SAPIENZA
UNIVERSITÀ DI ROMA

Matteo D'Agostino
Mirko Magistrado Dacara

Sapienza University of Rome
academic year 2022/23

UNDECIDABILITY OF LOGICAL IMPLICATION IN FIRST-ORDER LOGIC

Index

Part I

- Historical excursus
- *Entscheidungsproblem*
- Turing Machines (TM)
- Formalization of a generic TM in FOL: from the computational model to the formulas encoding

Part II

- Satisfiability in propositional logic and FOL
- From *theorem proving* to satisfiability
- *Halting problem*
- Proof of the first-order logic undecidability via reduction from the halting problem

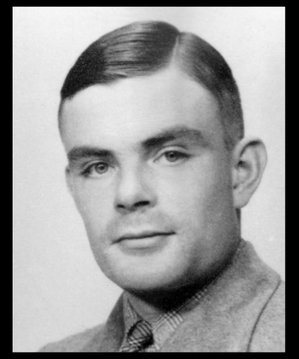
HISTORICAL EXCURSUS

LEIBNITZ (1670)

- Leibnizian ”Calculemus!”
- The need of a ”universal characteristic”

HILBERT (1900)

- The ”Hilbert problems”
- Entscheidungsproblem: the main problem of mathematical logic



TURING, CHURCH(1936)

- Turing Machines and λ-calculus
- A broken dream with related limitations

THE PROBLEM...

Let S be a logical system and ϕ one of its formulas. Is there an algorithm that can determine whether ϕ is a theorem of S?

- **Theorems and proofs in Formal Proof Systems**
- **Inference (syntax) vs Logical implication (semantics)**
- **Soundness and Completeness for "Proof systems"**
- **Gödel's completeness theorem (1929)**

A FOL formula is logically valid if and only if it is the conclusion of a formal deduction
(provability = validity)

... REFORMULATED

QUESTION

Is there an algorithm that can determine whether or not an FOL formula is valid (or also unsatisfiable by Refutation Principle)?

... REFORMULATED

QUESTION

Is there an algorithm that can determine whether or not an FOL formula is valid (or also unsatisfiable by Refutation Principle)?

ANSWER

No. It will be the purpose of the following slides to show that the set of satisfiable FOL sentences is undecidable, thus its characteristic function not computable

TURING MACHINE

$$(\mathbf{Q}, \mathbf{\Sigma}, \delta, \mathbf{q_0}, \mathbf{q_1})$$

- $\mathbf{Q}$ is the finite set of the states, $\mathbf{q_0} \in \mathbf{Q}$ final state and $\mathbf{q_1} \in \mathbf{Q}$ initial state
- $\mathbf{\Sigma}$ is a finite set of symbols called *alphabet*
- δ is the so-called *transition* ***function***, defined as follows:

$$\delta : \mathbf{Q} \setminus \{\mathbf{q_0}\} \times \mathbf{Q} \times \mathbf{\Sigma} \times \{\mathbf{L}, \mathbf{C}, \mathbf{R}\}$$

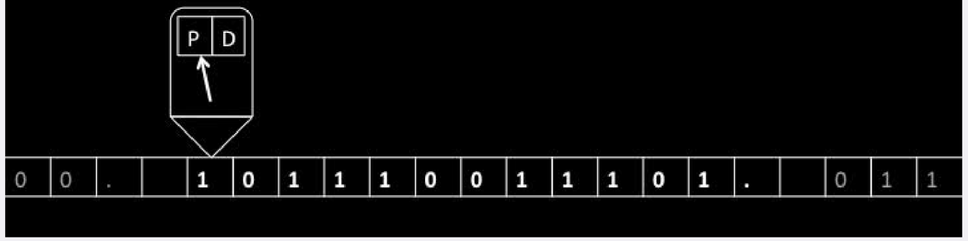
TURING MACHINE

$$(\mathbf{Q}, \mathbf{\Sigma}, \delta, \mathbf{q_0}, \mathbf{q_1})$$

- $\mathbf{Q}$ is the finite set of the states, $\mathbf{q_0} \in \mathbf{Q}$ final state and $\mathbf{q_1} \in \mathbf{Q}$ initial state
- $\mathbf{\Sigma}$ is a finite set of symbols called *alphabet*
- δ is the so-called *transition* ***function***, defined as follows:

$$\delta : \mathbf{Q} \setminus \{\mathbf{q_0}\} \times \mathbf{Q} \times \mathbf{\Sigma} \times \{\mathbf{L}, \mathbf{C}, \mathbf{R}\}$$

Stato iniziale	Simbolo letto	Simbolo scritto	Sposta	Stato finale
P	1	_	→	D
P	0	_	→	P
P	.	P	_	P
D	1	_	→	P
D	0	_	→	D
D	.	D	_	D



CONFIGURATION

A configuration (*complete configuration*) of a TM is an ordered triple $(\mathbf{x}, \mathbf{q}, \mathbf{t}) \in \mathbf{\Sigma}^* \times \mathbf{Q} \times \mathbf{N}$. At each time $\mathbf{t} \in \mathbf{N}$ the Turing machine will result in one and only one configuration.

CONFIGURATION

A configuration (*complete configuration*) of a TM is an ordered triple $(\mathbf{x}, \mathbf{q}, \mathbf{t}) \in \mathbf{\Sigma}^* \times \mathbf{Q} \times \mathbf{N}$. At each time $\mathbf{t} \in \mathbf{N}$ the Turing machine will result in one and only one configuration.

How, then, is it possible to go about formalizing such a concept in first-order Logic since the latter does not model discrete time by design (as it happens in Linear Temporal Logic)?

CONFIGURATION

A configuration (*complete configuration*) of a TM is an ordered triple $(\mathbf{x}, \mathbf{q}, \mathbf{t}) \in \mathbf{\Sigma}^* \times \mathbf{Q} \times \mathbf{N}$. At each time $\mathbf{t} \in \mathbf{N}$ the Turing machine will result in one and only one configuration.

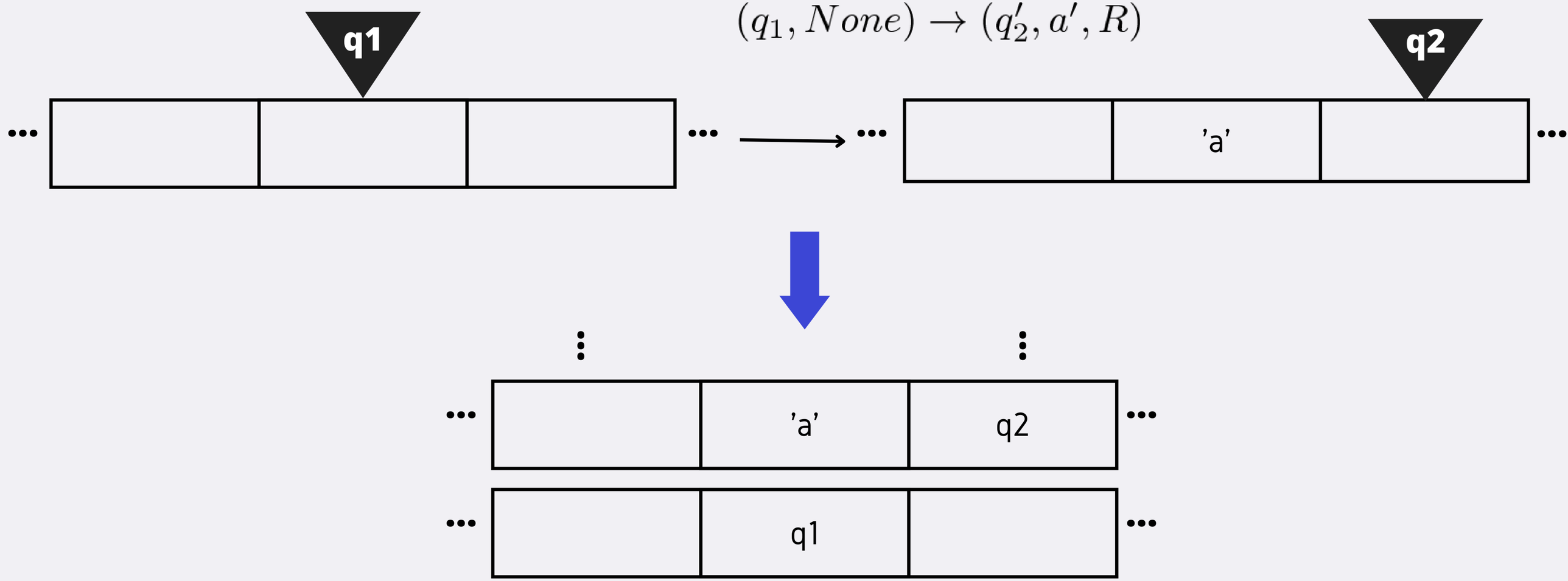
How, then, is it possible to go about formalizing such a concept in first-order Logic since the latter does not model discrete time by design (as it happens in Linear Temporal Logic)?

By considering the execution of a TM as a sequence of stacked configurations such that for each pair of adjacent configurations there exists the corresponding transition in the definition of the TM.

CONFIGURATION

If, for example, we had the following transition rule in δ:

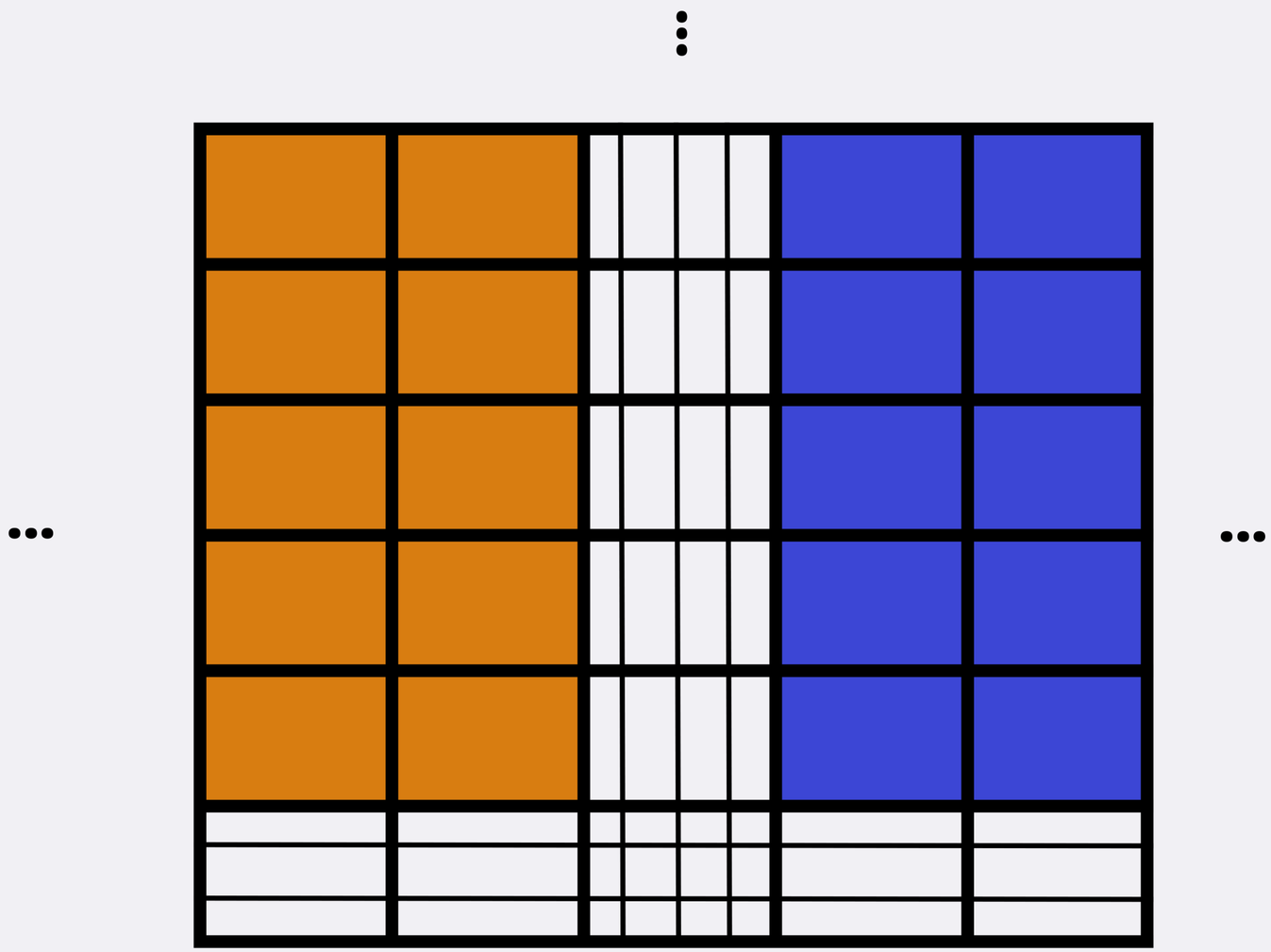
$$(q_1, None) \rightarrow (q_2', a', R)$$



FROM MT TO FOL

Constants	Functions/arity	Predicates/arity
origin	north/1 east/1 west/1	bottom/1 y-axis/1 left-side/1 right-side/1 head/1 S0, … , Sj / 1 Q0, … , Qk / 1

INTERPRETATION OF THE MODEL



{x : bottom(x)}

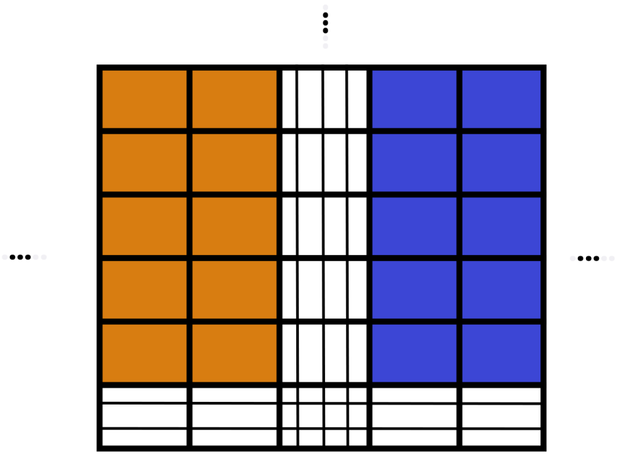
{x : y-axis(x)}

{x : left-side(x)}

{x : right-side(x)}

origin

FORMULAS FOR THE TAPE



$$\mathsf{bottom}(\mathsf{origin})$$

$$(\forall x)(\mathsf{bottom}(x) \leftrightarrow ((\mathsf{origin} = x) \vee (\exists y)(\mathsf{bottom}(y) \wedge \mathsf{east}(y) = x))$$

$$(\forall x)(\mathsf{bottom}(x) \leftrightarrow ((\mathsf{origin} = x) \vee (\exists y)(\mathsf{bottom}(y) \wedge \mathsf{west}(y) = x))$$

$$(\forall x, y)(\mathsf{north}(x) = \mathsf{north}(y) \rightarrow x = y)$$

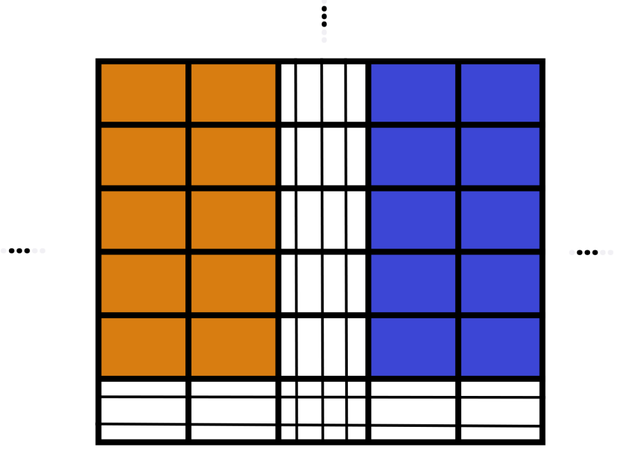
FORMULAS FOR THE TAPE

$$(\forall x)\neg \text{bottom}(\text{north}(x))$$

$$(\forall x)(\text{north}(\text{east}(x)) = \text{east}(\text{north}(x)))$$

$$(\forall x)((\text{west}(\text{east}(x)) = x) \wedge (x = \text{east}(\text{west}(x))))$$

$$(\forall x)(\text{left-side}(x) \vee \text{right-side}(x) \vee \text{y-axis}(x))$$



$$\text{right-side}(\text{east}(\text{origin})) \wedge \text{left-side}(\text{west}(\text{origin})) \wedge \neg\text{right-side}(\text{origin}) \wedge \neg\text{left-side}(\text{origin})$$

$$(\forall x)(\text{y-axis}(x) \rightarrow \neg(\text{left-side}(x) \vee \text{right-side}(x))$$

$$(\forall x)(\text{left-side}(x) \rightarrow \text{left-side}(\text{west}(x)))$$

$$(\forall x)(\text{right-side}(x) \rightarrow \text{right-side}(\text{east}(x)))$$

FORMULAS FOR A GENERIC MT

$$(\forall x) \bigvee_{i=0}^{j} \left(S_i(x) \wedge \bigwedge_{k \neq i} \neg S_k(x) \right)$$

$$(\forall x)(\mathsf{bottom}(x) \rightarrow (S_0(x) \wedge (x = \mathsf{origin} \leftrightarrow \mathsf{head}(x)))) \wedge Q_1(\mathsf{origin}) \wedge \bigwedge_{k \neq 1} \neg Q_k(\mathsf{origin})$$

$$(\forall x)\ \neg \mathbf{head}(x) \rightarrow (\bigvee_{i=0}^{j} (S_i(x) \leftrightarrow S_i(\mathbf{north}(x))))$$

FORMULAS FOR A SPECIFIC MT

$(s_j, q_i) \Rightarrow (s_b, L, q_d)$

$(\forall x)((\mathsf{head}(x) \wedge Q_i(x) \wedge S_j(x)) \rightarrow$
$(\mathsf{head}(\mathsf{north}(\mathsf{west}(x))) \wedge Q_d(\mathsf{north}(\mathsf{west}(x))) \wedge S_b(\mathsf{north}(x))))$

$(s_j, q_i) \Rightarrow (s_b, C, q_d)$

$(\forall x)((\mathsf{head}(x) \wedge Q_i(x) \wedge S_j(x)) \rightarrow (\mathsf{head}(\mathsf{north}(x)) \wedge Q_d(\mathsf{north}(x)) \wedge S_b(\mathsf{north}(x))))$

$(s_j, q_i) \Rightarrow (s_b, R, q_d)$

$(\forall x)((\mathsf{head}(x) \wedge Q_i(x) \wedge S_j(x)) \rightarrow$
$(\mathsf{head}(\mathsf{north}(\mathsf{east}(x))) \wedge Q_d(\mathsf{north}(\mathsf{east}(x))) \wedge S_b(\mathsf{north}(x))))$

FORMULAS FOR A SPECIFIC MT

$$(\forall x)(\mathsf{head}(x) \wedge Q_i(x) \wedge S_j(x)) \rightarrow$$
$$\Big[(x = \mathsf{origin}) \vee$$
$$(\exists y)\Big((\mathsf{north}(\mathsf{east}(y)) = x) \wedge \bigvee\nolimits_{(Q_a,S_b)\Rightarrow(Q_i,R,S_j)} \big((\mathsf{head}(y) \wedge Q_a(y) \wedge S_b(y))\big)$$
$$\vee \big((\mathsf{north}(y) = x) \wedge \bigvee\nolimits_{(Q_a,S_b)\Rightarrow(Q_i,C,S_j)} (\mathsf{head}(y) \wedge Q_a(y) \wedge S_b(y))\big)$$
$$\vee \big((\mathsf{north}(\mathsf{west}(y)) = x) \wedge \bigvee\nolimits_{(Q_a,S_b)\Rightarrow(Q_i,L,S_j)} (\mathsf{head}(y) \wedge Q_a(y) \wedge S_b(y))\big)\Big)\Big]$$

SATISFIABILITY IN PROPOSITIONAL LOGIC AND FOL

Is there an algorithm that, given a formula, checks whether it is satisfiable or not?

SATISFIABILITY IN PROPOSITIONAL LOGIC AND FOL

Is there an algorithm that, given a formula, checks whether it is satisfiable or not?

In **propositional logic**, yes:

- **TRUTH TABLES**
- **TABLEAUX**
- **DPLL**

SATISFIABILITY IN PROPOSITIONAL LOGIC AND FOL

Is there an algorithm that, given a formula, checks whether it is satisfiable or not?

In **propositional logic**, yes:

- **TRUTH TABLES**

  Naive, brute force

- **TABLEAUX**

  Brute force, better than truth tables

- **DPLL**

  The best in practice

SATISFIABILITY IN PROPOSITIONAL LOGIC AND FOL

Is there an algorithm that, given a formula, checks whether it is satisfiable or not?

In **propositional logic**, yes:

- **TRUTH TABLES**

  Naive, brute force

$$2^n$$

- **TABLEAUX**

  Brute force, better than truth tables

- **DPLL**

  The best in practice

SATISFIABILITY IN PROPOSITIONAL LOGIC AND FOL

Is there an algorithm that, given a formula, checks whether it is satisfiable or not?

In **first order logic**?

- 1890/1900 formalization of the theory of the natural numbers:

SATISFIABILITY IN PROPOSITIONAL LOGIC AND FOL

Is there an algorithm that, given a formula, checks whether it is satisfiable or not?

In **first order logic**?

- 1890/1900 formalization of the theory of the natural numbers:

  1) Can FOL be used to formalize mathematics?

SATISFIABILITY IN PROPOSITIONAL LOGIC AND FOL

Is there an algorithm that, given a formula, checks whether it is satisfiable or not?

In **first order logic**?

- 1890/1900 formalization of the theory of the natural numbers:

  1) Can FOL be used to formalize mathematics?

  2) Is there an algorithm for theorem proving? That is,
     Given a theory N and a theorem T, it is possible to verify:

$$N \models T \quad ?$$

FROM THEOREM PROVING TO SATISFIABILITY

From the properties of first-order logical implication:

Let Γ be a set of closed formulas, and A be a closed formula,
the REFUTATION PRINCIPLE (or PRINCIPLE OF PROOF BY CONTRADICTION) is valid.

$$\Gamma \models A \quad \text{iff} \quad \Gamma \cup \{\neg A\} \quad \text{unsatisfiable}$$

FROM THEOREM PROVING TO SATISFIABILITY

From the properties of first-order logical implication:

Let Γ be a set of closed formulas, and A be a closed formula,
the REFUTATION PRINCIPLE (or PRINCIPLE OF PROOF BY CONTRADICTION) is valid.

$$\Gamma \models A \quad \text{iff} \quad \Gamma \cup \{\neg A\} \quad \text{unsatisfiable}$$

So, in **first-order logic**,

Is there an algorithm that, given a formula, tests whether or not it is satisfiable, or equivalently, the logical implication problem is decidable?

FROM THEOREM PROVING TO SATISFIABILITY

NO

There is **NO** algorithm such that, **FOR EACH** formula ϕ in first-order logic, is able to check whether it is satisfiable or not

FROM THEOREM PROVING TO SATISFIABILITY

NO

There is **NO** algorithm such that, **FOR EACH** formula ϕ in first-order logic, is able to check whether it is satisfiable or not

THEOREM

The problem of checking whether a formula in FOL is satisfiable is UNDECIDABLE (especially SEMI-DECIDABLE)

FROM THEOREM PROVING TO SATISFIABILITY

NO

There is **NO** algorithm such that, **FOR EACH** formula ϕ in first-order logic, is able to check whether it is satisfiable or not

THEOREM

The problem of checking whether a formula in FOL is satisfiable is UNDECIDABLE (especially SEMI-DECIDABLE)

The proof is via reduction: the halting problem is reduced to undecidability of the FOL

What is a REDUCTION?

- Algorithm for transforming a problem **A** to a problem **B**
- Used for: upper/lower bound tests, **undecidability**
- Intuitively, problem A (known) is reducible to problem B (new problem), if an algorithm to solve B (if it exists) can be used to solve Aper risolvere B (se esiste) può essere usato per risolvere A, mapping the yes/no instances of A into the yes/no instances of B

Notation:

$$A \leq B$$

What is a REDUCTION?

Notation:

$$A \leq B$$

- Algorithm for transforming a problem **A** to a problem **B**
- Used for: upper/lower bound tests, **undecidability**
- Intuitively, problem A (known) is reducible to problem B (new problem), if an algorithm to solve B (if it exists) can be used to solve Aper risolvere B (se esiste) può essere usato per risolvere A, mapping the yes/no instances of A into the yes/no instances of B
    - If A ≤ B, solving A cannot be more difficult* than solving B, as a solution of B leads to a solution of A.
    - In terms of the theory of decidability, with A ≤ B:
        - if B is decidable, then A is also decidable
        - if A is undecidable, then B is also undecidable.

Reduction example:

- multiplication ≤ squaring

$$a \times b = \frac{\left((a+b)^2 - a^2 - b^2\right)}{2}$$

Assuming we can only add, subtract, square and divide by two, we can calculate the product between two numbers with this formula

HALTING PROBLEM - THEOREM

Given Σ an alphabet and a codification that associates each Turing machine $\mathbf{M}$ its description $\boldsymbol{c_M} \in \Sigma^*$.

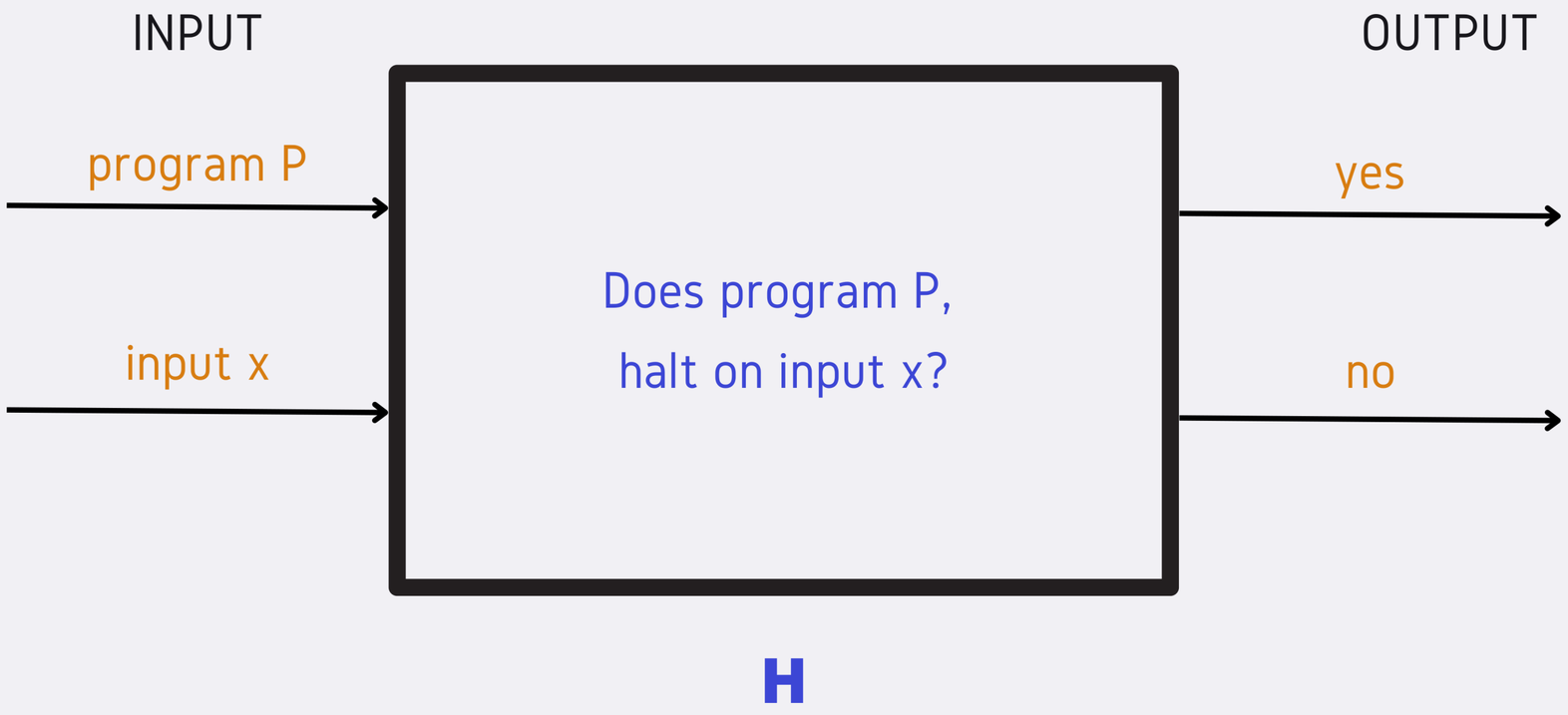
The function

$$h(c_M, x) = \begin{cases} 1 & \text{if M halts on input x} \\ 0 & \text{if M does not halt on input x} \end{cases}$$

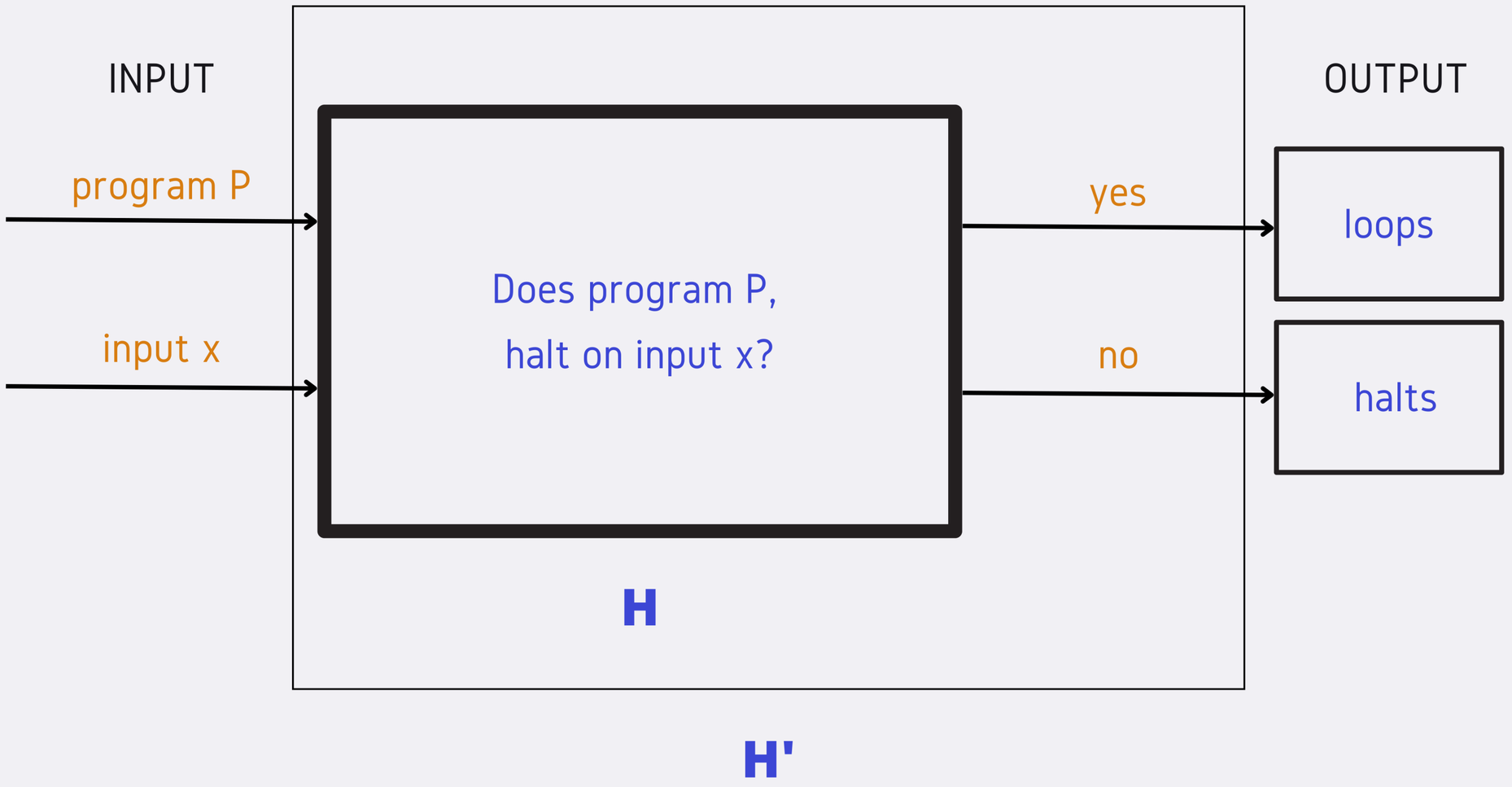
is not Turing computable

That is, ∄ a Turing machine that, ∀ Turing machine M, ∀ input x,
can tell whether or not M with input x, stops

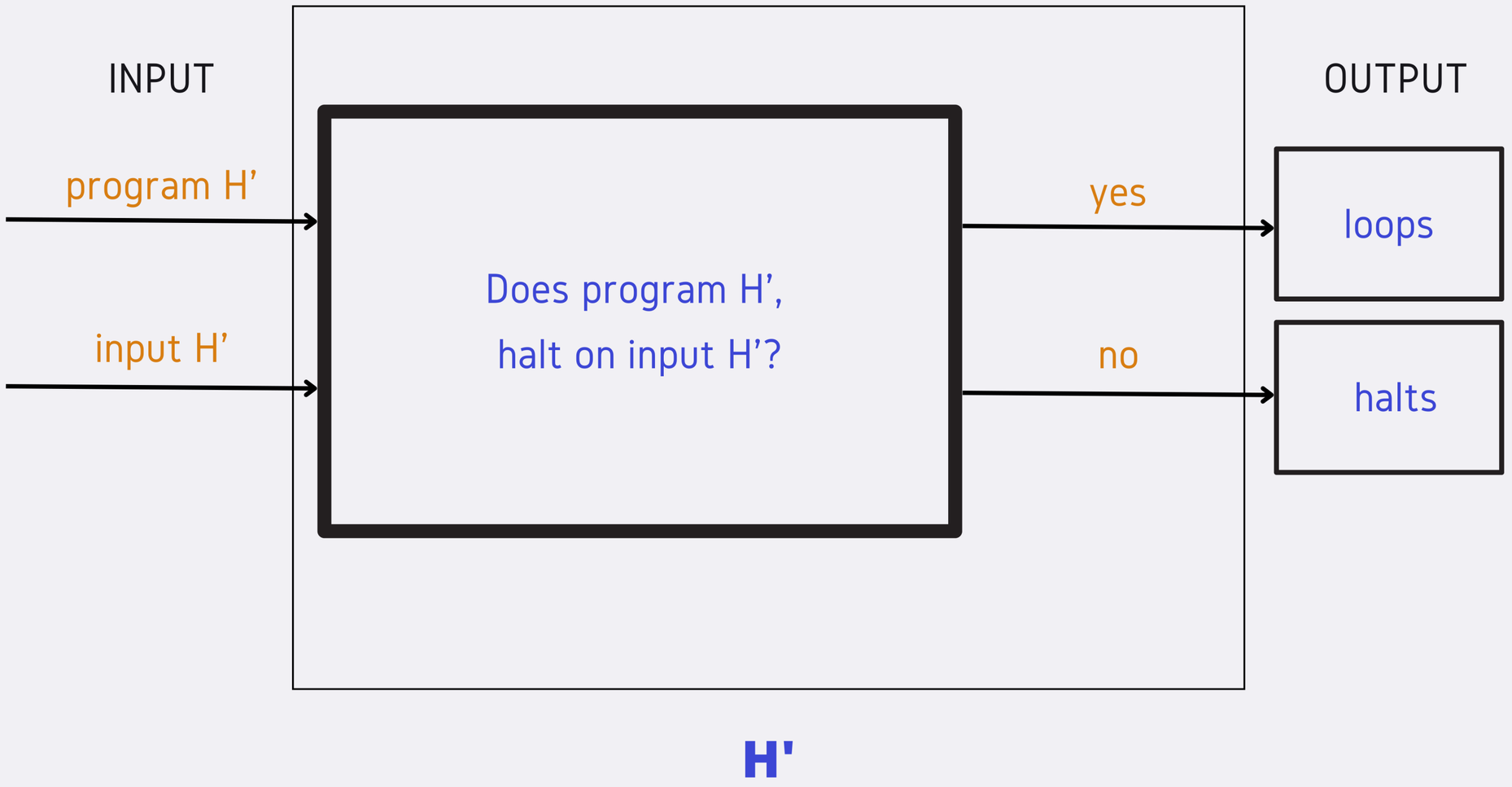
HALTING PROBLEM - PROOF



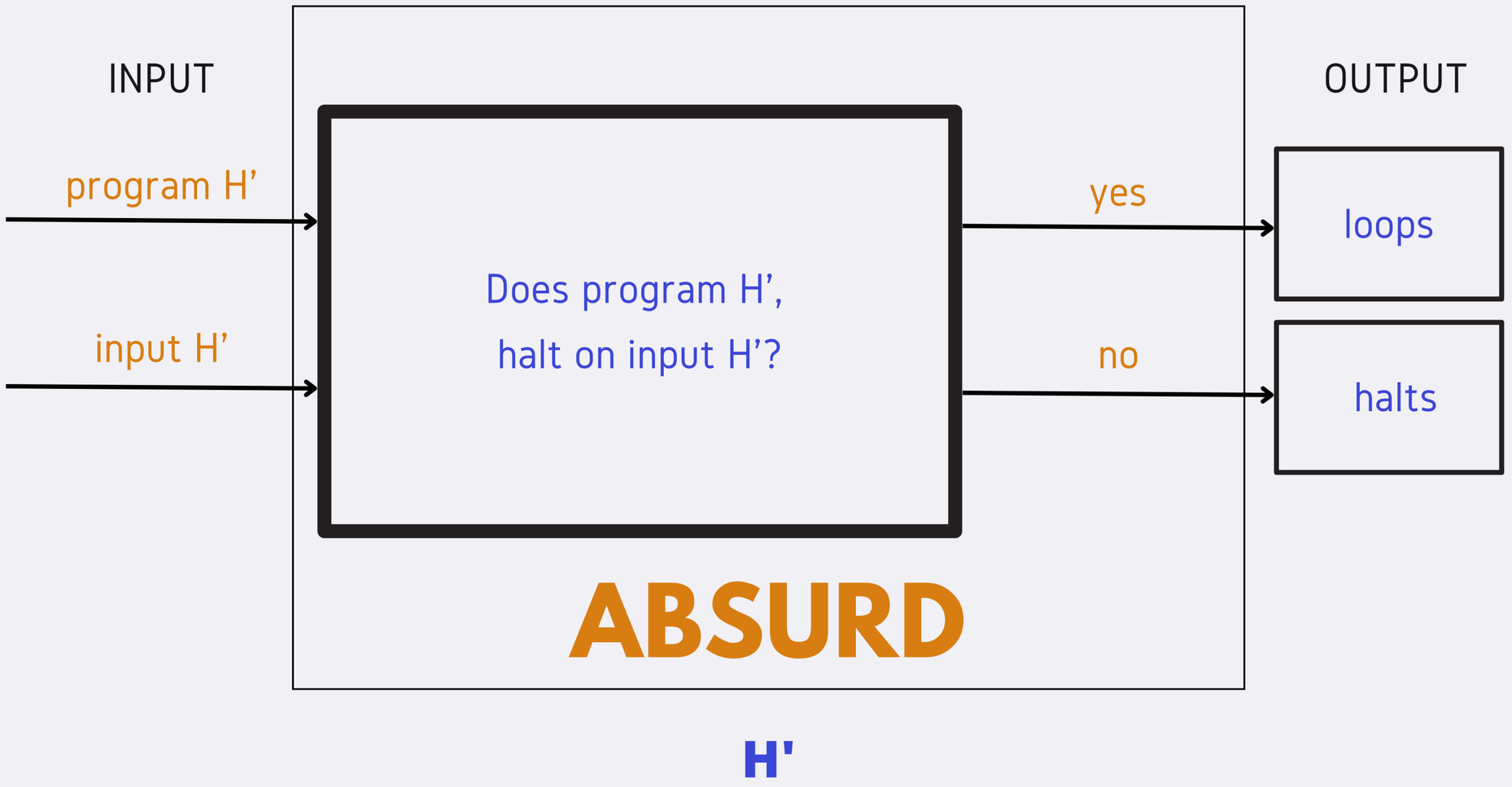
HALTING PROBLEM - PROOF



HALTING PROBLEM - PROOF



HALTING PROBLEM - PROOF



FROM THEOREM PROVING TO SATISFIABILITY

NO

There is **NO** algorithm such that, **FOR EACH** formula ϕ in first-order logic, is able to check whether it is satisfiable or not

THEOREM

The problem of checking whether a formula in FOL is satisfiable is UNDECIDABLE (especially SEMI-DECIDABLE)

The proof is via reduction: the halting problem is reduced to undecidability of the FOL

FIRST-ORDER LOGIC UNDECIDABILITY - PROOF

We will use the following reduction:

Halting problem ≤ Undecidability of logical implication in FOL

Let ϕ be the conjunction of formulas for the formalization of a Turing machine M, which has a finite number of transitions, so ϕ is also finite, hence a formula in FOL, and be w its input.

FIRST-ORDER LOGIC UNDECIDABILITY - PROOF

We will use the following reduction:

Halting problem ≤ Undecidability of logical implication in FOL

Let ϕ be the conjunction of formulas for the formalization of a Turing machine M, which has a finite number of transitions, so ϕ is also finite, hence a formula in FOL, and be w its input.

The formula ϕ expresses the existence of a sequence of configurations of M that follows its transition rules, (according to its transition function δ), thus corresponds to a computation of M on w.

FIRST-ORDER LOGIC UNDECIDABILITY - PROOF

We will use the following reduction:

Halting problem ≤ Undecidability of logical implication in FOL

Let ϕ be the conjunction of formulas for the formalization of a Turing machine M, which has a finite number of transitions, so ϕ is also finite, hence a formula in FOL, and be w its input.

The formula ϕ expresses the existence of a sequence of configurations of M that follows its transition rules, (according to its transition function δ), thus corresponds to a computation of M on w.

If M halts on w, then the formula ϕ will be verified (satisfiable), since a valid computation
If M does not halt on w, then the formula ϕ will be false (unsatisfiable), due to the absence of a valid computation

FIRST-ORDER LOGIC UNDECIDABILITY - PROOF

We will use the following reduction:

Halting problem ≤ Undecidability of logical implication in FOL

Let ϕ be the conjunction of formulas for the formalization of a Turing machine M, which has a finite number of transitions, so ϕ is also finite, hence a formula in FOL, and be w its input.

The formula ϕ expresses the existence of a sequence of configurations of M that follows its transition rules, (according to its transition function δ), thus corresponds to a computation of M on w.

If M halts on w, then the formula ϕ will be verified (satisfiable), since a valid computation
If M does not halt on w, then the formula ϕ will be false (unsatisfiable), due to the absence of a valid computation

Hence, using this algorithm for the satisfiability of ϕ we could solve the halting problem,
which is absurd since we know it is undecidable

BIBLIOGRAPHY

1. Catalani, L. (2018, April 23). Calculemus: Il sogno di Leibniz. Medium. Retrieved July 12, 2023, from https://medium.com/@luigicatalani/calculemus-il-sogno-di-leibniz-196b11a55766

2. Wikipedia contributors. (2023, July 11). Entscheidungsproblem. In Wikipedia, The Free Encyclopedia. Retrieved July 12, 2023, from https://en.wikipedia.org/wiki/Entscheidungsproblem

3. StackExchange User (2017, July 31). The Entscheidungsproblem and the notion of decidability in first-order logic. Mathematics Stack Exchange. Retrieved July 12, 2023, from https://math.stackexchange.com/questions/2346305/the-entscheidungsproblem-and-the-notion-of-decidability-in-first-order-logic

4. StackExchange User (2018, March 26). Semi-decidability of first-order logic. Mathematics Stack Exchange. Retrieved July 12, 2023, from https://math.stackexchange.com/questions/2680611/semi-decidability-of-first-order-logic

BIBLIOGRAPHY

5. StackExchange User (2015, June 8). How is first-order logic complete but not decidable? Philosophy Stack Exchange. Retrieved July 12, 2023, from https://philosophy.stackexchange.com/questions/15525/how-is-first-order-logic-complete-but-not-decidable

6. New Mexico State University. (n.d.). First Order Logic Undecidability. Retrieved July 12, 2023, from https://www.cs.nmsu.edu/historical-projects/Projects/FoLundecidability.pdf

7. Gries, D., & Schneider, F. B. (2002). First Order Logic. Stanford University. Retrieved July 12, 2023, from http://kilby.stanford.edu/~rvg/154/handouts/fol.html

8. F. D'Amore, G. Ausiello e G. Gambosi (2002). Linguaggi, modelli, complessità