Advanced Knowledge Based Systems
CS3411

Using First Order Logic

Enrico Franconi

http://www.cs.man.ac.uk/∼franconi/teaching/1999/3411/

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*).
    * $\boxed{\phi \models \psi \text{ iff } \phi \land \neg\psi \text{ is not satisfiable}}$.
- A *sound and complete* procedure deciding satisfiability is all we need, and 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:

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

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

$$\dfrac{\phi \vee \psi}{\phi \;\mid\; \psi}$$

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:

$$\frac{\forall x.\ \phi}{\begin{array}{c}\phi\{X/t\}\\ \forall x.\ \phi\end{array}}$$

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

$$\frac{\phi \wedge \psi}{\begin{array}{c}\phi\\ \psi\end{array}} \qquad \frac{\phi \vee \psi}{\begin{array}{c|c}\phi & \psi\end{array}} \qquad \frac{\forall x.\ \phi}{\phi\{X/t\}} \qquad \frac{\exists x.\ \phi}{\phi\{X/a\}}$$

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

Example

$$\frac{\phi \land \psi}{\begin{array}{c}\phi\\ \psi\end{array}} \qquad \frac{\phi \lor \psi}{\phi \mid \psi} \qquad \frac{\forall x.\ \phi}{\phi\{X/t\}} \qquad \frac{\exists x.\ \phi}{\phi\{X/a\}}$$

$$\boxed{\exists y.\ (p(y) \land \neg q(y)) \land \forall z. (p(z) \lor q(z))}$$

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

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

Example

$$\frac{\phi \wedge \psi}{\begin{array}{c}\phi\\ \psi\end{array}} \qquad \frac{\phi \vee \psi}{\phi \;\big|\; \psi} \qquad \frac{\forall x.\ \phi}{\phi\{X/t\}} \qquad \frac{\exists x.\ \phi}{\phi\{X/a\}}$$

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

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

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

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

Example

$$\frac{\phi \land \psi}{\begin{array}{c}\phi\\ \psi\end{array}} \qquad \frac{\phi \lor \psi}{\phi \;\Big|\; \psi} \qquad \frac{\forall x.\ \phi}{\phi\{X/t\}} \qquad \frac{\exists x.\ \phi}{\phi\{X/a\}}$$

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

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

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

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

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

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

Example

$$\frac{\phi \wedge \psi}{\begin{array}{c}\phi \\ \psi\end{array}} \qquad \frac{\phi \vee \psi}{\phi \;\Big|\; \psi} \qquad \frac{\forall x.\ \phi}{\phi\{X/t\}} \qquad \frac{\exists x.\ \phi}{\phi\{X/a\}}$$

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

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

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

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

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

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

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

Example

$$\frac{\phi \wedge \psi}{\begin{array}{c}\phi \\ \psi\end{array}} \qquad \frac{\phi \vee \psi}{\phi \mid \psi} \qquad \frac{\forall x.\ \phi}{\phi\{X/t\}} \qquad \frac{\exists x.\ \phi}{\phi\{X/a\}}$$

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

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

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

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

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

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

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

$p(\mathbf{a})$

$<$ `COMPLETED` $>$

Example

$$\frac{\phi \wedge \psi}{\begin{array}{c}\phi \\ \psi\end{array}} \qquad \frac{\phi \vee \psi}{\phi \mid \psi} \qquad \frac{\forall x.\ \phi}{\phi\{X/t\}} \qquad \frac{\exists x.\ \phi}{\phi\{X/a\}}$$

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

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

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

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

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

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

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

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

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

Example

$$\frac{\phi \wedge \psi}{\begin{array}{c}\phi \\ \psi\end{array}} \qquad \frac{\phi \vee \psi}{\begin{array}{c|c}\phi & \psi\end{array}} \qquad \frac{\forall x.\ \phi}{\phi\{X/t\}} \qquad \frac{\exists x.\ \phi}{\phi\{X/a\}}$$

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

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

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

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

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

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

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

$$\begin{array}{ccc} p(\mathbf{a}) & \qquad\qquad & q(\mathbf{a}) \\ <\texttt{COMPLETED}> & & <\texttt{CLASH}> \end{array}$$

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

Example

$$\frac{\phi \wedge \psi}{\begin{array}{c}\phi \\ \psi\end{array}} \qquad \frac{\phi \vee \psi}{\phi \;\Big|\; \psi} \qquad \frac{\forall x.\ \phi}{\phi\{X/t\}} \qquad \frac{\exists x.\ \phi}{\phi\{X/a\}}$$

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

Example

$$\frac{\phi \wedge \psi}{\begin{array}{c}\phi\\ \psi\end{array}} \qquad \frac{\phi \vee \psi}{\phi \mid \psi} \qquad \frac{\forall x.\ \phi}{\phi\{X/t\}} \qquad \frac{\exists x.\ \phi}{\phi\{X/a\}}$$

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

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

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

Example

$$\frac{\phi \wedge \psi}{\begin{array}{c}\phi\\ \psi\end{array}} \qquad \frac{\phi \vee \psi}{\phi \;\Big|\; \psi} \qquad \frac{\forall x.\ \phi}{\phi\{X/t\}} \qquad \frac{\exists x.\ \phi}{\phi\{X/a\}}$$

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

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

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

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

Example

$$\frac{\phi \wedge \psi}{\begin{array}{c}\phi \\ \psi\end{array}} \qquad \frac{\phi \vee \psi}{\phi \mid \psi} \qquad \frac{\forall x.\ \phi}{\phi\{X/t\}} \qquad \frac{\exists x.\ \phi}{\phi\{X/a\}}$$

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

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

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

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

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

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

Example

$$\frac{\phi \wedge \psi}{\begin{array}{c}\phi\\ \psi\end{array}} \qquad \frac{\phi \vee \psi}{\phi \mid \psi} \qquad \frac{\forall x.\ \phi}{\phi\{X/t\}} \qquad \frac{\exists x.\ \phi}{\phi\{X/a\}}$$

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

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

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

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

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

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

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

Example

$$\frac{\phi \wedge \psi}{\begin{array}{c}\phi \\ \psi\end{array}} \qquad \frac{\phi \vee \psi}{\phi \mid \psi} \qquad \frac{\forall x.\ \phi}{\phi\{X/t\}} \qquad \frac{\exists x.\ \phi}{\phi\{X/a\}}$$

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

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

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

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

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

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

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

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

$< \texttt{CLASH} >$

Example

$$\frac{\phi \wedge \psi}{\begin{array}{c}\phi \\ \psi\end{array}} \qquad \frac{\phi \vee \psi}{\phi \mid \psi} \qquad \frac{\forall x.\ \phi}{\phi\{X/t\}} \qquad \frac{\exists x.\ \phi}{\phi\{X/a\}}$$

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

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

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

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

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

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

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

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

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

Example

$$\begin{array}{c}\phi \wedge \psi \\ \hline \phi \\ \psi\end{array} \qquad \begin{array}{c}\phi \vee \psi \\ \hline \phi \mid \psi\end{array} \qquad \begin{array}{c}\forall x.\ \phi \\ \hline \phi\{X/t\}\end{array} \qquad \begin{array}{c}\exists x.\ \phi \\ \hline \phi\{X/a\}\end{array}$$

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

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

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

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

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

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

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

$$\begin{array}{cc}\neg p(\mathbf{a}) & q(\mathbf{a}) \\ < \texttt{CLASH} > & < \texttt{CLASH} >\end{array}$$

The formula is unsatisfiable.

Exercise

Jones, Smith, and Clark are a programmer, a knowledge engineer, and a manager (non necessarily in this order). Jones should give 50 pounds to the programmer. The wife of the manager does not want that his husband borrows money from somebody. Smith is not married. How can you tell the work of Jones, Smith, and Clark?

Exercise

1. All poems by Yeats are older than some books by Steinbeck.
2. It is not true that a writer is famous only if their age is greater than the age of Steinbeck plus the age of any Romantic.
3. Some books by Pushkin are better than all books by better known writers who are less talented.
4. If a book was written by a novelist other than Hemingway, it is not better than every book by Hemingway or else Mike would like it.
5. Poets who are less talented than Yeats either like Yeats or think all famous writers are crazy.

Exercise

Translate the following sentences in first order predicate calculus into plain English. Interpret the relation and object constants according to their English meaning.

1. $\forall$`x.`$\exists$`y.(Season(x)` $\Rightarrow$ `Flower(y)` $\wedge$ `Blooms(y, x))`
2. $\neg$ `(` $\neg$ `(` $\forall$`x.`$\exists$`y.(Problem(x)` $\wedge$ `Solution(y)` $\wedge$ `Solves(y, x)))` $\Rightarrow$
   $\forall$`x.(Problem(x)` $\wedge$ `Having(You, x)` $\Rightarrow$ `Desperate(You)))`
3. $\forall$`x.`$\exists$`y.(Problem(x)` $\Rightarrow$ `Solution(y)` $\wedge$ `Solves(y, x))`
4. $\exists$`y.`$\forall$`x.(Problem(x)` $\Rightarrow$ `Solution(y)` $\wedge$ `Solves(y, x))`
5. $\forall$`x.(SmartPerson(x)` $\wedge$ `Taking(x, CS157)` $\Rightarrow$
   `(Loves(x, Logic)` $\vee$
   `(` $\forall$`y.`$\exists$`z.(CSclass(z)`$\wedge$`Hates(y, SubjectOf(z))` $\Rightarrow$ `ThinkCrazy(x, y)))))`

Other Reasoning Problems

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

Example

Given a theory Γ,

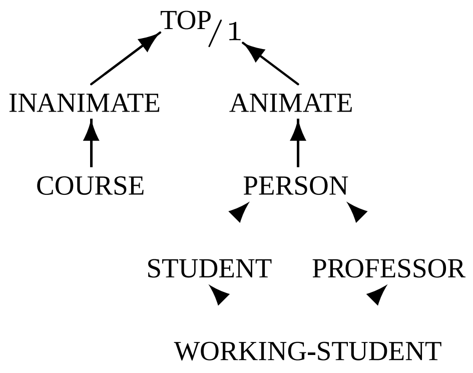
if $\mathsf{PERSON} \sqsupseteq \mathsf{STUDENT}$

and if $\mathtt{STUDENT}(\mathtt{john})$ is valid

then $\mathtt{PERSON}(\mathtt{john})$ is valid

Subsumption

- *Subsumption* can be seen as a binary relation in the space of predicates with same arity: $\varphi_{/n} \sqsupseteq \psi_{/n}$.
- 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.

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

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

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 formalize 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 characterizing the incompleteness of algorithms: *find a complete set of inference rules, and characterize 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.

Methodology:
a guide to axiomatizing domains in FOL.
1.

- Define a model or microworld, as specifically and precisely as the nature of the domain allows. (Obviously, a much more concrete model can be given in the domain of spatial relations than in the domain of human emotions.) You can then check for the soundness and consistency of your axioms by establishing that they are all true in the model.
- Do not cheat on your model. That is, if you find that you want to include in your axioms some new feature that is not in your model, do not just write down a new axiom. Rather, you should rethink the model to include the new feature, and check that all of your axioms still make sense relative to the revised model.

2.

Make two tables enumerating all the sorts and all the non-logical primitives you use in your logic, with their definitions in terms of the model. If it is not possible to give necessary and sufficient conditions for the primitive, write down in English as precise a definition as you can manage, and try at least to bound the concept by giving some necessary conditions and some sufficient conditions.

3.

Some of the tenets of good programming style hold for axiomatizations as well:

1. Non-logical primitives, like programming language identifiers, should be long enough that the reader can immediately know what they mean, but not so long that your axioms must be split up over many lines.

2. Modularize. If you find the same subformula coming up again and again, define a new primitive to capture it.

3. Use indentation and brackets to make the structure of the axiom apparent.

4.

Cast a cold eye on every material implication. Keep in mind that ”$p \longrightarrow q$” means exactly ”$\neg p \vee q$”, and nothing more.

5.

If you have written ”$\forall x.p(x) \wedge q(x)$” it is likely that you have made a mistake. Rewrite it as the two axioms ”$\forall x.p(x)$” and ”$\forall x.q(x)$” and see if it still makes sense.

6.

- If you have written ”$\exists x.(p(x) \rightarrow q(x))$” then it is 100 to 1 that you have made a mistake. Rewrite it in the logically equivalent form ”$(\exists x.\neg p(x)) \vee (\exists x.q(x))$” and you will probably realize where you have gone wrong.
- The only exception to this I have ever seen is where the variable x does not actually appear in the left hand side of the implication; for instance, in a form like ”$\forall y.\exists x.p(y) \rightarrow q(x,y)$”. Rewrite this for clarity as ”$\forall y.p(y) \rightarrow \exists x.q(x,y)$.”

7.

- If you have the implication "$p \rightarrow q$", see if you can turn it into a biconditional "$p \leftrightarrow q$." You may have to strengthen the right hand side to "$p \leftrightarrow (q \wedge r)$". This can be an easy way to get a more powerful theory.
- This does, however, require some care in choosing a grouping. Forms that are logically equivalent as implications may cease to be so when the implication is turned into a biconditional, and you have to choose the correct form. For instance, the two formulas

$$\forall x.(\exists y.x = y + y) \rightarrow \mathsf{even}(x)$$

and

$$\forall x, y.x = y + y \rightarrow \mathsf{even}(x)$$

are equivalent. However, though the first can be turned into the correct biconditional

$$\forall x.(\exists y.x = y + y) \leftrightarrow \mathsf{even}(x)$$

the second cannot.

7 bis.

- Another example: The assertion *”If a triangle is equilateral then it is equiangular”* can be represented

  $$\forall \mathtt{X}.(\mathtt{triangle}(\mathtt{x}) \wedge \mathtt{equilateral}(\mathtt{x})) \longrightarrow \mathtt{equiangular}(\mathtt{x})$$

  To turn this into a biconditional, however, requires separating out the condition ”$\mathtt{triangle}(\mathtt{x})$” as a condition on the entire biconditional.

  $$\forall \mathtt{x}.\mathtt{triangle}(\mathtt{x}) \longrightarrow (\mathtt{equilateral}(\mathtt{x})) \leftrightarrow \mathtt{equiangular}(\mathtt{x}))$$

8.

Equality and function symbols are powerful representational tools; by all means, use them. However, there are a number of common errors to be avoided.

- The formula ”$x = y$” means that x and y are identical things; it is not a translation of ”`x is y`”. Beware: ”`Block1 = Red; Block2 = Blue; Block3 = Red.`” This would imply that `Block1=Block3`.
- Keep in mind that functions in FOL are necessarily total. Therefore formulas like ”$\exists y. y = \mathtt{f}(x)$” are always true and therefore vacuous.

8 bis.

- If you write down an FOL sentence containing the form "$x = t$" for some variable x and term t, then you should think whether it is possible to simplify the sentence. In particular, the sentence "$\forall x . x = t \rightarrow p(x)$" is equivalent to "$\forall x . p(t)$". If t does not contain x, then this is equivalent to $p(t)$. Similarly, "$\exists x . x = t \wedge p(x)$" is equivalent to "$\exists x . p(t)$". If t does not contain x, this is equivalent to $p(t)$.

- For example, if you are translating the sentence, *"Your father is older than you are,"* (in the sense of "you" meaning "everyone"), it is natural to gloss this as *"If X is the father of Y then X is older than Y"* and then to write it as "$\forall \mathtt{y}, \mathtt{y} . \mathtt{x} = \mathtt{father}(\mathtt{y}) \rightarrow \mathtt{older}(\mathtt{y}, \mathtt{y})$". This can be expressed more elegantly as "$\forall \mathtt{y} . \mathtt{older}(\mathtt{father}(\mathtt{y}), \mathtt{y})$."

- On the other hand, the original form with the explicit implication may lend itself to conversion to a biconditional, as discussed in (7) above. For instance, the fact, *"2 is an even prime,"* if expressed in the form "$\forall (\mathtt{x}) \mathtt{x} = 2 \rightarrow (\mathtt{even}(\mathtt{x}) \wedge \mathtt{prime}(\mathtt{x}))$", can be easily converted to the biconditional *"2 is the only even prime,"* "$\forall \mathtt{x} . \mathtt{x} = 2 \leftrightarrow (\mathtt{even}(\mathtt{x}) \wedge \mathtt{prime}(\mathtt{x}))$."

9.

Keep in mind that FOL does not have negation as failure. This is not Prolog. This is particularly important in recursive definitions. If you try to define ”above” as the transitive closure of ”on”:

$$\forall \mathtt{x}, \mathtt{y}.\mathtt{on}(\mathtt{z}, \mathtt{y}) \longrightarrow \mathtt{above}(\mathtt{x}, \mathtt{x})$$

$$\forall \mathtt{x}, \mathtt{y}, \mathtt{z}.(\mathtt{on}(\mathtt{x}, \mathtt{y}) \wedge \mathtt{above}(\mathtt{y}, \mathtt{z})) \longrightarrow \mathtt{above}(\mathtt{x}, \mathtt{z})$$

you will be able to prove that some things are above others, but you will never be able to prove that something is not above something else, because this definition is consistent with everything in the world being above everything.

10.

Don’t do programming. FOL theories describe what’s true, not how to compute things. If you find yourself yearning for loops, reassignable variables, data structures, or gotos, then either you’re on the wrong track or you should stop working in FOL and switch to a programming language.

10 bis.

- Check to see whether there exists a simpler or more general model for your axiomatization than the microworld you have in mind. For example, if you think a microworld involves gravity, then imagine a gravity-less version, and ask whether your axioms are still true in that. Another example: some theories of spatial regions in the AI literature are in fact true of arbitrary sets.
- Finding such an alternative model can have its pros and cons. Clearly, it means that you have not captured all aspects of your microworld. On the other hand, in a given application, you may not need these aspects. Identifying the axiomatization as a description of a simpler model may lead you to effective algorithms, decidability results, and so on. But, one way or another, it is definitely a fact worth knowing about an axiomatic system.

Summary of Part I of the course

1. 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. When representing, we don’t need to worry about how the information will be used, making it very scalable and compositional. Consequently, logic lends itself to large-scale, distributed-design problems.

2. Each logic is made up of a syntax, a semantics, a definition of the reasoning problems and thei computational properties, and inference procedures for the reasoning problems (possinly 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.

3. 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.

4. 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 which takes $\mathcal{O}(2^k)$ time if we have k propositional constants. Proof methods such as tableaux are generally more efficient, easier to implement, and easier to understand (both the method and the proof).

5. First-order logic: e can now talk about objects and relations between them, and we can quantify over objects. Sufficient for representing most interesting domains, but inference is not only expensive, but may not terminate (it is semi-decidable).