Entailment and Algorithms

Marco Piastra

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 which there exists an algorithm, more precisely 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 with a specific input,
  is it possible to tell if whether it will eventually halt or run forever?

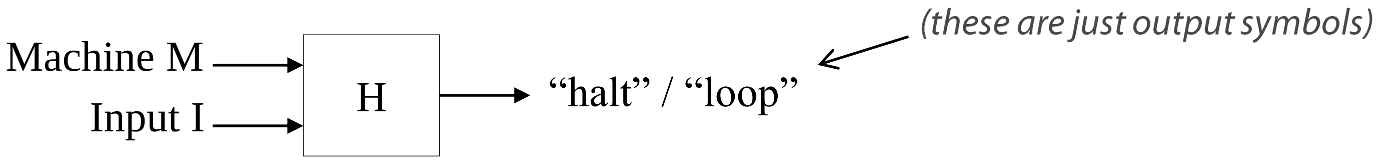
  In other words, does it exist a Turing machine that, given in input the description of *another* Turing machine, will always produce the answer desired?

  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)

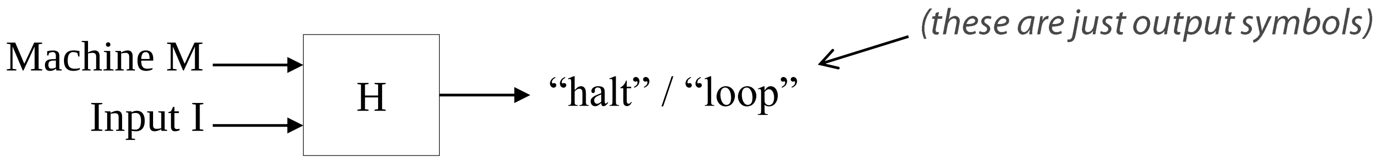
  Let's assume there exists a Turing machine H that, given the description of a Turing machine M with input I always terminates producing an output "halt" or "loop" depending on whether M with input I will terminate or not



An aside: The *Halting Problem*

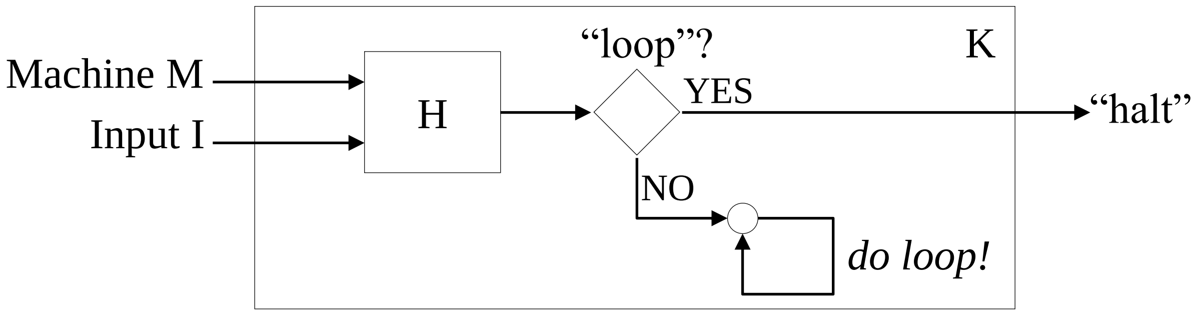
- Intuitive ideas behind the proof (i.e. of the *undecidability* of this problem)

  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</u> <u>H</u> <u>existed</u>

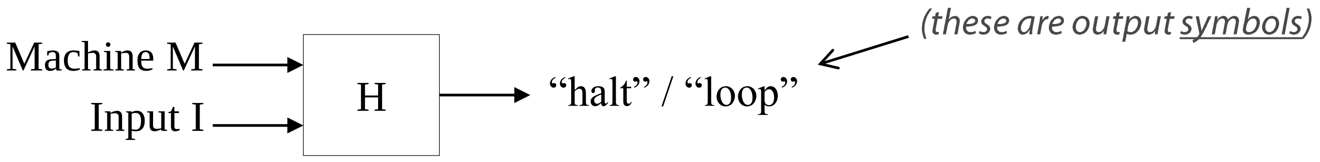
  We could build another Turing machine K that enters an infinite loop whenever the output of H is "halt" and that terminates, with output "halt", when H outputs "loop"



An aside: The *Halting Problem*

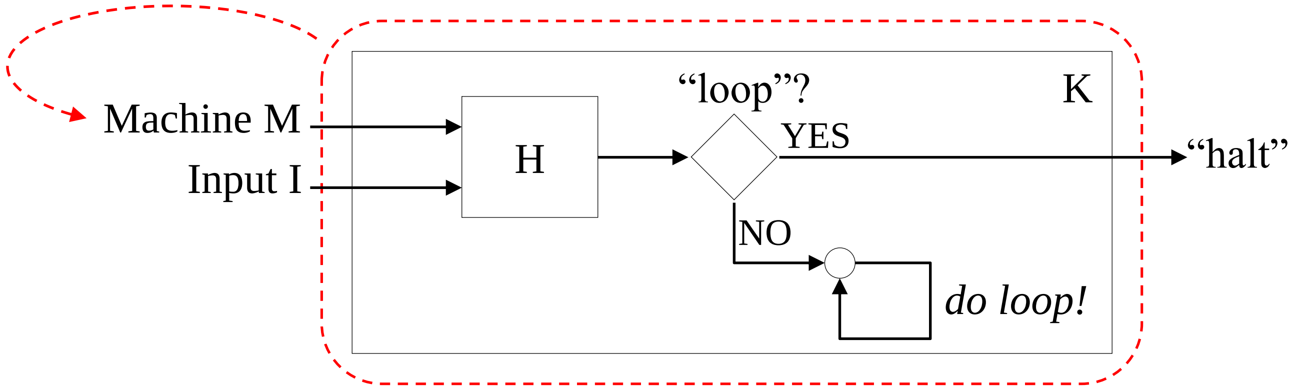
- Intuitive ideas behind the proof (i.e. of the *undecidability* of this problem)

  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"

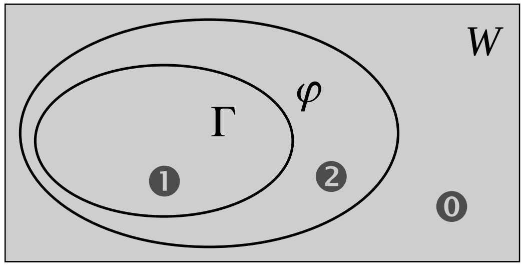


  What will be the output of K when given K *itself* as the input?
  K should *diverge* when K *terminates* and vice-versa: i.e. we have an absurdity

Transforming problems: entailment as satisfiability

- 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

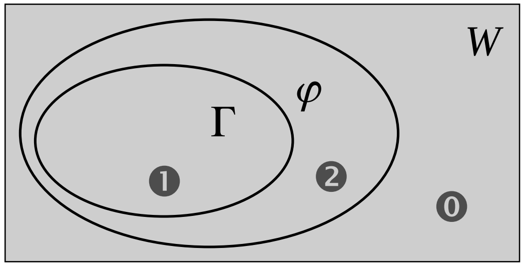
$w(\Gamma \cup \{\neg\varphi\}) = w(\Gamma) \cap w(\{\neg\varphi\})$

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

Transforming problems: entailment as satisfiability

- 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 = ∅

- 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

Satisfiability and decidability (in L_P)

- Is the decision problem "is the wff φ satisfiable?" <u>*decidable*</u>?

  It can be transformed into a *search* problem

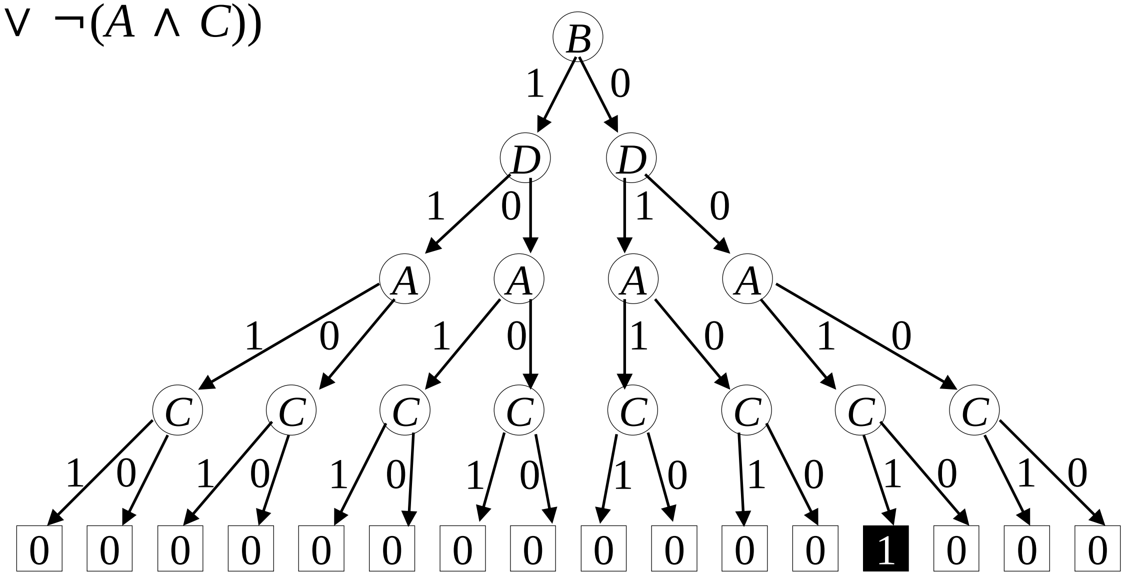
  i.e. finding a possible world (in the set of all possible worlds) that satisfies φ

  In the scientific literature, this problem is called "SAT"

  Intuition: we can try every possible value assignment for the atoms in φ

  Example:

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



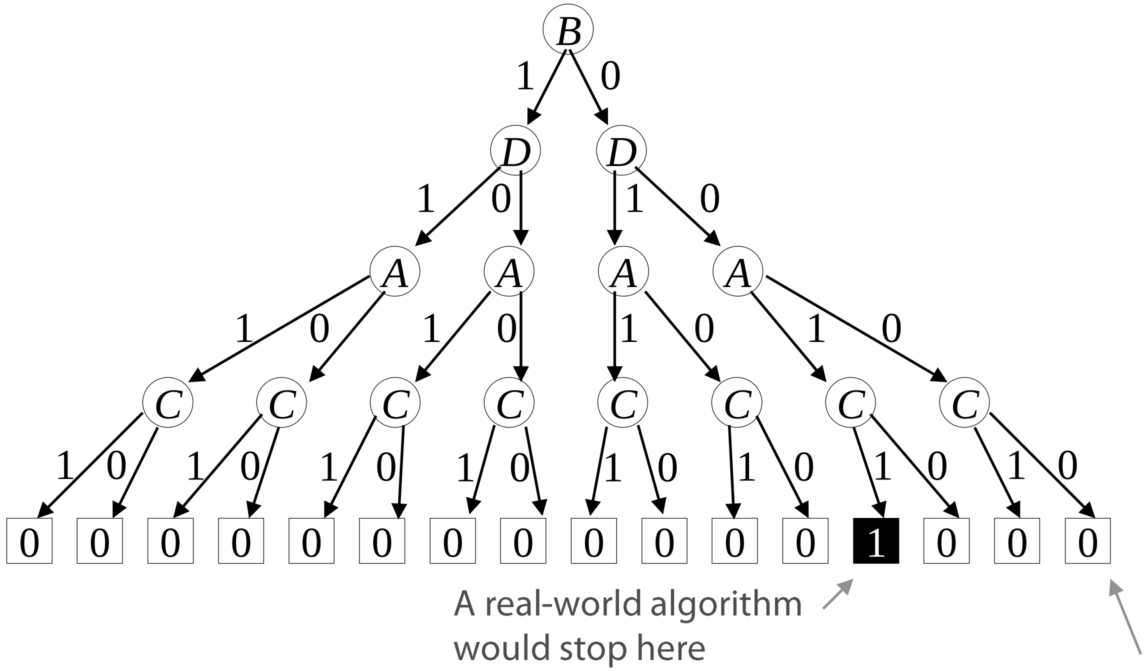
This method $O(2^n)$ time complexity, due to the number of value assignments

Satisfiability and decidability (in L_P)

Example:

$\neg(B \wedge D \wedge \neg(A \wedge C))$ which is equivalent to
$(\neg B \vee \neg D \vee (A \wedge C))$

Each branch in the tree represents a possible assignment:



The same algorithm is forced to try all possible assignments when ψ is *not* satisfiable.
For instance: $(\neg B \wedge \neg D \wedge \neg A \wedge \neg C)$

Computational complexity, classes P and NP

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 for the specific problem)

- 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

- 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, $\mathrm{P} \subseteq \mathrm{NP}$)

Class NP-complete and 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*)

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)
$$\begin{tikzcd}\neg(\neg\varphi) \arrow[d, dash] \\ \varphi\end{tikzcd}$$

(a2)
$$\begin{tikzcd}\varphi \land \psi \arrow[d, dash] \\ \varphi, \psi\end{tikzcd}$$

(a3)
$$\begin{tikzcd}\neg(\varphi \lor \psi) \arrow[d, dash] \\ \neg\varphi, \neg\psi\end{tikzcd}$$

(a4)
$$\begin{tikzcd}\neg(\varphi \to \psi) \arrow[d, dash] \\ \varphi, \neg\psi\end{tikzcd}$$

- Beta rules (i.e. bifurcation)

(b1)
$$\begin{tikzcd}& \varphi \lor \psi \arrow[dl, dash] \arrow[dr, dash] & \\ \varphi & & \psi\end{tikzcd}$$

(b2)
$$\begin{tikzcd}& \neg(\varphi \land \psi) \arrow[dl, dash] \arrow[dr, dash] & \\ \neg\varphi & & \neg\psi\end{tikzcd}$$

(b3)
$$\begin{tikzcd}& \varphi \to \psi \arrow[dl, dash] \arrow[dr, dash] & \\ \neg\varphi & & \psi\end{tikzcd}$$

(b4)
$$\begin{tikzcd}& \varphi \leftrightarrow \psi \arrow[dl, dash] \arrow[dr, dash] & \\ \neg\varphi, \neg\psi & & \varphi, \psi\end{tikzcd}$$

(b5)
$$\begin{tikzcd}& \neg(\varphi \leftrightarrow \psi) \arrow[dl, dash] \arrow[dr, dash] & \\ \neg\varphi, \psi & & \varphi, \neg\psi\end{tikzcd}$$

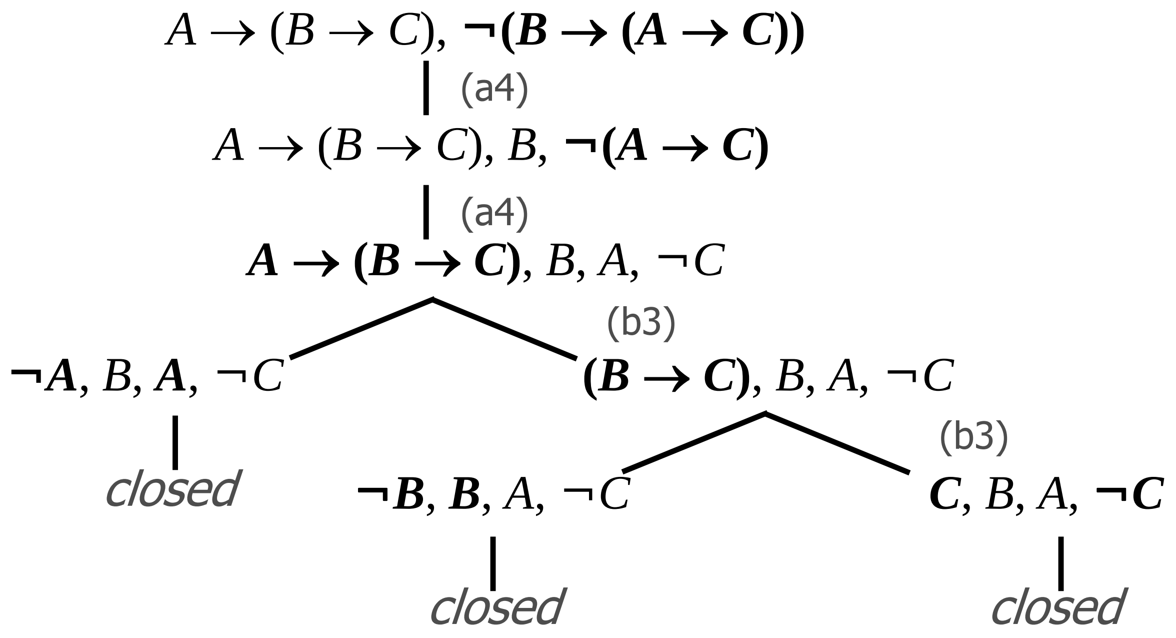
Semantic Tableau – a working example

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

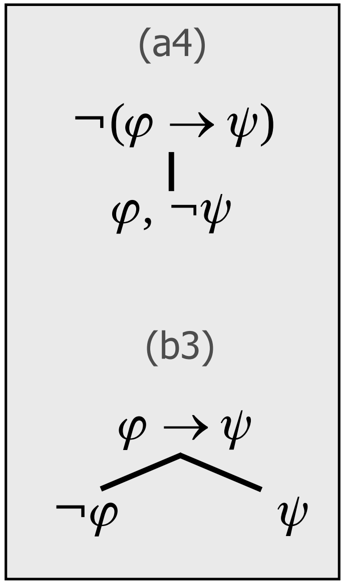
  Example input: $A \to (B \to C) \models B \to (A \to C)$?

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

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



$A \to (B \to C)$, $\neg(B \to (A \to C))$
(a4)
$A \to (B \to C)$, B, $\neg(A \to C)$
(a4)
$A \to (B \to C)$, B, A, $\neg C$
(b3)
$\neg A$, B, A, $\neg C$
closed
$(B \to C)$, B, A, $\neg C$
(b3)
$\neg B$, B, A, $\neg C$
closed
C, B, A, $\neg C$
closed



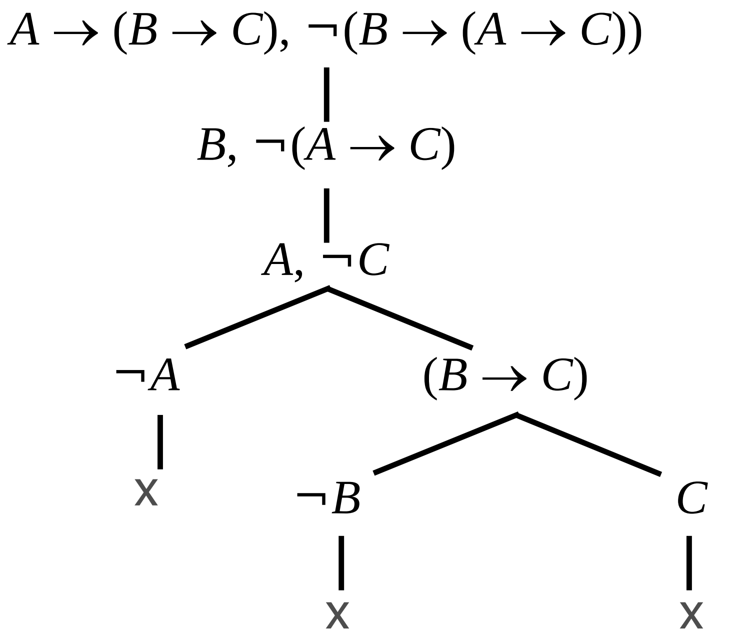
Semantic Tableau – a working example

- Original problem: “ $\Gamma \models \varphi$? ”

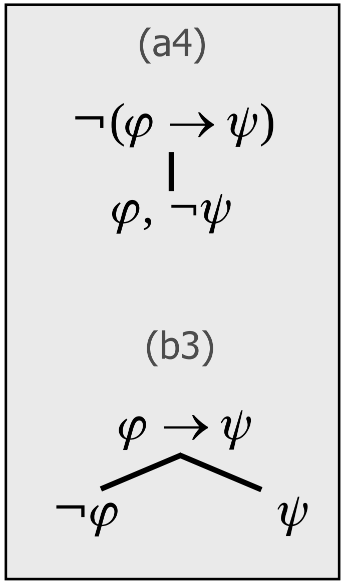
  Example input: $A \rightarrow (B \rightarrow C) \models B \rightarrow (A \rightarrow C)$?

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

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



$A \rightarrow (B \rightarrow C), \neg(B \rightarrow (A \rightarrow C))$
$B, \neg(A \rightarrow C)$
$A, \neg C$
$\neg A$
$(B \rightarrow C)$
x
$\neg B$
C
x
x



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):

  $$A \wedge B \wedge C \wedge \neg A$$

  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):

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

  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)