \models \varphi$? " can be transformed into a *satisfiability* problem

  In fact, $\Gamma \models \varphi$ iff $\Gamma \cup \{\neg\varphi\}$ is *not* satisfiable



($w(\Gamma)$ is the set of possible worlds that satisfy Γ)

$\Gamma \models \varphi \;\Rightarrow\; w(\Gamma) \subseteq w(\{\varphi\})$ ❶ ⊆ {❶, ❷}

$w(\{\neg\varphi\}) =$ ⓿

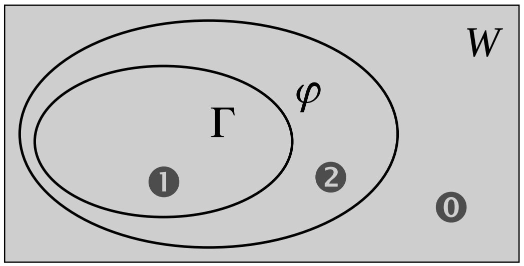
$w(\Gamma \cup \{\neg\varphi\}) = w(\Gamma) \cap w(\{\neg\varphi\})$

$w(\Gamma \cup \{\neg\varphi\}) = \varnothing$ ❶ ∩ ⓿ = ∅

Transforming problems: entailment as satisfiability

- Step 1: the decision problem "$\Gamma \models \varphi$?" can be transformed into a *satisfiability* problem

  In fact, $\Gamma \models \varphi$ iff $\Gamma \cup \{\neg\varphi\}$ is *not* satisfiable



($w(\Gamma)$ is the set of possible worlds that satisfy Γ)

$\Gamma \models \varphi \;\Rightarrow\; w(\Gamma) \subseteq w(\{\varphi\})$ ❶ ⊆ {❶, ❷}

$w(\{\neg\varphi\}) =$ ⓿

$w(\Gamma \cup \{\neg\varphi\}) = w(\Gamma) \cap w(\{\neg\varphi\})$

$w(\Gamma \cup \{\neg\varphi\}) = \varnothing$ ❶ ∩ ⓿ = ∅

- Step 2: the decision problem "is $\Gamma \cup \{\neg\varphi\}$ satisfiable?" can be transformed into a wff *satisfiability* problem

  Taking this one step further, we can transform $\Gamma \cup \{\neg\varphi\}$ into *just one formula*:

$$\bigwedge (\Gamma \cup \{\neg\varphi\})$$

This is the wff obtained by combing all the wffs in $\Gamma \cup \{\neg\varphi\}$ with $\wedge$, it is called the *conjunctive closure* of the set $\Gamma \cup \{\neg\varphi\}$

Satisfiability and decidability (in L_P)

- Is the decision problem "is the wff φ satisfiable?" ***decidable***?
  - It can be transformed into a *search* problem
    - i.e. finding a possible world (in the set of all possible worlds) that satisfies φ
      - In the scientific literature, this problem is called "SAT"
    - *Intuition*: we can try every possible value assignment for the atoms in φ
    - *Hint:* the problem is NP-complete

Exhaustive (Tree) Search

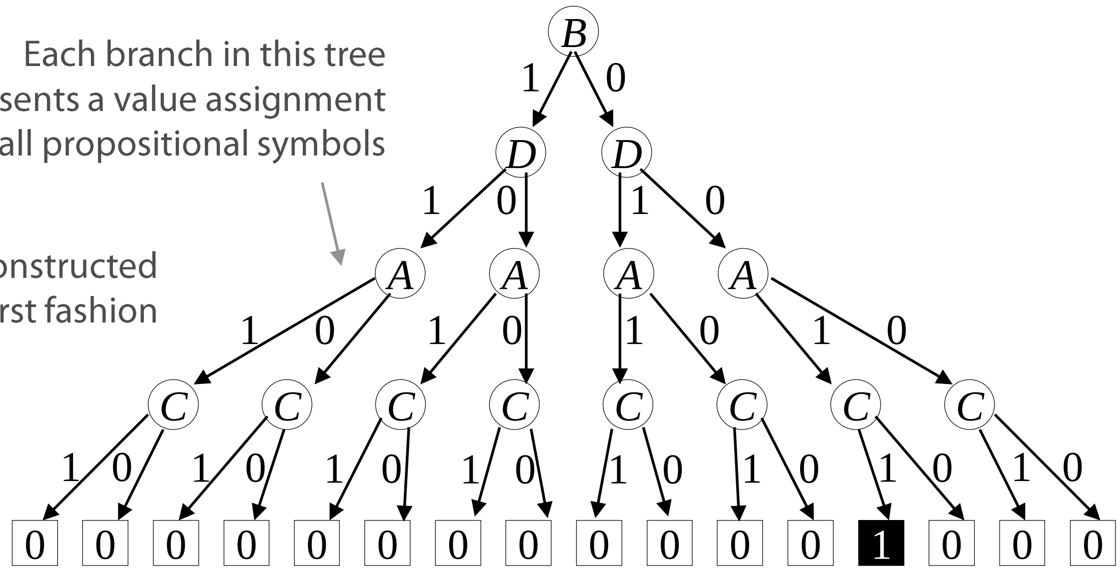
Satisfiability and decidability (in L_P)

Example: is this wff *satisfiable*?

$$\neg(B \land D \land \neg(A \land C))$$

Each branch in this tree represents a value assignment to all propositional symbols

The tree can be constructed in a depth-first fashion



Each leaf in the tree is the value of the wff with the corresponding value assignments

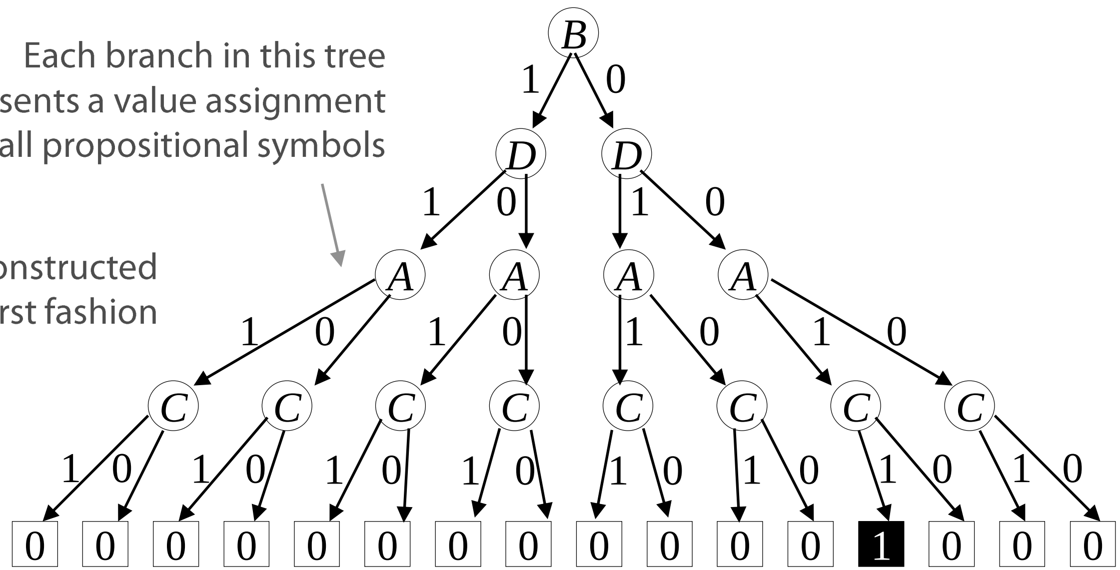
Satisfiability and decidability (in L_P)

Example: is this wff *satisfiable*?

$$\neg(B \wedge D \wedge \neg(A \wedge C))$$

Each branch in this tree represents a value assignment to all propositional symbols

The tree can be constructed in a depth-first fashion



Each leaf in the tree is the value of the wff with the corresponding value assignments

In this case, a depth-first algorithm stops here

But the algorithm is forced to try all possible assignments when ψ is *not* satisfiable, for example with: $(\neg B \wedge \neg D \wedge \neg A \wedge \neg C)$

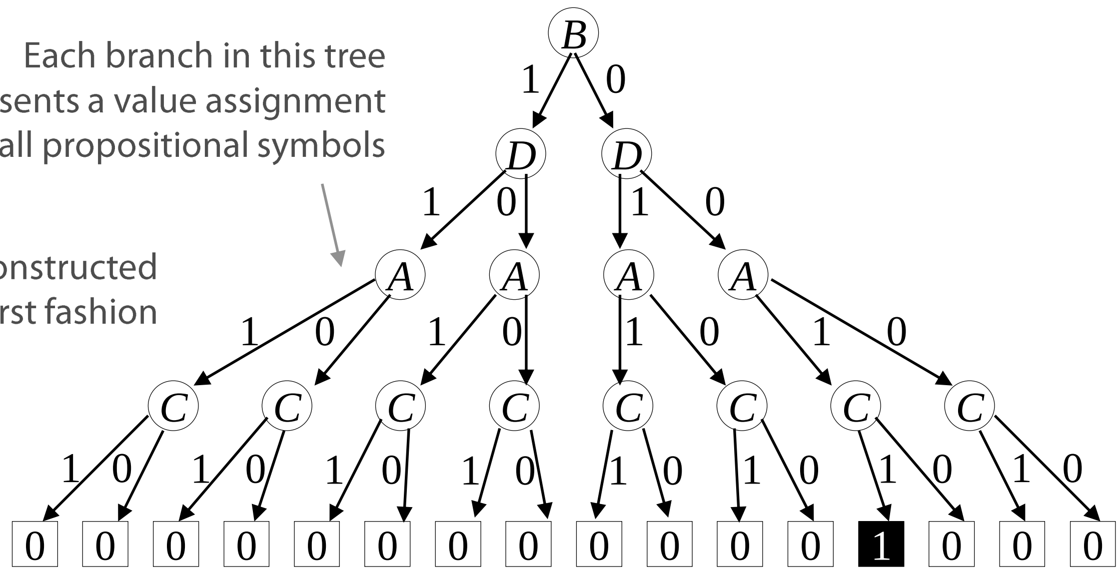
Satisfiability and decidability (in L_P)

Example: is this wff <u>*satisfiable*</u>?

$$\neg(B \land D \land \neg(A \land C))$$

Each branch in this tree represents a value assignment to all propositional symbols

The tree can be constructed in a depth-first fashion



Each leaf in the tree is the value of the wff with the corresponding value assignments

In this case, a depth-first algorithm stops here

But the algorithm is forced to try all possible assignments when ψ is *not* satisfiable, for example with: $(\neg B \land \neg D \land \neg A \land \neg C)$

This method has $O(2^n)$ time complexity, where n is the number of propositional symbols

Semantic Tableaux

Semantic Tableau, alpha and beta rules

- *Semantic tableau* is a method

  which can be implemented as a Turing machine

- It is a decision algorithm for the problem
  "is Σ satisfiable?"

  where Σ is a set of wffs in $\boldsymbol{L_P}$

  In spite of its name, it is a *symbolic* method: it works on the structure of wffs only

  No explicit assignments of (semantic) values are involved

Semantic Tableau, alpha and beta rules

- A tableau is a set of wffs in $\boldsymbol{L_P}$

  The method starts from an *initial* tableau

  (i.e. the set Σ whose satisfiability is to be determined)

  It is based on rules that transform each one wff into two wffs

- Alpha rules (i.e. expansion)

(a1) $\neg(\neg\varphi)$ → φ

(a2) $\varphi \wedge \psi$ → φ, ψ

(a3) $\neg(\varphi \vee \psi)$ → $\neg\varphi, \neg\psi$

(a4) $\neg(\varphi \to \psi)$ → $\varphi, \neg\psi$

- Beta rules (i.e. bifurcation)

(b1) $\varphi \vee \psi$ → φ | ψ

(b2) $\neg(\varphi \wedge \psi)$ → $\neg\varphi$ | $\neg\psi$

(b3) $\varphi \to \psi$ → $\neg\varphi$ | ψ

(b4) $\varphi \leftrightarrow \psi$ → $\neg\varphi, \neg\psi$ | φ, ψ

(b5) $\neg(\varphi \leftrightarrow \psi)$ → $\neg\varphi, \psi$ | $\varphi, \neg\psi$

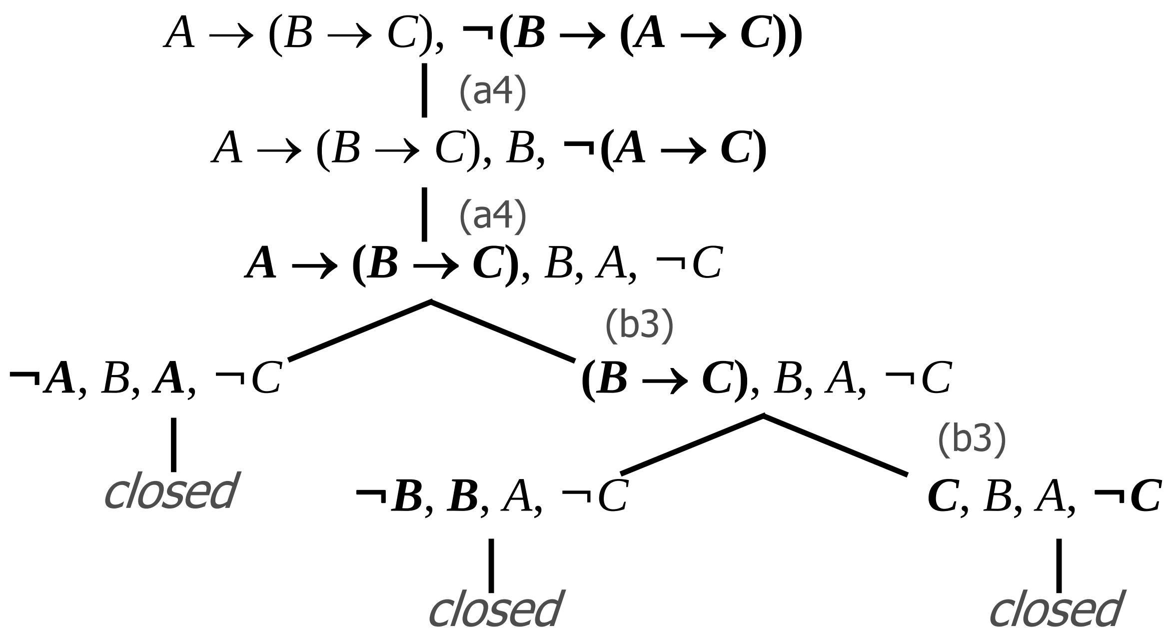
Semantic Tableau – a working example

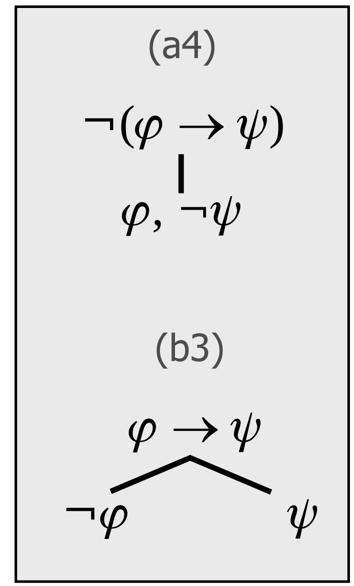
- Original problem: "$\Gamma \models \varphi$?"

  Example input: $A \to (B \to C) \models B \to (A \to C)$?

- Transformed problem: "is $\Gamma \cup \{\neg\varphi\}$ satisfiable?"

  Hence the initial tableau is $\Gamma \cup \{\neg\varphi\}$





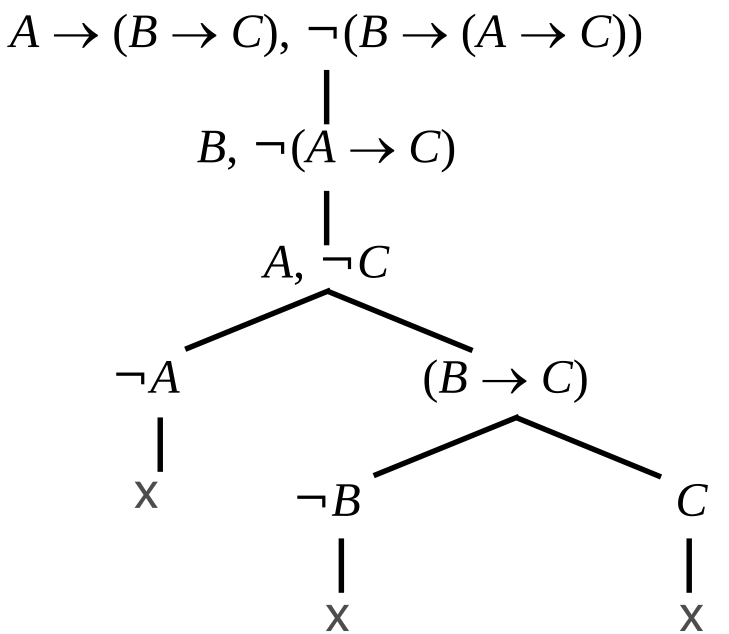
Semantic Tableau – a working example

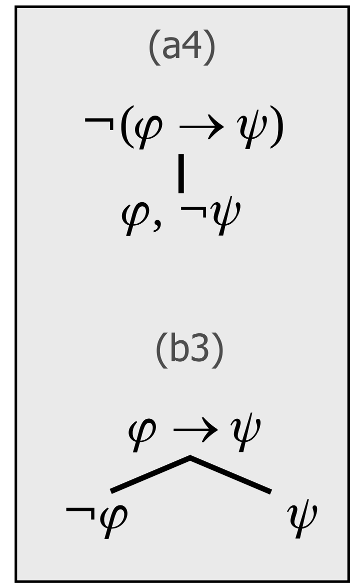
- Original problem: "$\Gamma \models \varphi$?"

  Example input: $A \to (B \to C) \models B \to (A \to C)$?

- Transformed problem: "is $\Gamma \cup \{\neg\varphi\}$ satisfiable?"

  Hence the initial tableau is $\Gamma \cup \{\neg\varphi\}$





The usual notation in textbooks is even more concise:
only those wffs that are *added* to the initial tableau in each branch are shown in the tree

Semantic Tableau – algorithm recap

- Algorithm (informal description – see Lab for the implementation):

  Input problem: " $\Gamma \models \varphi$? "

  The input problem is transformed into "is $\Gamma \cup \{\neg\varphi\}$ satisfiable?"

  Methods of this type are also called '*by refutation*'

  For each active tableau (i.e. the *leaves* in the tree),

  There could be two cases:

  1) The tableau contains only *literals*
     If the tableau contains a *complementary pair of literals*
     then declare it *closed*
     else declare it *open* (i.e. failure)
  2) The tableau contains one or more *composite* wff
     First try to apply an *alpha* rule,
     otherwise, if this is not possible, try to apply a *beta* rule.
     In either case, two new tableau will be generated

  Output: the tree structure of tableau

Semantic Tableau - (required) algorithm properties

- **Termination**

    The algorithm never *diverges* (i.e. it never enters an infinite loop)

    Each application of either alpha or beta rule *simplifies* a wff (i.e. it makes it *less* composite): so the application of rules cannot continue forever

- ***Symbolic derivation***

    As already stated, in spite of its name, this is a *symbolic* method

    We write

    $$\Gamma \vdash_{ST} \varphi$$

    iff the *Semantic Tableau* method is successful (i.e. all leaves are *closed*) for $\Gamma \cup \{\neg\varphi\}$

  How do we know that $\Gamma \vdash_{ST} \varphi \;\Rightarrow\; \Gamma \models \varphi$?

    (*Soundness* - also *correctness* - of the method)

    Exercise: prove it
    (*hint*: consider the condition on $\Gamma \cup \{\neg\varphi\}$ and think about how it relates to each *rule*)

  How do we know that $\Gamma \models \varphi \;\Rightarrow\; \Gamma \vdash_{ST} \varphi$?

    (*Completeness* of the method)

    Proving it is definitely more difficult: see textbook (i.e. Ben-Ari)

Semantic Tableau - (required) algorithm properties

- **Termination**

  The algorithm never *diverges* (i.e. it never enters an infinite loop)

  Each application of either alpha or beta rule *simplifies* a wff (i.e. it makes it *less* composite): so the application of rules cannot continue forever

- ***Soundness***

  $\Gamma \vdash_{ST} \varphi \;\Rightarrow\; \Gamma \models \varphi$

- **Completeness**

  $\Gamma \models \varphi \;\Rightarrow\; \Gamma \vdash_{ST} \varphi$

- **Termination + Soundness + Completeness = *Decision Algorithm***

  (for propositional logic)

Which method is faster?

- Time complexity (remember: consider the *worst case*)

  The `brute-force search' and *Semantic Tableau* have the same complexity : $O(2^n)$

- *How well do these method perform in practice?*

  It depends

Example 1 (try it)**:**

$$A \land B \land C \land \neg A$$

The `brute-force search' requires $2^3 = 8$ attempts

The Semantic Tableau method requires applying the same alpha rule 3 times

Example 2 (try it)**:**

$$(A \lor B) \land (A \lor \neg B) \land (\neg A \lor B) \land (\neg A \lor \neg B)$$

The `brute-force search' requires $2^2 = 4$ attempts

The Semantic Tableau method requires applying the same alpha rule 3 times; then the same beta rule is applied exhaustively producing a tree with 4 levels, with each node in a tree with a branching factor 2

At the end, the tree has $2^4 = 16$ leaves (all *closed* tableau)

Resolution

Inference rule: Resolution

$\varphi \vee \chi, \neg\chi \vee \psi \vdash \varphi \vee \psi$

$\varphi \vee \psi$ is also called the *resolvent* of $\varphi \vee \chi$ e $\neg\chi \vee \psi$

The resolution rule is *correct*

In fact $\varphi \vee \chi, \neg\chi \vee \psi \vdash \varphi \vee \psi \quad \Rightarrow \quad \varphi \vee \chi, \neg\chi \vee \psi \models \varphi \vee \psi$

φ	ψ	χ	$\varphi \vee \chi$	$\neg\chi \vee \psi$	$\varphi \vee \psi$
0	0	0	0	1	0
0	0	1	1	0	0
0	1	0	0	1	1
0	1	1	1	1	1
1	0	0	1	1	1
1	0	1	1	0	1
1	1	0	1	1	1
1	1	1	1	1	1

Normal forms

= translation of each wff into an equivalent wff having a specific structure

- **Conjunctive Normal Form** (CNF)

  A wff with a structure

  $\alpha_1 \land \alpha_2 \land \ldots \land \alpha_n$

  where each α_i has a structure

  $(\beta_1 \lor \beta_2 \lor \ldots \lor \beta_n)$

  where each β_j is a *literal* (i.e. an atomic symbol or the negation of an atomic symbol)

  Examples:

  $(B \lor D) \land (A \lor \neg C) \land C$

  $(B \lor \neg A \lor \neg C) \land (\neg D \lor \neg A \lor \neg C)$

- **Disjunctive Normal Form** (DNF)

  A wff with a structure

  $\beta_1 \lor \beta_2 \lor \ldots \lor \beta_n$

  where each β_i has a structure

  $(\alpha_1 \land \alpha_2 \land \ldots \land \alpha_n)$

  where each α_j is a *literal*

Conjunctive Normal Form

- **Translation into CNF** (it can be automated)

  Exhaustive application of the following rules:

  1) Rewrite $\rightarrow$ and $\leftrightarrow$ using $\wedge$, $\vee$, $\neg$

  2) Move $\neg$ inside composite formulae

     "De Morgan laws": $\neg(\varphi \wedge \psi) \equiv (\neg\varphi \vee \neg\psi)$

     $\neg(\varphi \vee \psi) \equiv (\neg\varphi \wedge \neg\psi)$

  3) Eliminate double negations: $\neg\neg$

  4) Distribute $\vee$

$$((\varphi \wedge \psi) \vee \chi) \equiv ((\varphi \vee \chi) \wedge (\psi \vee \chi))$$

Examples:

$(\neg B \rightarrow D) \vee \neg(A \wedge C)$

$B \vee D \vee \neg(A \wedge C)$ (rewrite $\rightarrow$)

$B \vee D \vee \neg A \vee \neg C$ (De Morgan)

$\neg(B \rightarrow D) \vee \neg(A \wedge C)$

$\neg(\neg B \vee D) \vee \neg(A \wedge C)$ (rewrite $\rightarrow$)

$(B \wedge \neg D) \vee (\neg A \vee \neg C)$ (De Morgan)

$(B \vee \neg A \vee \neg C) \wedge (\neg D \vee \neg A \vee \neg C)$ (distribute $\vee$)

Clausal Forms

= each wff is translated into an equivalent <u>set</u> of wffs having a specific structure

- **Clausal Form** (CF)

  Starting from a wff in CNF

  $\alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n$

  the clausal form is simply the set of all *clauses*

  $\{\alpha_1, \alpha_2, \ldots, \alpha_n\}$

  Examples:

  $(B \vee D) \wedge (A \vee \neg C) \wedge C$

  $\{(B \vee D), (A \vee \neg C), C\}$

- **Special notation**

  Each clause is usually written as a *set*

  $\beta_1 \vee \beta_2 \vee \ldots \vee \beta_n$

  $\{\beta_1, \beta_2, \ldots, \beta_n\}$

  Example:

  $\{\{B, D\}, \{A, \neg C\}, \{C\}\}$

  A set of *literals*:
  ordering is irrelevant
  no multiple copies

Resolution by refutation

- Algorithm

  Problem: "$\Gamma \vdash \varphi$" ?

  The problem is transformed into: is "$\Gamma \cup \{\neg\varphi\}$" *coherent*?

  If $\Gamma \vdash \varphi$ then $\Gamma \cup \{\neg\varphi\}$ is incoherent and therefore a contradiction can be derived

  $\Gamma \cup \{\neg\varphi\}$ is translated into CNF hence in CF

  The resolution algorithm is applied to the set of *clauses* $\Gamma \cup \{\neg\varphi\}$

  At each step:

  a) Select a pair of clauses $\{C_1, C_2\}$ containing a pair of *complementary literals* making sure that this combination has never been selected before

  b) Compute C as the *resolvent* of $\{C_1, C_2\}$ according to the resolution rule.

  c) Add C to the set of clauses

  Termination:

  When C is the empty clause $\{\ \}$

  or there are no more combinations to be selected in step a)

Resolution by refutation

- The same example as before

  $B \lor D \lor \neg A \lor \neg C, B \lor C,\ A \lor D,\ \neg B \vdash D$

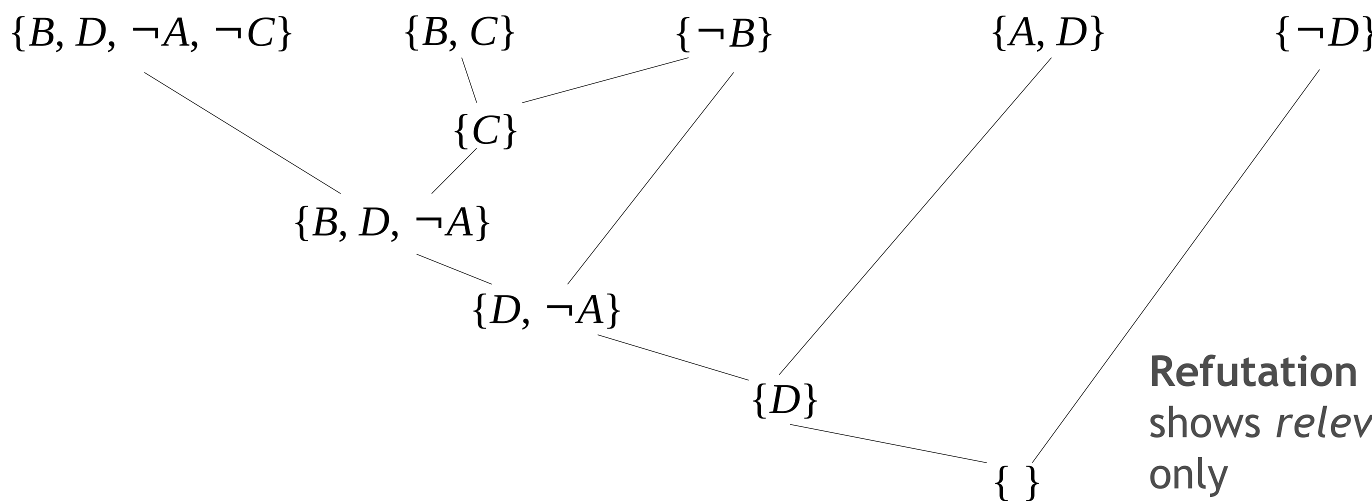
  Refutation + rewrite in CNF:

  $B \lor D \lor \neg A \lor \neg C, B \lor C,\ A \lor D,\ \neg B, \neg D$

  Rewrite in CF:

  $\{B, D, \neg A, \neg C\}, \{B, C\}, \{A, D\}, \{\neg B\}, \{\neg D\}$

  Applying the resolution rule, *one pair of literals at time*:



Refutation graph: shows *relevant* resolutions only

Resolution by refutation

- The same example as before

  $B \lor D \lor \neg A \lor \neg C, B \lor C,\ A \lor D,\ \neg B \vdash D$

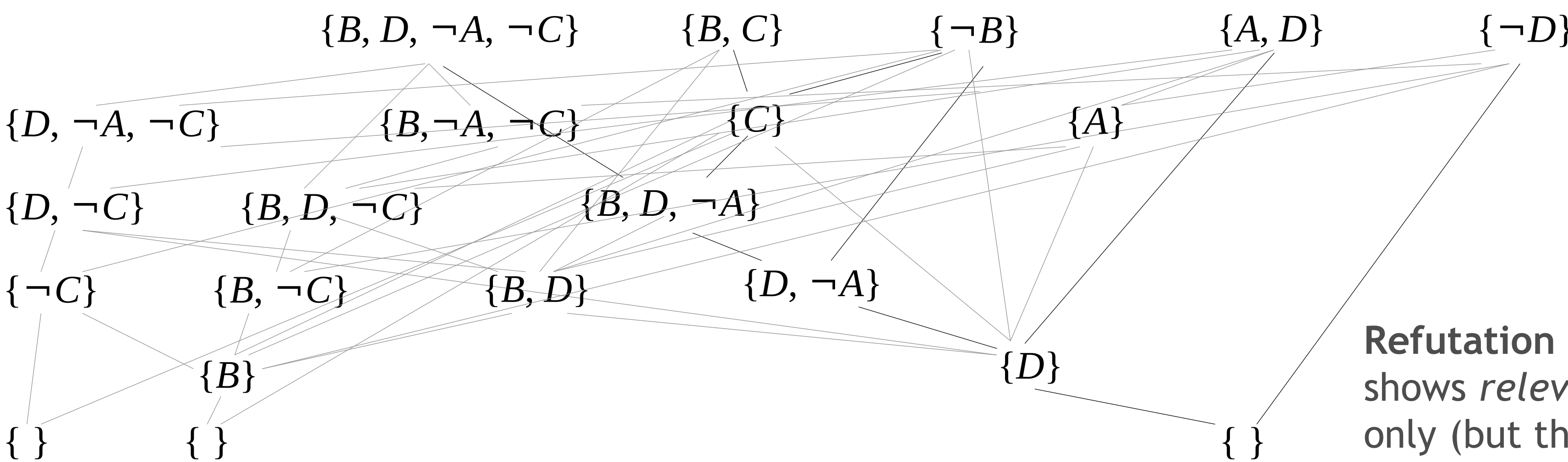
  Refutation + rewrite in CNF:

  $B \lor D \lor \neg A \lor \neg C, B \lor C,\ A \lor D,\ \neg B, \neg D$

  Rewrite in CF:

  $\{B, D, \neg A, \neg C\}, \{B, C\}, \{A, D\}, \{\neg B\}, \{\neg D\}$

  Applying the resolution rule:



Refutation graph:
shows *relevant* resolutions only (but there are more)

Resolution by refutation

- Resolution by refutation for propositional logic

  Is correct: $\Gamma \vdash \varphi \;\Rightarrow\; \Gamma \models \varphi$

  Is complete: $\Gamma \models \varphi \;\Rightarrow\; \Gamma \vdash \varphi$

  In this sense: if $\Gamma \models \varphi$ then there exists a refutation graph

- Algorithm

  It is a decision procedure for the problem $\Gamma \models \varphi$

  It has time complexity $O(2^n)$

  where n is the number of propositional symbols in $\Gamma \cup \{\neg\varphi\}$