Logic

Using First Order Logic

Enrico Franconi

franconi@inf.unibz.it
http://www.inf.unibz.it/~franconi

Faculty of Computer Science, Free University of Bozen-Bolzano

Reduction to satisfiability

Just like propositional logic:

- A formula ϕ is satisfiable iff there is some interpretation $\mathcal{I}$ that satisfies ϕ (i.e., ϕ is true under $\mathcal{I}$): $\mathcal{I} \models \phi$).
- Validity, equivalence, and entailment can be reduced to satisfiability:
  - ϕ is a valid (i.e., a tautology) iff $\neg\phi$ is not satisfiable.
  - ϕ is equivalent to ψ ($\phi \equiv \psi$) iff $\phi \leftrightarrow \psi$ is valid.
    - $\phi \equiv \psi$ iff $\phi \models \psi$ and $\psi \models \phi$
  - ϕ entails ψ ($\phi \models \psi$) iff $\phi \rightarrow \psi$ is valid (*deduction theorem*).
    - $\boxed{\phi \models \psi \text{ iff } \phi \wedge \neg\psi \text{ is not satisfiable}}$.
- We need a *sound and complete* procedure deciding satisfiability: the *tableaux method* is a decision procedure which checks the existence of a model.

Tableaux Calculus

Just like in propositional logic:

- The Tableaux Calculus is a decision procedure solving the problem of satisfiability.
- If a formula is satisfiable, the procedure will constructively exhibit a model of the formula.
- The basic idea is to incrementally build the model by looking at the formula, by decomposing it in a top/down fashion. The procedure exhaustively looks at all the possibilities, so that it can eventually prove that no model could be found for unsatisfiable formulas.

With respect to propositional logic, the notion of *model* is different.

Tableaux Calculus

Finds a model for a given collection of sentences *KB* in negation normal form.

1. Consider the knowledge base *KB* as the root node of a *refutation tree*. A node in a refutation tree is called *tableaux*.

2. Starting from the root, add new formulas to the tableaux, applying the *completion rules*.

3. Completion rules are either deterministic – they yield a uniquely determined successor node – or nondeterministic – yielding several possible alternative successor nodes (*branches*).

4. Apply the completion rules until either
   (a) an explicit contradiction due to the presence of two opposite ground literals in a node (a *clash*) is generated in each branch, or
   (b) there is a *completed* branch where no more rule is applicable (but. . .).

The Calculus

The completion rules for the propositional formulas:

$$\frac{\phi \wedge \psi}{\begin{array}{c}\phi \\ \psi\end{array}}$$

If a model satisfies a conjunction, then it also satisfies *each of* the conjuncts

$$\frac{\phi \vee \psi}{\phi \mid \psi}$$

If a model satisfies a disjunction, then it also satisfies *one of* the disjuncts. It is a non-deterministic rule, and it generates two *alternative* branches of the tableaux.

The Calculus

The completion rules for quantified formulas:

$$\frac{\forall x.\ \phi}{\phi\{X/t\}}$$

$$\forall x.\ \phi$$

If a model satisfies a universal quantified formula, the it also satisfies the formula where the quantified variable has been substituted with some term. The prescription is to use all the terms which appear in the tableaux.

$$\frac{\exists x.\ \phi}{\phi\{X/a\}}$$

If a model satisfies an existential quantified formula, then it also satisfies the formula where the quantified variable has been substituted with a fresh new *skolem* constant.

Models

- The completed branch of the refutation tree gives a model of *KB*: the *KB* is satisfiable. Since all formulas have been reduced to ground literals (i.e., either positive or negative atomic formulas which do not contain variables), it is possible to find an interpretation to predicates using the constants, which make all the sentences in the branch true.
- If there is no completed branch (i.e., every branch has a clash), then it is not possible to find an interpretation for the predicates making the original *KB* true: the *KB* is unsatisfiable. In fact, the original formulas from which the tree is constructed can not be true simultaneously.

Negation Normal Form

The above set of completion rules work only if the formula has been translated into Negation Normal Form, i.e., all the negations have been pushed down.

Example::

$$\neg(\exists x.\ [\ \forall y.\ [\ P(x) \rightarrow Q(y)\ ]\ ])$$

becomes

$$\forall x.\ [\ \exists y.\ [\ P(x) \wedge \neg Q(y)\ ]\ ]$$

(Why?)

Example

$\phi \wedge \psi$	$\phi \vee \psi$	$\forall x.\ \phi$	$\exists x.\ \phi$
ϕ	$\phi \mid \psi$	$\phi\{X/t\}$	$\phi\{X/a\}$
ψ			

$$\boxed{\exists y.\ (p(y) \wedge \neg q(y)) \wedge \forall z.\, (p(z) \vee q(z))}$$

Example

$\phi \wedge \psi$	$\phi \vee \psi$	$\forall x.\ \phi$	$\exists x.\ \phi$
ϕ	$\phi \mid \psi$	$\phi\{X/t\}$	$\phi\{X/a\}$
ψ			

$$\boxed{\exists y.\ (p(y) \wedge \neg q(y)) \wedge \forall z.\, (p(z) \vee q(z))}$$

$$\exists y.\ (p(y) \wedge \neg q(y))$$

$$\forall z.\ (p(z) \vee q(z))$$

Example

$\phi \wedge \psi$	$\phi \vee \psi$	$\forall x.\ \phi$	$\exists x.\ \phi$
ϕ	$\phi \mid \psi$	$\phi\{X/t\}$	$\phi\{X/a\}$
ψ			

$$\boxed{\exists y.\ (p(y) \wedge \neg q(y)) \wedge \forall z.\ (p(z) \vee q(z))}$$

$$\exists y.\ (p(y) \wedge \neg q(y))$$

$$\forall z.\ (p(z) \vee q(z))$$

$$p(\mathbf{a}) \wedge \neg q(\mathbf{a})$$

Example

$\phi \wedge \psi$	$\phi \vee \psi$		$\forall x.\ \phi$	$\exists x.\ \phi$
ϕ	ϕ	ψ	$\phi\{X/t\}$	$\phi\{X/a\}$
ψ				

$$\boxed{\exists y.\ (p(y) \wedge \neg q(y)) \wedge \forall z.\ (p(z) \vee q(z))}$$

$$\exists y.\ (p(y) \wedge \neg q(y))$$

$$\forall z.\ (p(z) \vee q(z))$$

$$p(\mathbf{a}) \wedge \neg q(\mathbf{a})$$

$$p(\mathbf{a})$$

$$\neg q(\mathbf{a})$$

Example

$\phi \wedge \psi$	$\phi \vee \psi$	$\forall x.\ \phi$	$\exists x.\ \phi$
ϕ	$\phi \mid \psi$	$\phi\{X/t\}$	$\phi\{X/a\}$
ψ			

$$\boxed{\exists y.\ (p(y) \wedge \neg q(y)) \wedge \forall z.\ (p(z) \vee q(z))}$$

$$\exists y.\ (p(y) \wedge \neg q(y))$$

$$\forall z.\ (p(z) \vee q(z))$$

$$p(\mathbf{a}) \wedge \neg q(\mathbf{a})$$

$$p(\mathbf{a})$$

$$\neg q(\mathbf{a})$$

$$p(\mathbf{a}) \vee q(\mathbf{a})$$

Example

$\phi \wedge \psi$	$\phi \vee \psi$	$\forall x.\ \phi$	$\exists x.\ \phi$
ϕ	$\phi \mid \psi$	$\phi\{X/t\}$	$\phi\{X/a\}$
ψ			

$$\boxed{\exists y.\ (p(y) \wedge \neg q(y)) \wedge \forall z.\ (p(z) \vee q(z))}$$

$$\exists y.\ (p(y) \wedge \neg q(y))$$

$$\forall z.\ (p(z) \vee q(z))$$

$$p(\mathbf{a}) \wedge \neg q(\mathbf{a})$$

$$p(\mathbf{a})$$

$$\neg q(\mathbf{a})$$

$$p(\mathbf{a}) \vee q(\mathbf{a})$$

$$p(\mathbf{a})$$

< COMPLETED >

Example

$\phi \wedge \psi$	$\phi \vee \psi$	$\forall x.\ \phi$	$\exists x.\ \phi$
ϕ	$\phi \mid \psi$	$\phi\{X/t\}$	$\phi\{X/a\}$
ψ			

$$\boxed{\exists y.\ (p(y) \wedge \neg q(y)) \wedge \forall z.\ (p(z) \vee q(z))}$$

$$\exists y.\ (p(y) \wedge \neg q(y))$$

$$\forall z.\ (p(z) \vee q(z))$$

$$p(\mathbf{a}) \wedge \neg q(\mathbf{a})$$

$$p(\mathbf{a})$$

$$\neg q(\mathbf{a})$$

$$p(\mathbf{a}) \vee q(\mathbf{a})$$

$p(\mathbf{a})$ $\qquad$ $q(\mathbf{a})$

$<$ COMPLETED $>$ $\qquad$ $<$ CLASH $>$

Example

$\phi \wedge \psi$	$\phi \vee \psi$	$\forall x.\ \phi$	$\exists x.\ \phi$
ϕ	$\phi \mid \psi$	$\phi\{X/t\}$	$\phi\{X/a\}$
ψ			

$$\boxed{\exists y.\ (p(y) \wedge \neg q(y)) \wedge \forall z.\ (p(z) \vee q(z))}$$

$$\exists y.\ (p(y) \wedge \neg q(y))$$

$$\forall z.\ (p(z) \vee q(z))$$

$$p(\mathbf{a}) \wedge \neg q(\mathbf{a})$$

$$p(\mathbf{a})$$

$$\neg q(\mathbf{a})$$

$$p(\mathbf{a}) \vee q(\mathbf{a})$$

$$p(\mathbf{a}) \qquad\qquad q(\mathbf{a})$$

$$< \texttt{COMPLETED} > \qquad\qquad < \texttt{CLASH} >$$

The formula is satisfiable. The devised model is $\Delta = \{\mathbf{a}\},\ p^{\mathcal{I}} = \{\mathbf{a}\},\ q^{\mathcal{I}} = \emptyset$.

Example

$\phi \wedge \psi$	$\phi \vee \psi$	$\forall x.\ \phi$	$\exists x.\ \phi$
ϕ	$\phi \mid \psi$	$\phi\{X/t\}$	$\phi\{X/a\}$
ψ			

$$\boxed{\exists y.\ (p(y) \wedge \neg q(y)) \wedge \forall z.\, (\neg p(z) \vee q(z))}$$

Example

$\phi \wedge \psi$	$\phi \vee \psi$	$\forall x.\ \phi$	$\exists x.\ \phi$
ϕ	$\phi \mid \psi$	$\phi\{X/t\}$	$\phi\{X/a\}$
ψ			

$$\boxed{\exists y.\ (p(y) \wedge \neg q(y)) \wedge \forall z.\, (\neg p(z) \vee q(z))}$$

$$\exists y.\ (p(y) \wedge \neg q(y))$$

$$\forall z.\ (\neg p(z) \vee q(z))$$

Example

$\phi \wedge \psi$	$\phi \vee \psi$	$\forall x.\ \phi$	$\exists x.\ \phi$
ϕ	$\phi \mid \psi$	$\phi\{X/t\}$	$\phi\{X/a\}$
ψ			

$$\boxed{\exists y.\ (p(y) \wedge \neg q(y)) \wedge \forall z.\, (\neg p(z) \vee q(z))}$$

$$\exists y.\ (p(y) \wedge \neg q(y))$$

$$\forall z.\ (\neg p(z) \vee q(z))$$

$$p(\mathbf{a}) \wedge \neg q(\mathbf{a})$$

Example

$\phi \wedge \psi$	$\phi \vee \psi$	$\forall x.\ \phi$	$\exists x.\ \phi$
ϕ	$\phi \mid \psi$	$\phi\{X/t\}$	$\phi\{X/a\}$
ψ			

$$\boxed{\exists y.\ (p(y) \wedge \neg q(y)) \wedge \forall z.\, (\neg p(z) \vee q(z))}$$

$$\exists y.\ (p(y) \wedge \neg q(y))$$

$$\forall z.\ (\neg p(z) \vee q(z))$$

$$p(\mathbf{a}) \wedge \neg q(\mathbf{a})$$

$$p(\mathbf{a})$$

$$\neg q(\mathbf{a})$$

Example

$\phi \wedge \psi$	$\phi \vee \psi$	$\forall x.\ \phi$	$\exists x.\ \phi$
ϕ	$\phi \mid \psi$	$\phi\{X/t\}$	$\phi\{X/a\}$
ψ			

$$\boxed{\exists y.\ (p(y) \wedge \neg q(y)) \wedge \forall z.\, (\neg p(z) \vee q(z))}$$

$$\exists y.\ (p(y) \wedge \neg q(y))$$

$$\forall z.\ (\neg p(z) \vee q(z))$$

$$p(\mathbf{a}) \wedge \neg q(\mathbf{a})$$

$$p(\mathbf{a})$$

$$\neg q(\mathbf{a})$$

$$\neg p(\mathbf{a}) \vee q(\mathbf{a})$$

Example

$\phi \wedge \psi$	$\phi \vee \psi$	$\forall x.\ \phi$	$\exists x.\ \phi$
ϕ	$\phi \mid \psi$	$\phi\{X/t\}$	$\phi\{X/a\}$
ψ			

$$\boxed{\exists y.\ (p(y) \wedge \neg q(y)) \wedge \forall z.\, (\neg p(z) \vee q(z))}$$

$$\exists y.\ (p(y) \wedge \neg q(y))$$

$$\forall z.\ (\neg p(z) \vee q(z))$$

$$p(\mathbf{a}) \wedge \neg q(\mathbf{a})$$

$$p(\mathbf{a})$$

$$\neg q(\mathbf{a})$$

$$\neg p(\mathbf{a}) \vee q(\mathbf{a})$$

$$\neg p(\mathbf{a})$$

$<$ CLASH $>$

Example

$\phi \wedge \psi$	$\phi \vee \psi$	$\forall x.\ \phi$	$\exists x.\ \phi$
ϕ	$\phi \mid \psi$	$\phi\{X/t\}$	$\phi\{X/a\}$
ψ			

$$\boxed{\exists y.\ (p(y) \wedge \neg q(y)) \wedge \forall z.\, (\neg p(z) \vee q(z))}$$

$$\exists y.\ (p(y) \wedge \neg q(y))$$

$$\forall z.\ (\neg p(z) \vee q(z))$$

$$p(\mathbf{a}) \wedge \neg q(\mathbf{a})$$

$$p(\mathbf{a})$$

$$\neg q(\mathbf{a})$$

$$\neg p(\mathbf{a}) \vee q(\mathbf{a})$$

$$\neg p(\mathbf{a}) \qquad\qquad q(\mathbf{a})$$

$$< \texttt{CLASH} > \qquad\qquad < \texttt{CLASH} >$$

Example

$\phi \wedge \psi$	$\phi \vee \psi$	$\forall x.\ \phi$	$\exists x.\ \phi$
ϕ	$\phi \mid \psi$	$\phi\{X/t\}$	$\phi\{X/a\}$
ψ			

$$\boxed{\exists y.\ (p(y) \wedge \neg q(y)) \wedge \forall z.\, (\neg p(z) \vee q(z))}$$

$$\exists y.\ (p(y) \wedge \neg q(y))$$

$$\forall z.\ (\neg p(z) \vee q(z))$$

$$p(\mathbf{a}) \wedge \neg q(\mathbf{a})$$

$$p(\mathbf{a})$$

$$\neg q(\mathbf{a})$$

$$\neg p(\mathbf{a}) \vee q(\mathbf{a})$$

$$\neg p(\mathbf{a}) \qquad\qquad q(\mathbf{a})$$

$$< \texttt{CLASH} > \qquad\qquad < \texttt{CLASH} >$$

The formula is unsatisfiable.

Other Reasoning Problems

- **Subsumption**
  - $\Gamma \models P \sqsupseteq Q$

    P and Q *predicate symbols* of the same arity
  - P subsumes Q in Γ
  - $\Gamma \models \forall \hat{x}. \, [\, Q(\hat{x}) \rightarrow P(\hat{x}) \,]$
- **Instance Checking**
  - The constant a is an instance of the unary predicate P
  - $\Gamma \models P(a)$

Example

Given a theory Γ,

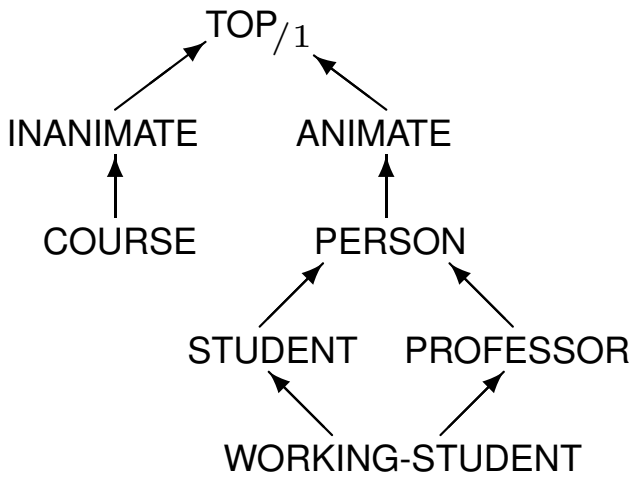
if $\texttt{PERSON} \sqsupseteq \texttt{STUDENT}$ in Γ

and if $\texttt{STUDENT}(\texttt{john})$ is valid in Γ

then $\texttt{PERSON}(\texttt{john})$ is valid in Γ

Subsumption

- *Subsumption* can be seen as a binary relation in the space of predicates with same arity: $P_{/n} \sqsupseteq Q_{/n}$.
- The subsumption relation is a *partial ordering* relation in the space of predicates of same arity.
  - **Exercise**: prove it.
  - *Hint*: a partial ordering relation is a transitive, reflexive, and antisymmetric relation.



Model Checking

Verify that a given interpretation $\mathcal{I}$ is a model for a closed formula φ:

$\mathcal{I} \models \varphi$

An interpretation is also called a *relational structure*.

Example:

$\Delta = \{a, b\},$
$P(a),$
$Q(b).$

is a model of the formula:

$\exists y.\ [\ P(y) \wedge \neg Q(y)\ ] \wedge \forall z.\ [\ P(z) \vee Q(z)\ ]$

Termination and Decidability

Given a logic $\mathcal{L}$, a reasoning problem is said to be **decidable** if there exists a computational process (e.g., an algorithm, a Turing Machine, a computer program, etc.) that solves the problem in a finite number of steps, i.e., the process always terminates.

- The problem of deciding whether a formula φ is logically implied by a theory Γ is undecidable in full FOL.
- Logical implication is decidable if we restrict to propositional calculus.
- Logical implication is decidable if we restrict to FOL using only at most *two* variable names; such language is called $\mathcal{L}_2$.

The property of (un)decidability is a general property of the problem and not of a particular algorithm solving it.

$\mathcal{L}_3$

An example of $\mathcal{L}_3$ formula:

"The only baseball player in town is married to one of Jeremy's daughters."

$$\begin{array}{l}
\exists \mathtt{x}.\ [\,\mathtt{B}(\mathtt{x}) \wedge \\
\qquad (\forall \mathtt{y}.\ [\,\mathtt{B}(\mathtt{y}) \rightarrow \mathtt{x} = \mathtt{y}\,]\,) \wedge \\
\qquad \exists \mathtt{z}.\ [\,\mathtt{M}(\mathtt{x},\mathtt{z}) \wedge \mathtt{D}(\mathtt{z},\mathtt{jeremy}) \wedge \\
\qquad\qquad (\exists \mathtt{k}.\ [\,\mathtt{D}(\mathtt{k},\mathtt{jeremy}) \wedge \mathtt{z} \neq \mathtt{k}\,]\,) \wedge \\
\qquad\qquad \forall \mathtt{v}.\ [\,\mathtt{M}(\mathtt{x},\mathtt{v}) \rightarrow \mathtt{v} = \mathtt{z}\,]\,]\,]
\end{array}$$

$$\begin{array}{l}
\exists \mathtt{x}.\ [\,\mathtt{B}(\mathtt{x}) \wedge \\
\qquad (\forall \mathtt{y}.\ [\,\mathtt{B}(\mathtt{y}) \rightarrow \mathtt{x} = \mathtt{y}\,]\,) \wedge \\
\qquad \exists \mathtt{y}.\ [\,\mathtt{M}(\mathtt{x},\mathtt{y}) \wedge \mathtt{D}(\mathtt{y},\mathtt{jeremy}) \wedge \\
\qquad\qquad (\exists \mathtt{x}.\ [\,\mathtt{D}(\mathtt{x},\mathtt{jeremy}) \wedge \mathtt{y} \neq \mathtt{x}\,]\,) \wedge \\
\qquad\qquad \forall \mathtt{v}.\ [\,\mathtt{M}(\mathtt{x},\mathtt{v}) \rightarrow \mathtt{v} = \mathtt{y}\,]\,]\,]
\end{array}$$

Expressive Power

- Some logics can be made decidable by sacrificing some *expressive power*.
- A logical language $\mathcal{L}_a$ has more expressive power than a logical language $\mathcal{L}_b$, if each formula of $\mathcal{L}_b$ denotes the "same" set of models of its correspondent formula of $\mathcal{L}_a$, and if there is a formula of $\mathcal{L}_a$ denoting a set of models which is denoted by no formula in $\mathcal{L}_b$.

Example:

Consider $\mathcal{L}_a$ as FOL, and $\mathcal{L}_b$ as FOL without negation and disjunction. Given a common domain, the $\mathcal{L}_a$ formula $\exists x.\ [\ P(x) \vee Q(x)\ ]$ has a set of models which can not be captured by any formula of $\mathcal{L}_b$.

(Exercise: check it out with $\Delta = \{a\}$

Problems and Algorithms

- A **problem** is a general question to be answered.

  A problem is described by giving:

  - a general description of all its parameters, and
  - a statement of what properties the answer, or *solution*, is required to satisfy.

  An *instance* of a problem is obtained by specifying values for all parameters.

- An **algorithm** is a step-by-step procedure fro solving problems.

  An algorithm is said to *solve* a problem Π if

  - it can be applied to any instance I of Π, and
  - it is guaranteed always to produce a solution for that instance I.

Computational Complexity

- The goal of complexity theory is to classify problems as to their intrinsic computational difficulty into general *complexity classes*.
- Given a problem, how much computing power and/or resources (e.g., *time*, *space*) do we need in order to solve it **in the worst case**?
- The complexity class to which a problem belongs is a general property of the problem and not of a particular algorithm solving it.
- Distinguish among:
  - worst case $\Longleftarrow$
  - average case
  - hard and easy cases

Complexity classes

- P
- NP — coNP
- PH
- PSPACE
- EXPTIME
- NEXPTIME
- DECIDABLE

Complexity of problems

Complexity and expressive power:

- Satisfiability of formulas in propositional calculus is NP-complete.
- Unsatisfiability of formulas in propositional calculus is coNP-complete.
- Satisfiability of QBF formulas in FOL where there is a finite nesting of quantifiers is PSPACE-complete:
  $Q_1 x_1 .\ Q_2 x_2 .\ \ldots\ [\ \varphi(x_1, x_2, \ldots)\ ]$,
  where Q_i is either $\forall$ or $\exists$.
- Satisfiability of formulas in the propositional *dynamic* normal modal logic (PDL) is EXPTIME-complete. *(Note: PDL has non first order expressible features, but it doesn't include full FOL)*
- Satisfiability of $\mathcal{L}_2$ formulas is NEXPTIME-complete.

Complexity of problems

Computational complexity depends on the reasoning problem:

- Model checking of FOL formulas is polynomial.

The classical inference systems

Usually proof theory studies the properties of inference systems for logical implication based on some sort of *sequent calculus*, or of *natural deduction*. While this seems appropriate from a mere theoretical point of view, it turns out that such systems are hardly implementable, due to their non-deterministic and non-constructive foundations.

Basically, these proof systems formalise the notion of *derivability* "$\Gamma \vdash \varphi$" with a set of inference rules. In this context:

- Soundness: if $\Gamma \vdash \varphi$ then $\Gamma \models \varphi$
- Completeness: if $\Gamma \models \varphi$ then $\Gamma \vdash \varphi$

Incompleteness

- Sequent calculus and natural deduction form sound and complete procedures for computing logical implication.
- However, for FOL it is not guaranteed their termination, since the problem is undecidable.
- It is easy to have incomplete but terminating procedures, by simply dropping some of the inference rules from the complete set.
- This is a general methodology for characterising the incompleteness of algorithms: *find a complete set of inference rules, and characterise the incomplete procedure with a subset of them.*

Completeness

- Given an incomplete reasoning procedure for a reasoning problem, sometimes people prefer to modify the definition of the problem in order to obtain a complete procedure.
- This can be accomplished by slightly changing ("weakening") the semantics of the logical language (e.g., a variant of *relevance* 4-valued FOL where modus ponens is no more valid is decidable, and it has a complete reasoning procedure).

Algorithm Complexity

- An **sound and complete** algorithm solving a reasoning problem may be or may be not optimal with respect the computational complexity of the problem.
- For example, a polynomial problem may be implemented with a non-polynomial algorithm, or a PSPACE problem may be implemented with an algorithm requiring exponential space.
- Optimal implementation of sound and complete algorithms may be impossible if there is no constructive proof for the computational complexity.

Sub-optimal algorithms

- Recall that computational complexity considers only the **worst cases**.
- Sub-optimal sound and complete algorithms can be faster for simple, average, and real instances of the problem, but less efficient for worst cases, than optimal algorithms.
- Sub-optimal sound and complete algorithms can be compliant to software engineering requirements, i.e., they can be modular and expandable.
- Sub-optimal sound and complete algorithms may be optimal when considering sub-languages belonging to a lower complexity class.

Summary of Part I of the course

- Main lesson: Logic can be a useful tool. It allows us to represent information about a domain in a very straight-forward way then deduce additional facts using one general domain-independent ”algorithm”: deduction. Consequently, logic lends itself to large-scale, distributed-design problems.
- Each logic is made up of a syntax, a semantics, a definition of the reasoning problems and the computational properties, and inference procedures for the reasoning problems (possibly sound and complete). The syntax describes how to write correct sentences in the language, the semantics tells us what sentences mean in the ”real world.” The inference procedure derives results logically implied by a set of premises.

- Logics differ in terms of their representation power and computational complexity of inference. The more restricted the representational power, the faster the inference in general.
- Propositional logic: we can only talk about facts and whether or not they are true. In the worst case, we can use the brute force truth-table method to do inference. Proof methods such as tableaux are generally more efficient, easier to implement, and easier to understand.
- First-order logic: we can now talk about objects and relations between them, and we can quantify over objects. Good for representing most interesting domains, but inference is not only expensive, but may not terminate.