

Logics for Computation

Carlos Areces and Patrick Blackburn
{carlos.areces,patrick.blackburn}@loria.fr

INRIA Nancy Grand Est
Nancy, France

ESSLLI 2008 - Hamburg - Germany

What is Logic, and why should I care?

- ▶ Probably all of you have heard about 'Logic' before.
- ▶ But what is logic for you? Perhaps it's the science that studies strange symbols like

$$(p \wedge q) \rightarrow (p \vee q)$$

$$\forall x(\text{Human}(x) \rightarrow \text{Mortal}(x))$$

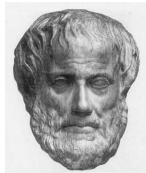
that are (allegedly) important in natural language semantics, computer science, computational linguistics, for (somewhat mysterious) reasons.

- ▶ And perhaps you've encountered what logicians called 'theorems', expressions like:

$$p \vee \neg p \quad \text{or} \quad p \rightarrow p.$$

Back to Aristotle

Or perhaps you have met logic in a philosophical setting? You're aware of the work of Aristotle (384 BC - 322 BC), and in particular his discussion of syllogisms.



All vampires are demons.
Angel is a vampire.
Therefore Angel is a demon.

Tomorrow it will rain or it won't...

Either way, logic may not have struck you as particularly exciting or relevant to your work.

- ▶ Sentences like "John loves Mary, or not" or "It will rain or it won't, tomorrow" sound a bit silly. They don't seem to be very informative.
- ▶ Nor do simple syllogisms seem to have much to do with reasoning in natural language (though, to be fair, they do seem similar to the types of arguments found when reasoning about simple ontologies or when working with WORDNET).
- ▶ And they certainly seem far removed from the type of arguments found in computer science and mathematics. And the mathematicians notion of 'theorem' seems very different (and much richer) than the logicians notion.

Don't despair!

- ▶ This foundational course was designed for people with little or no knowledge of logic, or for those unconvinced that it really can help them in their work.
- ▶ It explains a number of topics and technical skills in logic (for example, tableaux, the Davis Putnam Method, the Compactness Theorem).
- ▶ But more importantly, it offers a **framework for thinking about logic**.
- ▶ That is, we try to fill in the big picture that is so often missing.

Logic are languages

- ▶ We want you to **think of logics as languages**.
- ▶ In particular we want you to **think of logics as ways of talking about relational structures or models**.
- ▶ That is there are two key components in the way we will approach logic
 - ▶ The **logic**: fairly simple, precisely defined, formal languages. (This is where the funny symbols like \wedge and \exists live).
 - ▶ The **model** or **relational structure**: A simple 'world' (or 'database') that the logic talks about.

Semantic perspective

That is, our perspective on logic is fundamentally **semantic**. It is due to Alfred Tarski (1902–1983).



The semantic perspective is also known as the **model-theoretic** perspective, or even the **Tarskian** perspective.

Logic or logics?

The semantic perspective gives us a good way to think about the following question: How many logics are there?
There are (at least) two ways to think about Logic.

- ▶ **Option 1, The Monotheistic Approach**: Choosing one of all possible logical languages and saying "This is THE Logic", or
- ▶ **Option 2, The Polytheistic Approach**: As a discipline that investigate different logical languages.

Monotheism in the 20th century

Logical monotheism was a powerful force for much of the twentieth century.



Perhaps the most influential monotheist was Willard van Orman Quine, who championed first-order classical logic as the one-true-logic with vigor.

Though (disturbingly for the monotheists) there were always those who worshiped at other temples (such as the intuitionistic logicians and Arthur Prior).

Polytheism in the 21st century

- ▶ But polytheism gradually became the dominant thread as time went by.
- ▶ Why? Because logic spread everywhere. Computer scientists used it. Early artificial intelligence relied on it. It cropped up in economic and cognitive science. And it became corner stone of natural language semantics.

Fighting for polytheism



Nowadays logical polytheism is pretty philosophically respectable too. Graham Priest is one of it's most interesting and original proponents.

Polytheism and semantics

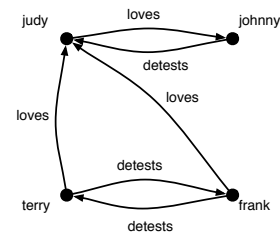
- ▶ However the most important point for this course, is that polytheism as regards logic is very natural from a semantic perspective.
- ▶ Once we have fixed the model or relational structures we wish to work with (that is, once we have fixed out 'world') it becomes natural to play with different ways of talking about it.
- ▶ Indeed today we are going to talk about relational structures without bothering too much about the logics at all.

Relational Structures (informal)

A relational structure (or model) consists of the following

- ▶ A non-empty set (often called D , for **domain**) of the model; think of these as the objects of interest.
- ▶ A collection of **relations** R on the objects in D ; think of these as the relations of interest (we shall only work with **binary relations** (that is, two place relations like "loves", " $<$ ", or "to-the-right-of" in this course) to keep the notation simple.
- ▶ A collection of **properties** on the objects in D ; think of these as the properties of interests (perhaps "is red", "is activated", or "is an even number").
- ▶ A collection of **designated individuals**, that is, elements of D that we find really special (maybe "Buffy", "0", or "1")

Our first relational structure



Reminder

A small mathematical reminder:

- ▶ **Properties are thought of as subsets.** That is, given any set D , a property on D is simply a subset P of D ; that is $P \subseteq D$.
- ▶ **Binary relations are thought of as sets of ordered pairs.** That is, given any set D , a binary relation R is a subset of $D \times D$; that is, $R \subseteq D \times D$.

Relational Structures (more formally)

A **relational structure** (or **model**) is a tuple of the form:

$$\langle D, \{R_m\}, \{P_n\}, \{C_l\} \rangle$$

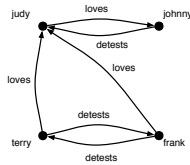
Sometimes we work with simpler forms. For example the following

$$\langle D, R, \emptyset, \emptyset \rangle$$

we would usually write as:

$$\langle D, R \rangle$$

Another look at our first relational structure



$$\langle D, \{L, D\}, \emptyset, C \rangle \text{ or } \langle D, \{L, D\}, \emptyset, C \rangle$$

$$D = \{\text{judy}, \text{johnny}, \text{terry}, \text{frank}\}$$

$$L = \{(\text{judy}, \text{johnny}), (\text{terry}, \text{judy}), (\text{frank}, \text{judy}), \}$$

$$D = \{(\text{johnny}, \text{judy}), (\text{terry}, \text{frank}), (\text{frank}, \text{terry}), \}$$

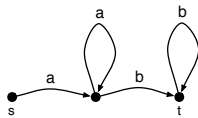
$$C = \{\text{judy}, \text{johnny}, \text{terry}, \text{frank}\}$$

What can be thought of as a relational structure. . . ?

That's the wrong question — the real question is, what **can't** be thought of as a relational structure?

In fact, it is very hard to think of **anything** (barring some rather extreme mathematical examples) that can't be viewed as a relational structure.

Example



This shows a finite state automaton for the formal language $a^n b^m$ ($n, m > 0$), that is, for the set of all strings consisting of a non-empty block of *a*s followed by a non-empty block of *b*s.

A general and important modelling tool

- ▶ All common mathematical structures can be thought of as relational structures. For a start, functions are simply special kinds of relations.
- ▶ Moreover, groups, rings, field, vector spaces, ... can all be viewed as relational structures.
- ▶ Thus Tarski's idea had substantial mathematical impact, and this was decisive in establishing model-theory as an academic discipline.
- ▶ Now it is the turn of other disciplines — and in particular, those disciplines related to the LLI theme of this summer school — to draw on the Tarskian insights.

The three big themes

In this course we use the model-theoretic perspective to provide a window on the following three issues:

- ▶ **Inference**: roughly speaking, what methods are there for gaining new information by working with this logic?
- ▶ **Expressivity**: roughly speaking, what can I describe (and what can't I describe) using this logic?
- ▶ **Computation**: too many to list — can computers help with this logic? If so how, and how much?
- ▶ **Interestingness**: Hard to quantify — but perhaps equally fundamental.

Inference tasks

The semantic perspective give us a good way to think about inference tasks. We shall concentrate on the following three:

- ▶ Given a certain logical description ϕ , and a model \mathcal{M} , does the formula correctly describe (some aspect of) the model? More simply: is the formula true (or satisfied) in the model? This task is called the **model checking** task.
- ▶ Given a certain logical description ϕ , does there exist a model where that formula is true? This task is called the **satisfiability checking** task (or the **model building** task).
- ▶ Given a logical description φ , is it true (or satisfied) in **all** models? This task is called the **validity checking** task.

The Model Checking task

- ▶ As we shall learn, this is the simplest of the three tasks.
- ▶ However, it has also proved to be one of the most useful.
- ▶ A classic application is hardware verification. The model \mathcal{M} is a mathematical picture of (say) a chip. The logical description ϕ describes some desirable feature of the chip. If the \mathcal{M} makes ϕ true, then the chip will have that property.
- ▶ Incidentally, this example already suggests the need for "designing logics for their application". After all, there is not reason to think that an off-the-shelf logic will provide exactly what is needed to talk usefully about chips and their properties.

Satisfiability checking (or model building)

- ▶ A nice way to think of this problem is in terms of constraints. we have some description. Is there anything that matches this description? That is, does a model making this description actually exist, and can we build it?
- ▶ Very useful. The description might be almost anything: for example, a description of a parse tree (if you're doing computational linguistics).

Validity checking

- ▶ A great deal of attention has been devoted to this task — essentially, when people talk about “writing a theorem prover”, they are talking about creating a computational tool for solving this task.
- ▶ The real question is: why? After all, we've already mentioned that $p \vee \neg p$ and $p \rightarrow p$ are not going to set too many pulses racing...

Logic is a tool for working with theories

- ▶ Let's turn to mathematics. The intuitive idea is that we write down a set Σ of all our **axioms**. These are the properties that we assume are fundamental and indisputable; what we take for granted. Σ is our theory.
- ▶ For example **Peano axioms** are a theory for the natural numbers.
- ▶ Checking if the Goldbach theory is true in the natural numbers boils down to verifying that

$$(\bigwedge \text{PEANO}) \rightarrow \text{GOLDBACH}$$

is a 'trivial' formula, that is, a validity.

Axiomatics is an ancient idea



The idea goes back to Euclid's celebrated book “The Elements”. This is rightly considered one of the foundational blocks of mathematics. It is certainly that, but it is also one of the foundations of modern logic.

Expressivity

- ▶ The theme of expressivity is fundamental to this course — and to a model-theoretically inclined logician, the theme is absolutely fundamental — though this early in the course is difficult to say very much about it.
- ▶ But the fundamental point is this. Once we have said which relational structures we are interested in, there are **many** logics suitable for talking about them. **Each offers a different (often a fascinatingly different perspective) on the same “world”**.
- ▶ Linguists may like to recall the Sapir-Whorf hypothesis: loosely speaking, the limits of our language are the limits of the world. This analogy should not be taken to literally, but it may be suggestive.

Computation

However, we can already say quite a bit about computation and how it enters the course. In fact it does so at a number of levels.

- ▶ First, ideas from theoretical computer science (such as computational complexity) are fundamental tools for analyzing logics.
- ▶ Second, more and more computer science is setting the agenda in logic.
- ▶ Third, at a practical level we simply need computers when working with logic.

Let's consider these points in turn...

How easy is it? Is it even possible?

- ▶ They say that there are some things that cannot be bought for all the money in the world. (True Love?).
- ▶ There are problems that **cannot** be algorithmically solved even with unlimited computing resources.
- ▶ **The Halting Problem**: Given a program P , decide whether P ends or not.
- ▶ Some logics are **algorithmically unsolvable** in this sense (or to be more precise, the inference problems they give rise to are algorithmically unsolvable).
- ▶ Even when an inference problem can be **algorithmically** solvable, the question arises: how hard is it?

Computational Logics: Logic in Action!

- ▶ Logic was born as part of philosophy, and achieved greatness as a branch of mathematics.
 - ▶ Originally meant to model human reasoning processes
 - ▶ and to help making **correct** correct inferences.
 - ▶ Mathematicians then turned it into a new tool for mathematics.
- ▶ With the advent of computer science, things changed
 - ▶ Logic played a **fundamental** part in the development of computers (logic circuits)
 - ▶ but nowadays **computer science fuels logic**.
- ▶ In this course a computational view on logical systems will never be far away.

Why do we Need Computers?

- ▶ Why do we need computers?
 - ▶ well, after all, if we are lazy and don't want to do the work, it would be nice if somebody else could do it for us!
 - ▶ even if we could overcome our laziness, we **wouldn't be able** to do the task ourselves.
- ▶ Some of the inference tasks we want to tackle are simply **too difficult** to perform without the help of computers
 - ▶ sometimes billions of possibilities need to be checked to verify that a system satisfies a certain property we want to enforce
 - ▶ and even using computers we need to be **clever**, or all the time till the end of the universe won't be enough. that is, computational logic is not (just) about clever engineering.

Where to from here?

- ▶ There are nine more lectures — we're viewing the first and second parts as separate lectures (so we want to keep the break in the middle short).
- ▶ Mixture of technical and conceptual — with a little bit of history mixed in.
- ▶ But above all, we want to tell a story, and we want you to listen (hence we're only putting the slides online after the course).
- ▶ And the story, of course, is the story of logic (or at least, our version of that story).

What we Covered Today

- ▶ Logic as language: the model theoretic (or semantic, or Tarskian) perspective
- ▶ Relational structures: the heart of the semantic approach.
- ▶ The themes of inference, expressivity, and computation.
- ▶ Three inference tasks: the model checking task, the satisfiability checking (or model building task), and the validity checking task.
- ▶ Briefly mentioned expressivity and raised a number of issues concerning computation.

Relevant Bibliography

There are many good introductions to logic out there. Two interesting ones, written from radically different perspectives are:

- ▶ *Philosophy of Logic* by Willard Van Orman Quine, Harvard University Press; New edition, 1980). Still in stock at amazon.
- ▶ *An Introduction to Non-Classical Logic*, by Graham Priest, Cambridge University Press; 2nd edition, 2008.

The first is a monotheistic bible. The second raises polytheism to levels worthy of Terry Pratchett's novel "Small Gods".

Relevant Bibliography

- ▶ *An introduction to good old fashioned model theory*, by Harold Simmons, <http://www.cs.man.ac.uk/~hsimmons/BOOKS/books.html>
- ▶ *The Logic of Time: A Model-Theoretic Investigation into the Varieties of Temporal Ontology and Temporal Discourse*, by Johan van Benthem, Synthese Library, 2nd revised edition, 1991.
- ▶ The life of the "the greatest sane logician" and inventor of Model Theory. *Alfred Tarski: Life and Logic*, by Anita Burdman Feferman and Solomon Feferman, Cambridge University Press, 1 paperback edition, 2008

Relevant Bibliography

Finally, earlier I warned you not to take the Sapir-Whorf hypotheses too seriously. Unfortunately, many people have done this, with appalling results. For a brief, elegant, end very very funny critique of such "work", read the following article:

- ▶ The great Eskimo vocabulary hoax, by Geoffrey Pullum, in his "Topic . . . Comment" Column, *Natural Language and Linguistic Theory*, 7, 275–281, 1989.

Logics for Computation

Lecture #2: To Pee or not to Pee?

Carlos Areces and Patrick Blackburn
{carlos.areces,patrick.blackburn}@loria.fr

INRIA Nancy Grand Est
Nancy, France

ESSLLI 2008 - Hamburg - Germany

The Story So Far

- ▶ We introduced the concept of logic (indeed, of logics) from a broad semantic (or Tarskian, or model-theoretic) perspective.
- ▶ We introduced the key unifying notion of the course: relational structures.
- ▶ We introduced the three major themes that we are interested in:
 - ▶ Inference:
 - ▶ Model checking.
 - ▶ Satisfiability checking (a.k.a. Model building).
 - ▶ Validity checking.
 - ▶ Expressivity.
 - ▶ Computation.

Areces & Blackburn: Logics for Computation

INRIA Nancy Grand Est

What do we do Today?

- ▶ We introduce the simplest logic discussed in this course, one you are probably familiar with: **Propositional Logic (PL)**.
- ▶ We introduce/review the language, the familiar **truth functional** semantics, and solve a simple problem with PL.
- ▶ We then discuss inference in more detail: we see that model checking in PL is (very) easy. Moreover, we see that models can be elegantly built using **tableaux method**, hence we solve satisfiability checking task too.
- ▶ We then show that **any method for solving the satisfiability task also solves the validity task** — hence our tableaux method solves the validity task too.
- ▶ We conclude with brief remarks on expressivity and computability.

Areces & Blackburn: Logics for Computation

INRIA Nancy Grand Est

Propositional Logic: Syntax

The language of propositional logic is simple. We have the following basic symbols:

Propositional symbols: $p, q, r, p_1, p_2, p_3, \dots$

Logical symbols: $\top, \perp, \neg, \vee, \wedge, \rightarrow, \leftrightarrow$

Grouping symbols: $(,)$

PL is sometimes called **Boolean logic** (after the pioneering English logician George Boole) and the symbols $\top, \perp, \neg, \vee, \wedge, \rightarrow$ and \leftrightarrow are often called **Boolean connectives**.

Areces & Blackburn: Logics for Computation

INRIA Nancy Grand Est

Propositional Logic: Syntax

We then say that \top, \perp and any propositional symbol are **formulas** (or **well-formed formulas**, or **wff**). These single-symbol formulas are often called **atomic formulas**.

We then construct **complex formulas** (or **compound formulas**) in accordance with the following **recursive definition**:

- ▶ If ϕ and ψ are formulas then so are $\neg\psi$, $(\phi \vee \psi)$, $(\phi \wedge \psi)$, $(\phi \rightarrow \psi)$, and $(\phi \leftrightarrow \psi)$.
- ▶ Nothing else is a formula.

That's the official syntax — but we often simplify the bracketing. For example we would typically write $((p \wedge q) \rightarrow r)$ as $(p \wedge q) \rightarrow r$.

Areces & Blackburn: Logics for Computation

INRIA Nancy Grand Est

Propositional Logic: Semantics

- ▶ The semantics is also straightforward. A **model for this language** is simply an assignment V of true (T) or false (F) to the propositional symbols. So this is a very simple conception of model; sometimes V is called an assignment.
- ▶ Thus the models for PL are **not** relational structures. Or at least, so it seems. Actually, there is a natural way to view PL in terms of relational structures — but that can wait.



Areces & Blackburn: Logics for Computation

INRIA Nancy Grand Est

Propositional Logic: Semantics

So: V determines the truth values of the propositional formulas. We then determine whether other formulas are true or false with respect to V by using the following rules. Note: **iff** is short for **if and only if**:

\top	is always true
\perp	is always false
$\neg\phi$ is true	iff ϕ is false
$\phi \vee \psi$ is true	iff either ϕ or ψ (or both) are
$\phi \wedge \psi$ is true	iff both ϕ and ψ are
$\phi \rightarrow \psi$ is true	iff either ϕ is false or ψ is true
$\phi \leftrightarrow \psi$ is true	iff ϕ and ψ are both true, or they are both false

Areces & Blackburn: Logics for Computation

INRIA Nancy Grand Est

Truth Tables

A couple of remarks. First, you may have seen the previous truth definition in the guise of a "truth table".

p	q	$\neg p$	$p \vee q$	$p \wedge q$	$p \rightarrow q$	$p \leftrightarrow q$
T	T	F	T	T	T	T
F	T	T	T	F	T	F
T	F	F	T	F	F	F
F	F	T	F	F	T	T

Truth tables are a conceptually simple (if tedious) way of working with PL.

Areces & Blackburn: Logics for Computation

INRIA Nancy Grand Est

We don't need all these connectives

Secondly, as many of you will know, many connectives can be defined in terms of others. We don't need them all.

- ▶ \perp is $\neg\top$
- ▶ \top is $p \vee \neg p$.
- ▶ $\varphi \rightarrow \psi$ is $\neg\varphi \vee \psi$.
- ▶ $\varphi \wedge \psi$ is $\neg(\neg\varphi \vee \neg\psi)$.
- ▶ $\varphi \vee \psi$ is $\neg(\neg\varphi \wedge \neg\psi)$.

A set of logical symbols that can define all the others is called **truth functionally complete**. For example, $\{\neg, \wedge\}$, $\{\neg, \vee\}$, and $\{\rightarrow, \perp\}$ are truth functionally complete sets. The set $\{\vee, \wedge\}$ is not.

Fundamental Semantic Concepts

And now we come to the key semantic concepts that we mentioned in the previous lecture:

- ▶ If a model V makes a formula φ true we say V **satisfies** φ , or φ is **true** in φ , and write $V \models \varphi$.
- ▶ If it is possible to find some model V that makes φ true, then we say φ is **satisfiable**.
- ▶ If φ is true, no matter what model V we use, then we say that φ is **valid** and write $\models \varphi$.

A Diplomatic Problem

You are chief of protocol for the embassy ball. The crown prince instructs you either to invite Peru or to exclude Qatar. The queen asks you to invite either Qatar or Romania or both. The king, in a spiteful mood, wants to snub either Romania or Peru or both. Who do you invite?

- ▶ Can we model this using just propositional logic?
- ▶ And what do we **gain** by doing that?
- ▶ What kind of questions can we "ask" our model?

Formalizing the Diplomatic Problem

- ▶ Three propositional symbols
 $P \equiv$ invite Peru $\neg P \equiv$ exclude Peru
 $Q \equiv$ invite Qatar $\neg Q \equiv$ exclude Qatar
 $R \equiv$ invite Romania $\neg R \equiv$ exclude Romania
- ▶ The problem can be formalized as
prince: $P \vee \neg Q \equiv$ invite Peru or exclude Qatar
queen: $Q \vee R \equiv$ invite Qatar or Romania or both
king: $\neg R \vee \neg P \equiv$ snub Romania or Peru or both
- ▶ Let $\Sigma = (P \vee \neg Q) \wedge (Q \vee R) \wedge (\neg R \vee \neg P)$. Solving the problem amounts to seeing whether Σ has a model (that is, whether it is possible to make all three formulas in Σ simultaneously true).

Solving the Diplomatic Problem: informal reasoning

- ▶ What can we deduce from Σ ?

$$\begin{array}{l} \text{prince: } P \vee \neg Q \quad \text{queen: } Q \vee R \\ \hline P \vee R \end{array}$$

- ▶ That is, one consequence of satisfying the prince and the queen is that we must invite Peru or Romania (or both).
- ▶ So, is Σ satisfiable? Yes, 2 out of 8 possible truth assignments satisfy Σ

$$\begin{array}{lll} P = \text{true} & Q = \text{true} & R = \text{false} \\ P = \text{false} & Q = \text{false} & R = \text{true} \end{array}$$

So **either** invite Peru and Qatar and not Romania
or invite Romania and not Peru and not Qatar

Solving the Diplomatic Problem 2: truth tables

- ▶ Is there a way of computing this solution?
- ▶ Yes. Use truth tables.

P	Q	R	$P \vee \neg Q$	$Q \vee R$	$\neg R \vee \neg P$	Σ
T	T	T	T	T	F	F
T	T	F	T	T	T	T
T	F	T	T	T	F	F
T	F	F	T	F	T	F
F	T	T	F	T	T	F
F	T	F	F	T	T	F
F	F	T	T	F	T	T
F	F	F	T	F	T	F

- ▶ This works — but it's about as exciting as watching paint dry. And may take considerably longer; truth tables are 2^n in the number of propositional symbols. There could be a lot of rice on the chessboard before we're finished ...

Inference in PL

So it's time to look at inference in PL more systematically. We'll look at all three tasks:

- ▶ Model checking
- ▶ Satisfiability building
- ▶ Validity checking

Inference in PL

So it's time to look at inference in PL more systematically. We'll look at all three tasks:

- ▶ Model checking (**Easy! As in very!**)
- ▶ Satisfiability building
- ▶ Validity checking

Inference in PL

So it's time to look at inference in PL more systematically. We'll look at all three tasks:

- ▶ Model checking (Easy! As in very!)
- ▶ Satisfiability building (We'll use tableaux to build models)
- ▶ Validity checking

Inference in PL

So it's time to look at inference in PL more systematically. We'll look at all three tasks:

- ▶ Model checking (Easy! As in very!)
- ▶ Satisfiability building (We'll use tableaux to build models)
- ▶ Validity checking (We'll see that tableaux can be used to determine validity)

Inference in PL

So it's time to look at inference in PL more systematically. We'll look at all three tasks:

- ▶ Model checking (Easy! As in very!)
- ▶ Satisfiability building (We'll use tableaux to build models)
- ▶ Validity checking (We'll see that tableaux can be used to determine validity)

Our discussion – and in particular, the tableaux method and the relationship between satisfiability checking and validity checking – will play an important role in subsequent lectures.

Model Checking

Suppose we are given a model V in which p is true and q is false, and we are asked to check whether $((p \wedge \neg q) \vee q) \rightarrow q$ is true in this model or not. We could just fill in the row of the truth table.

$$\begin{aligned} & ((p \wedge \neg q) \vee q) \rightarrow q \\ & ((\top \wedge \neg \perp) \vee \perp) \rightarrow \perp \\ & ((\top \wedge \top) \vee \perp) \rightarrow \perp \\ & (\top \vee \perp) \rightarrow \perp \\ & \top \rightarrow \perp \\ & \perp \end{aligned}$$

If you think of \top as 1 and \perp as 0 you can easily implement this in hardware (an idea that goes back to Shannon's Master's thesis).

So let's turn to satisfiability checking . . .

- ▶ We'll use tableaux to perform this task.
- ▶ A tableaux is essentially a tree-like data structure that records attempts to build a model.
- ▶ Tableaux are built by applying rules to an input formulas. These rules systematically tear the formula to detect all possible ways of building a model.
- ▶ Each branch of a tableaux records one way of trying to build a model. Some branches ("closed branches") don't lead to models. Others branching ("open branches") do.
- ▶ The best way to learn is via an example. . .

Tableaux for PI

Let's see if we can build a model for $(\neg(p \wedge q) \wedge \neg \neg r) \wedge p$.

Rules for \neg and \wedge

$$\begin{aligned} & \frac{(\varphi \wedge \psi)}{\varphi \quad \psi} (\wedge) \\ & \frac{\neg(\varphi \wedge \psi)}{\neg\varphi \quad \neg\psi} (\neg\wedge) \\ & \frac{\neg\neg\varphi}{\varphi} (\neg\neg) \end{aligned}$$

$$\begin{aligned} & (\neg(p \wedge q) \wedge \neg \neg r) \wedge p \\ & \neg(p \wedge q) \wedge \neg \neg r \\ & \quad p \\ & \quad \neg(p \wedge q) \\ & \quad \neg \neg r \\ & \quad \quad r \\ & \neg p \quad \neg q \\ & \text{Contradiction!!!} \quad \text{Model} \end{aligned}$$

Satisfiability and Validity are Dual

- ▶ A formula φ is valid iff $\neg\varphi$ is not satisfiable.
- ▶ A consequence of this observation is: if we have a method for solving the satisfiability problem (that is, if we have an algorithm for building models) then we have a way of solving the validity problem.
- ▶ Why? Because: to test whether φ is valid, simply give $\neg\varphi$ to the algorithm for solving satisfiability. If it can't satisfy it, then φ is not valid.
- ▶ Well, we have an algorithm for satisfiability (namely the tableaux method), so let's put this observation to work.

Validity via Tableaux

Let's show that $(p \wedge q) \rightarrow p$ is valid

Rules for \rightarrow

$$\begin{aligned} & \frac{\varphi \rightarrow \psi}{\neg\varphi \quad \psi} (\rightarrow) \\ & \frac{\neg(\varphi \rightarrow \psi)}{\varphi \quad \neg\psi} (\neg\rightarrow) \end{aligned}$$

$$\begin{aligned} & \neg((p \wedge q) \rightarrow p) \\ & p \wedge q \\ & \neg p \\ & p \\ & q \end{aligned}$$

Contradiction!!!

It is impossible to apply any more rules, and there are no open branches. Hence no model exists for the input $\neg\varphi$. Hence φ is valid.

Expressivity

- ▶ As we remarked earlier, it is possible to think about the semantics of PL in terms of relational structure.
- ▶ This gives us a way of comparing the expressivity of PL with the more powerful logics we shall study later.
- ▶ The idea is simple: think of PL as a way of talking about one element (!) relational structures of the form $\langle \{d\}, \{P_n\} \rangle$.
- ▶ That is, we have one individual, and one property for every propositional letter p_n (think of each P_n as a colour — we are covering the individual with coloured dots).
- ▶ This way of thinking about PL semantics is equivalent to the truth conditional semantics. Can you see why?
- ▶ That is, PL validity is completely determined by one element relational structures! Measured this way, its expressivity is low.

Computability

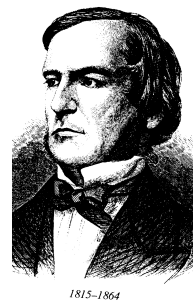
- ▶ We haven't directly said much about computability, but it should be clear that PL is a "computable logic".
- ▶ For a start, model checking is clearly computationally straightforward — it's linear in the length of the input formula.
- ▶ And checking satisfiability (and hence validity) is clearly computable too. The truth table method shows that we can do it in 2^n steps, where n is the number of propositional symbols in the input formal.
- ▶ Can we do better than 2^n steps? In particular, can the tableaux method do satisfiability/validity checking more efficiently.
- ▶ Sadly, it seem the answer is no.

What we Covered in this Lecture

- ▶ We introduced PL and dealt with all three inference tasks.
- ▶ Model checking was simple (linear-time algorithm) and satisfiability/validity checking turn out to be solvable using the same model building algorithm.
- ▶ We discussed a model building algorithm that looked more elegant than truth tables; it is more elegant, and often more efficient, but too turns out to share the same 2^n upper bound.

Relevant Bibliography

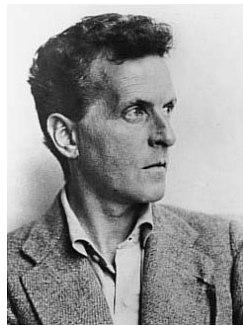
- ▶ One of the main founders of PL was George Boole (1815–1864), mathematician and philosopher.
- ▶ His book "An Investigation of the Laws of Thought" was one of the first mathematical treatments of logic, and one of the most important conceptual advances in logic from the Aristotelian syllogistic.



1815–1864

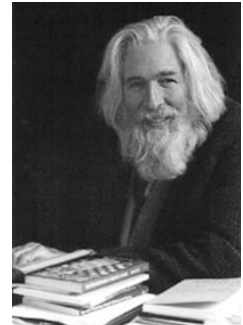
Relevant Bibliography

- ▶ The truth table method was pioneered by the philosopher Ludwig Wittgenstein (1889–1951) in his first famous philosophical work, the "Tractatus Logico-Philosophicus".
- ▶ He used the method in support of his celebrated "picture" theory of meaning.



Relevant Bibliography

- ▶ The tableaux method in the form presented here is due to Raymond Smullyan, logician, magician, and puzzle-supremo.
- ▶ His classic exposition of the method is in his book "First-Order Logic" (1968), which remains one of the best technical expositions of the subject.
- ▶ A better book for beginners is Graham Priest's "An introduction to non-classical logic" (2001), Cambridge University Press.



The Next Lecture

**How many Angels can Dance
on the Head of a Pin?**

Logics for Computation

Lecture #3: How many Angels can Dance on the Head of a Pin?

Carlos Areces and Patrick Blackburn
{carlos.areces,patrick.blackburn}@loria.fr

INRIA Nancy Grand Est
Nancy, France

ESSLLI 2008 - Hamburg - Germany

The Story so Far

- ▶ We've talk about **Logics** ...
- ▶ ...and we've seen an example: **PL**
- ▶ But how interesting is PL?
 - ▶ As a language it is pretty **simple**: $\forall, \neg, p, q, \dots$
 - ▶ It's relational models are **boring**: just a single, labelled point.
- ▶ How can we measure the **interestingness** of a Logic?
Any ideas?
- ▶ One way to do it:
 - 1) What can we **encode** in the language?
 - 2) How much does it **cost**?

Areces & Blackburn: Logics for Computation

INRIA Nancy Grand Est

What do we do Today?

How Many Angels can Dance on the Head of a Pin?
Or, how much can you encode in a 1-point structure?

- ▶ We will see that we can code quite complex problems in PL.
- ▶ In particular, we will show that we can code the **Graph Coloring problem**.
- ▶ Then, we will introduce an efficient algorithm for deciding satisfiability of PL-SAT: the **Davis Putnam algorithm**.

Areces & Blackburn: Logics for Computation

INRIA Nancy Grand Est

More than Diplomacy

- ▶ We saw yesterday a simple use of propositional logic in the "Diplomatic Problem".
- ▶ But the expressive power of PL is enough for doing many more interesting things:
 - ▶ graph coloring
 - ▶ constraint satisfaction problems (e.g., Sudoku)
 - ▶ hardware verification
 - ▶ planning (e.g., graphplan).
- ▶ Note that these problems have **real world applications**!

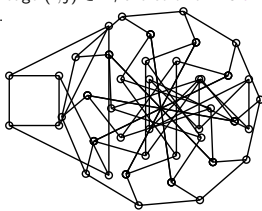
Areces & Blackburn: Logics for Computation

INRIA Nancy Grand Est

Graph Coloring

- ▶ **The Problem**: Given a graph $G = \langle N, E \rangle$ where N is a set of nodes and E a set of edges, and a fixed number k of **colors**. Decide if we can assign colors to nodes in N s.t.
 - ▶ All nodes are colored with one of the k colors.
 - ▶ For every edge $(i, j) \in E$, the color of i is **different** from the color of j .

- ▶ **An Example**:



Areces & Blackburn: Logics for Computation

INRIA Nancy Grand Est

Graph Coloring: The Nitty-Gritty Details

- ▶ We will use $n \times k$ propositional symbols that we write p_{ij} (n is the number of nodes in N , k the number of colors)
- ▶ We will read p_{ij} as **node i has color j**
- ▶ We have to say that
 1. Each node has (at least) one color.
 2. Each node has no more than one color.
 3. Related nodes have different colors.
- 1. **Each node has one color**: $p_{i1} \vee \dots \vee p_{ik}$, for $1 \leq i \leq n$
- 2. **Each node has no more than one color**: $\neg p_{ij} \vee \neg p_{im}$, for $1 \leq i \leq n$, and $1 \leq l < m \leq k$
- 3. **Neighboring nodes have different colors**: $\neg p_{il} \vee \neg p_{jl}$, for i and j neighboring nodes, and $1 \leq l \leq k$

Areces & Blackburn: Logics for Computation

INRIA Nancy Grand Est

Graph Coloring: Complexity

- ▶ Using the encoding in the previous page we effectively obtain for each graph G and k color, a formula $\varphi_{G,k}$ in PL such that **every model of $\varphi_{G,k}$ tell us a way of painting G with k colors**
- ▶ If \mathcal{M} is a model of $\varphi_{G,k}$ in which p_{ij} is true, then paint node i in G with color k .
- ▶ What have we done?!!!
 - ▶ Perhaps you know that graph coloring is a **difficult algorithmic problem**.
 - ▶ It is actually what is called an **NP-complete problem** (i.e., one of the hardest problems in the class of non-deterministic polynomial problems).
 - ▶ Assuming that, we just proved that **PL-SAT is also NP-complete**.

Areces & Blackburn: Logics for Computation

INRIA Nancy Grand Est

Decision Methods for PL

- ▶ The most traditional methods for solving the SAT problem for propositional logics (PL-SAT) behave as follows:
 - ▶ They always answer **SATISFIABLE** or **UNSATISFIABLE** after a finite time, for any input formula φ .
 - ▶ They always answer **correctly**.
- ▶ The best known complete methods probably are
 - ▶ truth tables
 - ▶ tableaux
 - ▶ axiomatics, Gentzen calculi, natural deduction, resolution
 - ▶ Davis-Putnam

Areces & Blackburn: Logics for Computation

INRIA Nancy Grand Est

Moving into Clausal Form

- **Clausal Form:** Write φ in conjunctive normal form (CNF)

$$\varphi = \bigwedge_{l \in L} \bigvee_{m \in M} \psi_{(l,m)}, \psi \text{ a literal (i.e., } p \text{ or } \neg p \text{)}.$$

This just means: **No conjunctions inside disjunctions**
Negations only on propositional symbols

- Using the following equivalences:

$$\begin{aligned}(\neg(p \vee q)) &\rightsquigarrow (\neg p \wedge \neg q) \\(\neg(p \wedge q)) &\rightsquigarrow (\neg p \vee \neg q) \\(\neg\neg p) &\rightsquigarrow p \\(p \vee (q \wedge r)) &\rightsquigarrow ((p \vee q) \wedge (p \vee r))\end{aligned}$$

The clause set associated to

$$(l_{11} \vee \dots \vee l_{1n_1}) \wedge (l_{21} \vee \dots \vee l_{2n_2}) \wedge \dots \wedge (l_{k1} \vee \dots \vee l_{kn_k}) \quad \text{is} \\ \{\{l_{11} \vee \dots \vee l_{1n_1}\} \wedge \{l_{21} \vee \dots \vee l_{2n_2}\} \wedge \dots \wedge \{l_{k1} \vee \dots \vee l_{kn_k}\}\}$$

Example

1. $\neg(\neg(p \vee q) \vee (\neg\neg q \vee (p \vee q)))$
2. $\neg(\neg(p \vee q) \vee (q \vee (p \vee q)))$
3. $(\neg\neg(p \vee q) \wedge \neg(q \vee (p \vee q)))$
4. $((p \vee q) \wedge \neg(q \vee (p \vee q)))$
5. $((p \vee q) \wedge (\neg q \wedge \neg(p \vee q)))$
6. $((p \vee q) \wedge (\neg q \wedge (\neg p \wedge \neg q)))$
7. $\{\{p, q\}, \{\neg q\}, \{\neg p\}, \{\neg q\}\}$
8. $\{\{p, q\}, \{\neg q\}, \{\neg p\}\}$

The Diplomatic Problem:

$$(P \vee \neg Q) \wedge (Q \vee R) \wedge (\neg R \vee \neg P) \\ \{\{P, \neg Q\}, \{Q, R\}, \{\neg R \vee \neg P\}\}$$

The Davis-Putnam Algorithm

- The Davis-Putnam method is perhaps one of the most widely used algorithms for solving the SAT problem of PL
- Despite its age, it is still one of the most popular and successful complete methods

Let Σ be the clause set associated to a formula φ

```
procedure DP( $\Sigma$ )
if  $\Sigma = \{\}$  then return SAT           // (SAT)
if  $\{\} \in \Sigma$  then return UNSAT     // (UNSAT)
if  $\Sigma$  has unit clause  $\{l\}$ 
then DP( $\Sigma \setminus \{l\}$ )             // (Unit Pr.)
Choose literal  $l$  and
if DP( $\Sigma \setminus \{l\}$ ) return SAT
then return SAT
else return DP( $\Sigma \setminus \{l\}$ )       // (Split)
```

Examples

$$\neg(\neg(p \vee q) \vee (\neg\neg q \vee (p \vee q))) \rightarrow \text{CNF} \rightarrow \{\{p, q\}, \{\neg q\}, \{\neg p\}\} \\ \{\{P, \neg Q\}, \{Q, R\}, \{\neg R \vee \neg P\}\}$$

DP: Performance

- The worst case complexity of the algorithm we show is $O(1,696^n)$, and a small modification moves it to $O(1,618^n)$.
- This is an improvement!... Notice that, for example,
 $2^{100} = 1.267.650.000.000.000.000.000.000.000$
 $1.696^{100} = 87.616.270.000.000.000.000.000.000$
 $1.618^{100} = 790.408.700.000.000.000.000.000$
- DP can reliably solve problems with up to 500 variables
- Sadly real world applications easily go into the **thouthands of variables** (remember coloring: $\#nodes \times \#colors$).
- But these is **worst time complexity**. You might get lucky...

You Might get Lucky

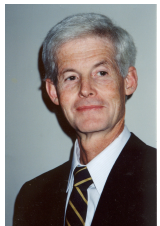
- Indeed, some method (called 'incomplete methods') rely in that you **might get lucky**.
- We can't cover them in the course, but intuitively,
 - they are stochastic methods
 - that randomly generate valuations
 - and try to maximize the probability that the valuation actually satisfies the input formula.
- Examples of these methods are **GSAT** and **WalkSAT**.
- For example, a k -coloring algorithm based on GSAT was **able to beat** specialize coloring algorithms.

What we Covered in this Lecture

- Our first **really computational** lecture of the course.
- We discussed the balance between **expressive power** and **complexity**.
 - We can code **complex problems** in PL (but the coding can be unintuitive, long, complex)
 - We have **efficient decision methods** for PL (able to cope with problems with hundres of propositional symbols, but our codings easily get into the thouthands).
- Still, no matter how nicely we paint them, 1-point relational structures are **booooooooooring**.

Relevant Bibliography I

Cook's Theorem: the satisfiability problem for propositional logic is NP-complete.

- That is, any problem in NP can be reduced in polynomial time to PL SAT.
- In plain English: if we find a cheap way of solving PL SAT, we'll also have a cheap way of solving **a hell of a lot more**. (Coda: probably there is no cheap way. Too good to be true. But still, it has not been shown. $P \stackrel{?}{=} NP$).
- Cook's Web page:
<http://www.cs.toronto.edu/~sacook/>
 Cook, Stephen (1971). *The complexity of theorem proving procedures*, Proceedings of the Third Annual ACM Symposium on Theory of Computing, 151–158.

Relevant Bibliography II

The Davis Putnam Algorithm

- ▶ It was **not** developed by Prof. David Putnam, but by Martin Davis and Hilary Putnam.
- ▶ It was then improved (with the split rule) by Martin Davis, George Logemann and Donald Loveland. The correct name is **DPLL**.
- ▶ Davis' Web page: <http://www.cs.nyu.edu/cs/faculty/davism/>



Davis, Martin; Putnam, Hilary (1960). *A Computing Procedure for Quantification Theory*. Journal of the ACM 7 (1): 201–215.



Davis, Martin; Logemann, George, and Loveland, Donald (1962). *A Machine Program for Theorem Proving*. Communications of the ACM 5 (7): 394–397.



Interesting Links

- ▶ The **SATLive!** page. The page for all things SAT
<http://www.satlive.org/>
- ▶ The international SAT Competitions web page
<http://www.satcompetition.org/>
- ▶ Some provers
 - ▶ **RSat** <http://reasoning.cs.ucla.edu/rsat/>
 - ▶ **MiniSat** <http://minisat.se/>
 - ▶ **PicoSat** <http://fmv.jku.at/picosat/>
- ▶ The Walksat Home Page
<http://www.cs.rochester.edu/~kautz/walksat/>

The Next Lecture

Points & Lines™

Bored of working always with the same old point?
UPGRADE your model! Get OTHER points and even LINES!
(Offer available for a limited time)

Logics for Computation

Lecture #4: Points & Lines™

Bored of working always with the same old point?
UPGRADE your model! Get OTHER points and even LINES!
(Offer available for a limited time).

Carlos Areces and Patrick Blackburn
{carlos.areces,patrick.blackburn}@loria.fr

INRIA Nancy Grand Est
Nancy, France

ESLLI 2008 - Hamburg - Germany

The Story So Far

- ▶ We looked at SAT-solving (that is, model building) for propositional calculus in some detail.
- ▶ In particular, we discussed the Davis-Putnam algorithm.
- ▶ We also briefly met the concept of an NP-complete problem.
- ▶ In a nutshell, what we learned was this: although PL looks very simple, it is actually capable of coding some very tough problems indeed.

Areces & Blackburn: Logics for Computation

INRIA Nancy Grand Est

Diamonds are forever!

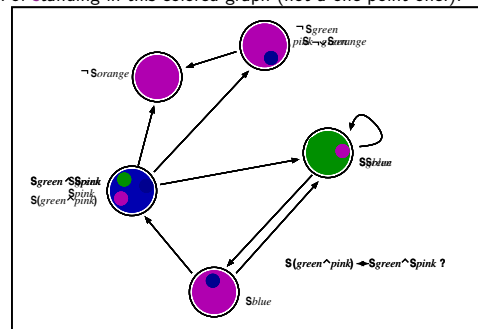
- ▶ But let's face it, we're here to work with relational structures. And although we saw yesterday that PL has a semantics in terms of one-point relational structures (wow!), and although we saw today that PL can, in a certain, code up information about graphs, PL isn't exactly our dream language.
- ▶ Why not? Because relational structures are full of points and lines and other nice things, and we really want to be able to get our hands on these directly! We want to be able to describe them, to compute with them, and to draw inferences about them. Using PL for this is like stroking a cat while wearing a suit of armor!
- ▶ In this lecture we introduce a special language which let's us do this: we call the **diamond language**. How will this language work. From the inside ...

Areces & Blackburn: Logics for Computation

INRIA Nancy Grand Est

Simple Structures / Simple Languages

Think of standing in this colored graph (not a one-point one!):

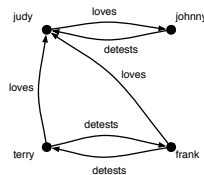


Areces & Blackburn: Logics for Computation

INRIA Nancy Grand Est

Graphs with multiple relations

The previous sequence of slides motivates the language we will use except that we will use a diamond symbol \Diamond instead of S , and it also motivates the way we will define the semantics (we will continue to "stand inside models"). But there is more addition to make.



The previous graphs only had one relation. We often want to work with more than one relation (as in the above relational structure) so we will want multiple diamonds, one for each relation.

Areces & Blackburn: Logics for Computation

INRIA Nancy Grand Est

Diamond languages: Syntax

We build the diamond language on top of PL. It is a very simple extension. First we decide how many relations R we want to work with, and then we add the following two symbols for each R :

$$\langle R \rangle \quad [R]$$

If we are only going to work with a single relation, we usually write these as:

$$\Diamond \quad \Box$$

We then extend the definition of formula by saying that if φ is a formula, then so are $\langle R \rangle \varphi$ and $[R] \varphi$.

Areces & Blackburn: Logics for Computation

INRIA Nancy Grand Est

Diamond languages: Semantics

Suppose we are given a relational structure

$$\langle W, \{R_n\}, \{P_m\} \rangle$$

where we have one relation R for each diamond $\langle R \rangle$, and one property P for each proposition symbol p . Then we define:

$$\begin{aligned} \mathcal{M}, w \models p & \text{ iff } w \in P, \\ \mathcal{M}, w \models \neg \varphi & \text{ iff not } \mathcal{M}, w \models \varphi \text{ (notation: } \mathcal{M}, w \not\models \varphi), \\ \mathcal{M}, w \models \varphi \wedge \psi & \text{ iff } \mathcal{M}, w \models \varphi \text{ and } \mathcal{M}, w \models \psi, \\ \mathcal{M}, w \models \langle R \rangle \varphi & \text{ iff for some } v \in W \text{ such that } Rww \\ & \text{ we have } \mathcal{M}, v \models \varphi, \\ \mathcal{M}, w \models [R] \varphi & \text{ iff for all } v \in W \text{ such that } Rww \\ & \text{ we have } \mathcal{M}, v \models \varphi. \end{aligned}$$

Areces & Blackburn: Logics for Computation

INRIA Nancy Grand Est

We don't need both boxes and diamonds



$$\Box \varphi \text{ is } \neg \Diamond \neg \varphi.$$

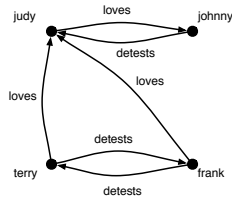
$$\Diamond \varphi \text{ is } \neg \Box \neg \varphi.$$

So, like WALL-E, we can choose either the diamond or the box — but we choose the diamond!

Areces & Blackburn: Logics for Computation

INRIA Nancy Grand Est

Waterloo Sunset ...

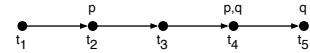


$\langle \text{LOVES} \rangle \top \wedge \langle \text{DETESTS} \rangle \langle \text{LOVES} \rangle \top$

Note that this is true when evaluated at Terry

A temporal model

The following relational structure is meant to be a simple “flow of time”. We are interested in the **transitive closure** of the relation indicated by the arrows.



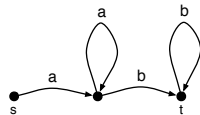
- ▶ $\Diamond q$ is true at t_1, t_2, t_3 and t_4 .
- ▶ $\Diamond(p \wedge q)$ is satisfied at points t_1, t_2 and t_3 (because all these points are to the left of t_4 where both p and q are true together) but it is not satisfied at t_4 and t_5 .
- ▶ Note that for any formula φ whatsoever, $\Box\varphi$ is satisfied at t_5 . Why?

Example

All formulas of the form

$\langle a \rangle \dots \langle a \rangle \langle b \rangle \dots \langle b \rangle t$

(that is, an unbroken block of $\langle a \rangle$ diamonds preceding an unbroken block of $\langle b \rangle$ diamonds in front of a proposition symbol t which is only true at the terminal node t) are satisfied at the start node s of the following structure:



Note that diamond sequences of this form correspond to the strings accepted by the automaton.

Fundamental Semantic Concepts

- ▶ A formula φ is **globally satisfied** (or **globally true**) in a model \mathcal{M} if it is satisfied at all points in \mathcal{M} , and if this is the case we write $\mathcal{M} \models \varphi$.
- ▶ A formula φ is **valid** if it is globally satisfied in all models, and if this is the case we write $\models \varphi$.
- ▶ A formula φ is **satisfiable in a model** \mathcal{M} if there is some point in \mathcal{M} at which φ is satisfied, and φ is **satisfiable** if there is some point in some model at which it is satisfied.

Two example validities

The following two formulas are validities:

- ▶ $\langle R \rangle (p \wedge q) \rightarrow \langle R \rangle p \wedge \langle R \rangle q$
- ▶ $\langle R \rangle \top \wedge \neg \langle R \rangle \neg p \rightarrow \langle R \rangle p$

Can you see why? Note: you’ve seen the first one already ...

Inference and computation in the diamond language

- ▶ It should be clear that defining an inference systems (such as tableaux systems) for the diamond language is going to be trickier (and more interesting!) than for PL.
- ▶ It should also be clear that there are computational issues to be settled — there is no obvious way to prove that the diamond language is computable by “counting models” as we did with PL.
- ▶ That is, we have traded in some tractability for expressivity.
- ▶ We’ll look at other inferential/computational issues in the next lecture. For now, we’ll simply look briefly at model checking, which is still easy, before turning to expressivity.

Model checking I

Use a bottom-up **labeling algorithm**. To model check a formula φ :

- ▶ Label every point in the model with all the subformulas of φ that are true at that point.
- ▶ We start with the proposition symbols: the valuation tells us where these are true, so we label all the appropriate points.
- ▶ We then label with more complex formulas. The booleans are handled in the obvious way: for example, we label w with $\psi \wedge \theta$ if w is labeled with both ψ and θ .
- ▶ As for the modalities, we label w with $\Diamond\varphi$ if one of its R -successors is labeled with φ , and we label it with $\Box\varphi$ if all of its R -successors are labeled with φ .

Model Checking 2

- ▶ The beauty of this algorithm is that we never need to duplicate work: once a point is labeled as making φ true, that’s it.
- ▶ This makes the algorithm run in time polynomial in the size of the input formula and model: the algorithm takes time of the order of

$$\text{con}(\varphi) \times \text{nodes}(\mathcal{M}) \times \text{nodes}(\mathcal{M}),$$

where $\text{con}(\varphi)$ is the number of connectives in φ , and $\text{nodes}(\mathcal{M})$ is the number of nodes in \mathcal{M} .

- ▶ To see this, note that $\text{con}(\varphi)$ tells us how many rounds of labeling we need to perform, one of the $\text{nodes}(\mathcal{M})$ factors is simply the upper bound on the nodes that need to be labeled, while the other is the upper bound on the number of successor nodes that need to be checked.

Model checking is important

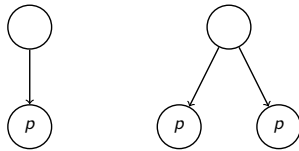
- ▶ Although this looks simple, this kind of model checking is important.
- ▶ It is possible to strengthen the diamond language in various ways to make it excellent for talking about graph structures representing hardware chips and much else besides.
- ▶ The above algorithm can (and have been) extended to such languages and applied to industrial hardware and software design problems.
- ▶ Model checking is the simplest form of inference — but it is important and useful.

Expressivity and bisimulation

- ▶ What can we say with diamonds? And what can't we say? That is, how expressive is our diamond language?
- ▶ We will ask ourselves the following question: how good are diamond languages at *distinguishing between models*?
- ▶ Let's look at an example...

Are these model diamond distinguishable?

Are these two models diamond distinguishable? To make the question more precise: is there an diamond formula that is true at the root node of one model, and not at the other?



No. There is no diamond formula that distinguishes them. The diamond language, it seems, cannot count!

Now the key question: why exactly can't the diamond language distinguish the two models?

Bisimulations (informal)

Two models are bisimilar if their points can be related in so that:

- ▶ Related points make the same propositional symbols true.
- ▶ If you make a transition in one model, you can make "matching" transition in the other. Here "matching" means that the points you reach by making the transitions are related.

Bisimulations (formal definition)

Here is the formal definition of bisimulation (for models with one relation R). A bisimulation between models $\mathcal{M} = (W, R, V)$ and $\mathcal{M}' = (W', R', V')$ is a non-empty binary relation E between their domains (that is, $E \subseteq W \times W'$) such that whenever wEw' we have that:

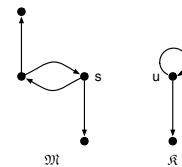
Atomic harmony: w and w' satisfy the same proposition symbols,

Zig: if Rww , then there exists a point v' (in \mathcal{M}') such that vEv' and $R'w'v'$, and

Zag: if $R'w'v'$, then there exists a point v (in \mathcal{M}) such that vEv' and Rww .

Bisimilar or not?

Are these two models bisimilar or not (assume all propositional symbols are false all points)?

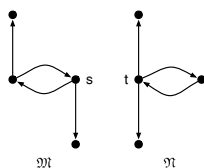


If they are bisimilar, what is the bisimulation? If they are not bisimilar, what is a formula that distinguishes them?

Yes, they are bisimilar; to this this, bend the upward-pointing arrow on the left of the left-hand model downwards.

Bisimilar or not?

Are these two models bisimilar or not (assume all propositional symbols are false all points)?



If they are bisimilar, what is the bisimulation? If they are not bisimilar, what is a formula that distinguishes them?

$\Box(\Box \perp \vee \Diamond \Box \perp)$ is a formula that distinguishes these models: it is true in \mathcal{M} at s , but false in \mathcal{N} at t .

What we Covered Today

- ▶ We motivated the diamond language, defined its syntax and semantics, and gave example of the concepts of satisfaction at a point in a model, and of validity.
- ▶ We showed that model checking with diamonds is a simple task.
- ▶ We discussed expressivity, introducing and illustrating the concept of bisimulation.
- ▶ However we also briefly remarked that it is not immediately obvious if the diamond language is computable (there is no obvious way of counting models as there was for PL), and it is not yet clear how to handle satisfiability/validity checking.

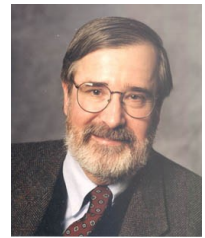
Relevant Bibliography I

Saul Kripke is the person largely responsible for the relational semantics for the diamond language. In fact, when working with diamond languages, relational structures are often called Kripke models. Kripke, a child prodigy, was 15 when he developed his first ideas on the topic. Kripke has many other substantial contributions to logic and philosophy.



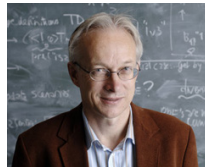
Relevant Bibliography II

Nowadays, diamond languages (and sophisticated extensions and variants) are used far more in computer science than philosophy. Moreover, in that community they are prized for their model checking properties. Model checking is one of the big success stories of computer science: theory and applications combine beautifully. Ed Clarke, one of the pioneers, won the Turing Prize in 2007 for his contributions to the field.



Relevant Bibliography III

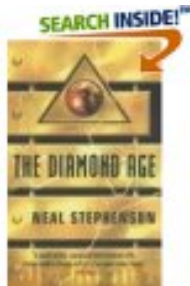
He wears a crown of diamonds — Johan van Benthem, the king of bisimulation. The researcher who, more than any other, made people see how general and useful diamond languages and their relatives actually are, and who is largely responsible for creating the LLI community.



Relevant Bibliography

- ▶ A New Introduction to Modal Logic, by George Hughes and Max Cresswell, Routledge, 1996.
- ▶ Modal Logic: an Introduction, by Brian Chellas, Cambridge University Press. 1996.
- ▶ Modal logic: a Semantic Perspective, by Patrick Blackburn and Johan van Benthem, pages 1–84 in *Handbook of Modal Logic*, edited by Blackburn, van Benthem, and Wolter. Elsevier, 2007. Download at <http://www.loria.fr/~blackbur/papers/blackvanben.pdf>
- ▶ *The Description Logic Handbook Theory, Implementation and Applications*, edited by Baader, Calvanese, McGuinness, Nardi, Patel-Schneider. Cambridge University Press, 2003.

Relevant Bibliography IV



The Next Lecture

Trees, and how to cut them

Logics for Computation

Lecture #5: About Trees, and How to Cut Them

Carlos Areces and Patrick Blackburn
{carlos.areces,patrick.blackburn}@loria.fr

INRIA Nancy Grand Est
Nancy, France

ESSLLI 2008 - Hamburg - Germany

The Story so Far

- ▶ We have introduced the $\langle R \rangle$ operator to talk about complex relational structures.
Nothing fancy (yet), just a simple extension of PL.
- ▶ We have used it to describe some properties over models.
E.g., the following formulas are valid: $\langle R \rangle(p \wedge q) \rightarrow \langle R \rangle p$
 $\langle R \rangle \top \wedge \neg \langle R \rangle \neg p \rightarrow \langle R \rangle \neg p$
- ▶ We have discussed when two models are the same.
- ▶ We have seen an algorithm to check whether a formula is true in a given model.

Areces & Blackburn: Logics for Computation

INRIA Nancy Grand Est

What do we do Today

- ▶ We will define a **tableaux algorithm** for satisfiability of formulas containing $\langle R \rangle$.
 - ▶ We already know how to check, given a model, if the formula holds in the model (**model checking**).
 - ▶ Today, we will see how do we check whether a formula has a model.
- ▶ We will also go back to the question **When are two models the same?**
 - ▶ and talk about trees ...
 - ▶ ... and how to cut them.

Areces & Blackburn: Logics for Computation

INRIA Nancy Grand Est

Counting models

- ▶ The proof that the satisfiability problem for PL is **decidable** is very simple:
 - ▶ Suppose that you are given a formula φ and you are looking for a model of φ .
 - ▶ First note that propositional symbols that do not appear in φ are **irrelevant**.
 - ▶ We know that our models has **only one point**.
 - ▶ Hence, we only need to list all possible ways of labelling that single node with propositional symbols in φ .
- ▶ What about the $\langle R \rangle$ language?

Areces & Blackburn: Logics for Computation

INRIA Nancy Grand Est

The Tableaux Method for Relational Structures

- ▶ We want to devise a tableau method for the language we introduced to talk about complex relational structures.
- ▶ Let's review the **tableaux method** that we introduced for propositional logic:

$\frac{s:(\varphi \wedge \psi)}{s:\varphi \quad s:\psi} (\wedge)$	▶ Pretty neat: 3 rules for an NP-complete problem!
$\frac{s:\neg(\varphi \wedge \psi)}{s:\neg\varphi \quad s:\neg\psi} (\neg\wedge)$	▶ But now we want to deal with more than a single point .
$\frac{s:\neg\varphi \quad s:\neg\psi}{s:\neg\varphi \quad s:\neg\psi} (\neg\neg)$	▶ The solution is: labels! \square
	▶ They will help us keep track of what is going on in each point in our model.

Areces & Blackburn: Logics for Computation

INRIA Nancy Grand Est

Now Lines!

- ▶ We have dealt in the previous slide with **multiple points**.
What about lines?
- ▶ Remember that the operator we introduced to **talk about lines** in our language was $\langle R \rangle \varphi$ and we said that
 $\mathcal{M}, w \models \langle R \rangle \varphi$ iff there is w' s.t. wRw' and $\mathcal{M}, w' \models \varphi$.

- ▶ Start with the labelled formula $s:\langle R \rangle \varphi$.
If this formula is satisfiable, it is because there is an R -successor t where φ holds.
$$\frac{s:\langle R \rangle \varphi}{s:\varphi \quad t:\varphi} ((R))$$

for t a new label
- ▶ Start with the labelled formula $s:\neg\langle R \rangle \varphi$.
If there is an R -successor t , then φ should not hold at t .
$$\frac{s:\neg\langle R \rangle \varphi}{s:\neg\varphi \quad t:\neg\varphi} (\neg(R))$$

Areces & Blackburn: Logics for Computation

INRIA Nancy Grand Est

The Complete Cast, plus an Example

$\frac{s:(\varphi \wedge \psi)}{s:\varphi \quad s:\psi}$	$\frac{s:\langle R \rangle \varphi}{sRt \quad t:\varphi}$	$s:(\neg\langle R \rangle p \wedge (\langle R \rangle q \wedge \langle R \rangle p))$ $s:\neg\langle R \rangle p$ $s:(\langle R \rangle q \wedge \langle R \rangle p)$ $s:\langle R \rangle q$ $s:\langle R \rangle p$ sRt $t:q$ sRu $u:p$ $t:\neg p$ $u:\neg p$
$\frac{s:\neg(\varphi \wedge \psi)}{s:\neg\varphi \quad s:\neg\psi}$	$\frac{s:\neg\langle R \rangle \varphi}{sRt \quad t:\neg\varphi}$	

for t a new label

contradiction!!!

- ▶ Which are the **similarities/differences** with tableaux for PL?
- ▶ How do we know that **we got it right**?
- ▶ What can we **learn** from the calculus?

Areces & Blackburn: Logics for Computation

INRIA Nancy Grand Est

A Closer Look

- ▶ Which **similarities / differences** with tableaux for PL?
 - ▶ Does the calculus **terminate**?
 - ▶ What are **labels**? What are they doing? Can we use them?
 - ▶ Is this an **algorithm**?
 - ▶ Is it a **good** algorithm?

- ▶ Did we **get it right**?
 - ▶ Did we get it right in the PL case, to start with?
Consider the rule:
- ▶ What can we **learn** from the calculus?
 - ▶ Something **about** models!

$\frac{s:(\varphi \wedge \psi)}{s:\varphi \quad s:\psi}$	$\frac{s:\langle R \rangle \varphi}{sRt \quad t:\varphi}$	$s:\neg(\varphi \wedge \psi)$ $s:\neg\varphi \quad s:\neg\psi$ $s:\neg\varphi$ $s:\neg\psi$
$\frac{s:\neg(\varphi \wedge \psi)}{s:\neg\varphi \quad s:\neg\psi}$	$\frac{s:\neg\langle R \rangle \varphi}{sRt \quad t:\neg\varphi}$	

for t a new label

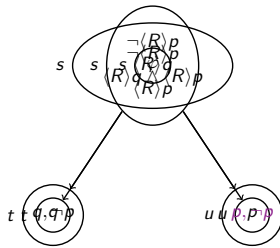
Areces & Blackburn: Logics for Computation

INRIA Nancy Grand Est

Tree Models

- Let us see the tableaux proof we did before again, for the formula

$$\varphi = \neg \langle R \rangle p \wedge (\langle R \rangle q \wedge \langle R \rangle p)$$



$$\begin{aligned} & s: \neg \langle R \rangle p \wedge (\langle R \rangle q \wedge \langle R \rangle p) \\ & s: \neg \langle R \rangle p \\ & s: \langle R \rangle q \wedge \langle R \rangle p \\ & s: \langle R \rangle q \\ & s: \langle R \rangle p \\ & sRt \\ & t: q \\ & sRu \\ & u: p \\ & t: \neg p \\ & u: \neg p \end{aligned}$$

Tree and Finite Model Properties

- Using the rules of the tableaux calculus we only explore **finite, tree models**.

- Let's assume that the calculus is correct (you will have to believe me).

- Then the $\langle R \rangle$ -language

- cannot say **infinite**,
- cannot say **non-tree**.

$\frac{s: (\varphi \wedge \psi)}{s: \varphi}$	$\frac{s: \langle R \rangle \varphi}{sRt} \quad \text{for } t \text{ a new label}$
$\frac{s: \neg \langle R \rangle \varphi}{s: \neg \psi}$	$\frac{s: \neg \langle R \rangle \varphi}{sRt} \quad \text{for } t \text{ a new label}$

Theorem: A formula in the $\langle R \rangle$ -language is satisfiable if and only if it is satisfiable in a finite, tree relational structure.

What we Covered in this Lecture

- We introduce a tableaux method to check satisfiability for the language with $\langle R \rangle$.
- We saw that we can use **labels** to describe what is going on in each point of a relational structure.
- More importantly: we saw that tableaux are a way to **systematically explore** relational structures.
- Actually, from the tableaux algorithm we could learn some model properties: **we only need to consider finite tree models**.

Relevant Bibliography I

Tableaux Algorithms

- The Tableaux Method is the core algorithm of most **current theorem provers** for relational languages.
- You might have heard about **description logics**. Racer, FaCT++, Pellet are all based on a calculus similar to the one we studied today.
- Fitting's Web page: <http://comet.lehman.cuny.edu/fitting/>



Fitting, Melvin (1983). *Proof Methods for Modal and Intuitionistic Logics*. D. Reidel Publishing Co., Dordrecht.



Relevant Bibliography II

Tree and Finite Model Properties

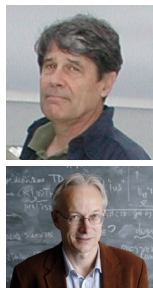
- Unraveling**, the procedure to turn arbitrary models into trees, was introduced by Segerberg.
- Segerberg's Web page: <http://www.phil.ualgary.ca/philosophy/people/segerberg.html>
- A more general result about **turning things into other things** can be proved using bisimulations.
- van Benthem's Web page: <http://staff.science.uva.nl/~johan/>



Segerberg, Krister (1971). *An Essay in Classical Modal Logic*, Department of Philosophy Uppsala University, Sweden. Uppsala Philosophical Studies.



van Benthem, Johan (1985). *Modal Logic and Classical Logic*, Bibliopolis.



Interesting Links

- Some provers for the $\langle R \rangle$ language based on the tableaux algorithm:
 - Racer** <http://www.racer-systems.com/>
 - FaCT++** <http://owl.man.ac.uk/factplusplus/>
 - Pellet** <http://pellet.owldl.com/>
 - HTab** <http://trac.loria.fr/projects/htab/wiki>
- Some based on **other algorithms**:
 - MSPass** (translation based) <http://www.cs.man.ac.uk/~schmidt/mspass/>
 - HyLoRes** (resolution based) <http://trac.loria.fr/projects/hylores>

The Next Lecture

No Way to Say Warm in French

Logics for Computation

Lecture #6: No Way to Say Warm in French

Carlos Areces and Patrick Blackburn
{carlos.areces,patrick.blackburn}@loria.fr

INRIA Nancy Grand Est
Nancy, France

ESSLLI 2008 - Hamburg - Germany

The Story so Far

- ▶ We are working with the $\langle R \rangle$ language
- ▶ We saw that the language cannot say
 - ▶ "I am not a tree"
 - ▶ "I am infinite"
- ▶ On the positive side
 - ▶ It has a simple reasoning calculus: labelled tableaux
 - ▶ It is decidable.

Areces & Blackburn: Logics for Computation

INRIA Nancy Grand Est

What do we do Today

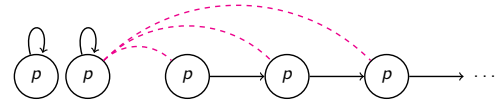
- ▶ We will extend the expressive power of the language...
- ▶ ...and explore quite a number of possibilities.
- ▶ We will learn to count till n .
- ▶ We will learn to name nodes.
- ▶ We will learn to say "everywhere".
- ▶ We will learn to say "it won't take forever".

Areces & Blackburn: Logics for Computation

INRIA Nancy Grand Est

One is the Same as Infinite

- ▶ We said in the previous lecture that we cannot say **infinite** in the $\langle R \rangle$ language.
- ▶ Let's see this in more detail. Consider the model:



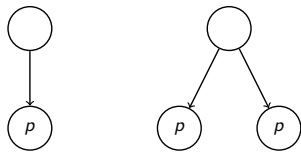
- ▶ This is **not** a tree. Hence, there should be a tree like structure which should be **the same** as this one for the $\langle R \rangle$ language.

Areces & Blackburn: Logics for Computation

INRIA Nancy Grand Est

One is the Same as Two

- ▶ But let's consider a simpler example (after all, **infinite** is quite a big number).
- ▶ We saw that the $\langle R \rangle$ language cannot distinguish between **one** and **two**!!!

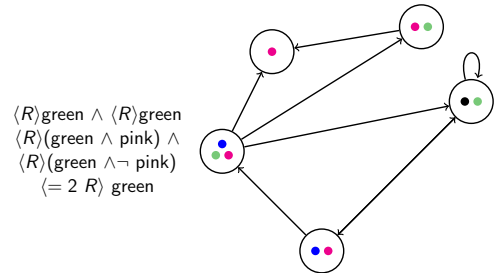


Areces & Blackburn: Logics for Computation

INRIA Nancy Grand Est

Learning to Count

Suppose we want to say that **two green nodes** are accessible ...



$\langle R \rangle \text{green} \wedge \langle R \rangle \text{green}$
 $\langle R \rangle (\text{green} \wedge \text{pink}) \wedge$
 $\langle R \rangle (\text{green} \wedge \neg \text{pink})$
 $\langle = 2 R \rangle \text{green}$

Areces & Blackburn: Logics for Computation

INRIA Nancy Grand Est

Alice in Wonderland

Humpty Dumpty: When I use a word, it means just what I choose it to mean – neither more nor less.

Alice: The question is, whether you can make words mean so many different things.

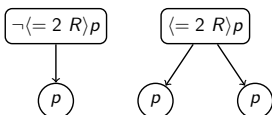
Humpty Dumpty: The question is: which is to be master – that's all.

- ▶ If the language **cannot express** something we are interested in, we just **extend the language**!

- ▶ **Counting successors:**

$$\mathcal{M}, w \models \langle = n R \rangle \varphi \text{ iff } |\{w' \mid wRw' \text{ and } \mathcal{M}, w' \models \varphi\}| = n$$

- ▶ Clearly:



The models are not **the same** for the $\langle = n R \rangle$ language.

Areces & Blackburn: Logics for Computation

INRIA Nancy Grand Est

Extending the Language

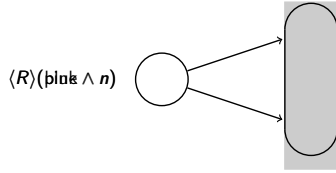
- ▶ What other things **we cannot say** in the $\langle R \rangle$ language?
- ▶ Plenty:
 1. In that particular node.
 2. Everywhere in the model.
 3. In a finite number of steps.
- ▶ ...
- ▶ Luckily, as Humpty Dumpty says, **we are the masters**, and we can design the language that better pleases us.
- ▶ Let's get to work...

Areces & Blackburn: Logics for Computation

INRIA Nancy Grand Est

Names for Points

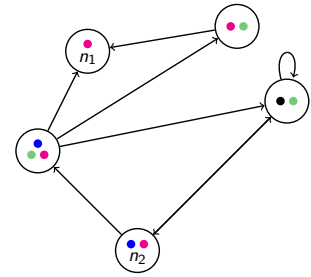
Suppose that n is a **name** for a point. That is, it can **label** a unique point in any relational structure.



- ▶ Looks useful...
- ▶ We can introduce names into our language (you probably know them as **constants**).

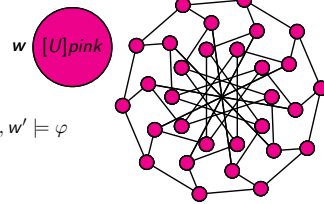
Names for Points

- ▶ When we allow names in our language, our models will look like this:
- ▶ We have already used something like **names**. Anybody remembers when?
- ▶ Tableaux for the $\langle R \rangle$ language!
- ▶ If we also introduce the **:-operator** we can write things like
 - $n_1:\text{pink}$
 - $n_2:\langle R \rangle \text{pink}$
 - $n_2:\langle R \rangle \langle R \rangle n_2$



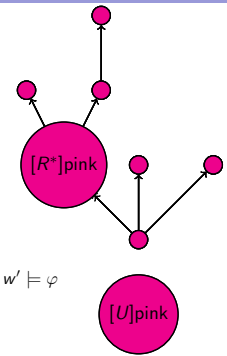
Everywhere in the model

- ▶ Suppose we want to paint everything **pink** (we love pink).
- ▶ Can we do it? Let's see and example, consider this model:
- ▶ Is there a formula of the $\langle R \rangle$ language, that we can make true at w , so that **pink** is true everywhere in the model?
- ▶ Define the $[U]$ operator as $\mathcal{M}, w \models [U]\varphi$ iff for all $w', \mathcal{M}, w' \models \varphi$
- ▶ Then $\mathcal{M}, w \models [U]\text{pink}$ if the whole model is **pink**.
- ▶ Jah!



In a Finite Number of Steps

- ▶ In the $\langle R \rangle$ language we can say
 - ▶ In one step $p: \langle R \rangle p$
 - ▶ In two steps $p: \langle R \rangle \langle R \rangle p$
 - ▶ ...
 - ▶ But we cannot say, in a **finite** (zero or more, but unspecified) number of steps p :
 $p \vee \langle R \rangle p \vee \langle R \rangle \langle R \rangle p \vee \dots$
- ▶ Define the $\langle R^* \rangle$ operator as $\mathcal{M}, w \models \langle R^* \rangle \varphi$ iff there is w' s.t. wR^*w' and $\mathcal{M}, w' \models \varphi$ for R^* is the **reflexive and transitive closure** of R . (Let's write $[R^*]\varphi$ for $\neg \langle R^* \rangle \neg \varphi$)
- ▶ Pretty choosy! (ok, let's say **selective**)



The Complete Menu

Our models are structures like $\mathcal{M} = \langle W, \{R_i\}, \{P_i\}, \{N_i\} \rangle$

- ▶ Counting successors:
 $\mathcal{M}, w \models \langle n R \rangle \varphi$ iff $|\{w' \mid wRw' \text{ and } \mathcal{M}, w' \models \varphi\}| = n$.
 (compare with $\mathcal{M}, w \models \langle R \rangle \varphi$ iff there is w' s.t. wRw' and $\mathcal{M}, w' \models \varphi$.)
- ▶ Names and the $:-$ operator:
 $\mathcal{M}, w \models n_i$ iff $w = N_i$
 (compare with $\mathcal{M}, w \models p_i$ iff $w \in P_i$)
 $\mathcal{M}, w \models n_1:\varphi$ iff $\mathcal{M}, N_1 \models \varphi$
 (compare with $\mathcal{M}, w \models \langle R \rangle \varphi$ iff there is w' s.t. wRw' and $\mathcal{M}, w' \models \varphi$.)
- ▶ The $[U]$ -operator:
 $\mathcal{M}, w \models [U]\varphi$ iff for all $w', \mathcal{M}, w' \models \varphi$
 (compare with $\mathcal{M}, w \models \langle R \rangle \varphi$ iff there is w' s.t. wRw' and $\mathcal{M}, w' \models \varphi$.)

Back to the Intuitions!

The main idea we want to get across is:

There are plenty of options, go and chose what you need!

Even more, if it is not there, then define it yourself

Remember what Humpty Dumpty said:
 "The question is: Which is to be master"

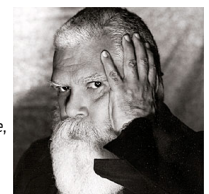
- ▶ By combining the operators we have been discussing we obtain a wide variety of languages.
 - ▶ We go from languages of low expressivity (PL) to languages of high expressivity (the selective $[R^*]$).
 - ▶ We go from languages of 'low' complexity (NP-complete) to languages of high complexity (EXPTIME-complete).
- ▶ By choosing the right expressivity for a given application we will pay the exact price required.

What we Covered Today

- ▶ We discussed the **polytheistic approach** in full glory:
 - ▶ counting
 - ▶ constants
 - ▶ universal quantification
 - ▶ reflexive and transitive closure
- ▶ By combining all these operators we obtain very diverse logics.
- ▶ But, they all share the same semantics: relational structures!
- ▶ They are just **different ways of talking** about something.

Relevant Bibliography I

[...] No way to say *warm* in French. There was only *hot* and *tepid*. If there's no word for it, how do you think about it? [...] Imagine, in Spanish having to assign a gender to every object: dog, table, tree, can-opener. Imagine, in Hungarian, not being able to assign a gender to anything: *he*, *she*, *it* all the same word.



- ▶ My French is not good enough to say if it's true...
- ▶ But it's definitely a **great science fiction book!**

Delany, Samuel (1966). *Babel-17*. Ace Books.

Relevant Bibliography II

- ▶ Many of the languages that we have been discussing are investigated in detail in the area known as **Modal Logics**.
- ▶ The name 'modal' (in many cases as opposed to 'classical') doesn't make much sense.
- ▶ Some of these languages have been extensively studied by somebody you know quite well by now.
Blackburn's Web page: <http://www.loria.fr/~blackbur>
- ▶ M. de Rijke also pushed the idea of working with modal logics extending the $\langle R \rangle$ language.
de Rijke's Web page: <http://staff.science.uva.nl/~mdr/>



Blackburn, Patrick and van Benthem, J (2006). *Chapter 1 of the Handbook of Modal Logics*, Blackburn, P.; Wolter, F.; and van Benthem, J., editors, Elsevier.



de Rijke, Maarten (1993). *Extending Modal Logic* PhD Thesis. Institute for Logic, Language and Computation, University of Amsterdam.



The Next Lecture

DIY First Order Logic

Logics for Computation

Lecture #7: DIY First-Order Logic

Carlos Areces and Patrick Blackburn
{carlos.areces,patrick.blackburn}@loria.fr

INRIA Nancy Grand Est
Nancy, France

ESSLLI 2008 - Hamburg - Germany

The Story so Far

- ▶ Yesterday we saw how we can introduce different operators...
- ▶ ...and 'cook' our own logic.
- ▶ Now, Patrick wants to talk about **First Order Logic** after the pause today.
- ▶ And he asked whether we can do something about it.
- ▶ What do you think? Can we mix the **First Order Recipe**?

Areces & Blackburn: Logics for Computation

INRIA Nancy Grand Est

What do we do Today

- ▶ We'll cook First Order Logic à la ESSLLI08.
- ▶ We will see what we can reuse of what we already have...
- ▶ ...and extend the language if necessary.
- ▶ We will then show that the language we obtain is actually equivalent to the 'classical' First Order Language.

Areces & Blackburn: Logics for Computation

INRIA Nancy Grand Est

Checking our Stock

- ▶ We want to define for Patrick, a language equivalent to First Order Logic (with equality and constants, but without function symbols, he told me he doesn't need them).
- ▶ During the previous lectures we introduced a number of operators.
 - ▶ Propositional symbols ✓
 - ▶ The boolean operators \wedge, \neg ✓
 - ▶ The $\langle R \rangle$ operator ✓
 - ▶ Constants ✓
 - ▶ The $:$ operator ✓
 - ▶ The counting operators $\langle = n R \rangle$ ✗
 - ▶ The universal operator $[U]$ Close, but no cigar!
 - ▶ The reflexive and transitive closure operator $\langle R^* \rangle$ ✗
- ▶ Which one can we use?

Areces & Blackburn: Logics for Computation

INRIA Nancy Grand Est

The $[U]$ operator is not enough

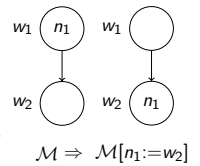
- ▶ Granted: **we need universal quantification**.
- ▶ But the $[U]$ operator is not expressive enough.
 - ▶ We won't prove it here (one way to do it, for example is noting that the language containing $[U]$ is still decidable, while full first order logic should be undecidable).
 - ▶ The universal operator is not fine grained enough:
 $[U]$ says **for all**
and we need **for all x**
- ▶ First order quantification gives as a delicate control (via variables) of **what we are quantifying on**.

Areces & Blackburn: Logics for Computation

INRIA Nancy Grand Est

A Detour: Renaming Points

- ▶ I will define a little piece of notation that I will need in the next slide.
- ▶ As I want it to be very clear, I'll do it here and give an example.
- ▶ Let
 - ▶ $\mathcal{M} = \langle D, \{R_i\}, \{P_i\}, \{N_i\} \rangle$ be a model,
 - ▶ w an element in D ($w \in D$),
 - ▶ and n_i a name.
- ▶ We write $\mathcal{M}[n_i := w]$ for the model obtained from \mathcal{M} where the only change is that now n_i is interpreted as w .



Areces & Blackburn: Logics for Computation

INRIA Nancy Grand Est

First Order Quantification

- ▶ We introduce the operator $\langle n \rangle$ where n is a **name** (we will call the operator **rename n**) as:
 $\mathcal{M}, w \models \langle n \rangle \varphi$ iff for some w' $\mathcal{M}[n := w']$, $w \models \varphi$
- ▶ Compare with
 $\mathcal{M}, w \models \langle R \rangle \varphi$ iff there is w' s.t. wRw' and $\mathcal{M}, w' \models \varphi$.
- ▶ Compare with $\langle U \rangle \varphi := \neg [U] \neg \varphi$
 $\mathcal{M}, w \models \langle U \rangle \varphi$ iff for sum w' , $\mathcal{M}, w' \models \varphi$
- ▶ Actually, using $\langle n \rangle$ and $:$ together we can **define** $[U]$:

$$[U]\varphi \text{ iff } \neg \langle n \rangle (n : \neg \varphi)$$

Areces & Blackburn: Logics for Computation

INRIA Nancy Grand Est

Capturing all First Order Logic

- ▶ We will now show that we can capture all First Order Logic using: the $\langle R \rangle$ language, names, $:$ and $\langle n \rangle$.
- ▶ We will (recursively) define a translation that will assign to each formula of the First Order Language, and equivalent formula in our language

$$\begin{aligned} Tr(s = t) &= s : t \\ Tr(P(s)) &= s : p \\ Tr(R(s, t)) &= s : \langle R \rangle t \\ Tr(\neg \varphi) &= \neg Tr(\varphi) \\ Tr(\varphi \wedge \psi) &= Tr(\varphi) \wedge Tr(\psi) \\ Tr(\exists s. \varphi) &= \langle s \rangle Tr(\varphi) \\ (Tr(\forall s. \varphi) &= \neg \langle s \rangle \neg Tr(\varphi) = [x] Tr(\varphi)) \end{aligned}$$

Areces & Blackburn: Logics for Computation

INRIA Nancy Grand Est

Examples

- Let's write down a couple of formulas in First Order Logic and translate them to our language. Again, let's use the convention $[X]$ for $\neg(X)\neg$

$$\begin{aligned} & Tr(\forall x.(Man(x) \rightarrow \exists y.(Woman(y) \wedge Loves(x, y)))) \\ & [x](Tr(Man(x) \rightarrow \exists y.(Woman(y) \wedge Loves(x, y)))) \\ & [x](Tr(Man(x)) \rightarrow Tr(\exists y.(Woman(y) \wedge Loves(x, y)))) \\ & [x](x:Man \rightarrow Tr(\exists y.(Woman(y) \wedge Loves(x, y)))) \\ & [x](x:Man \rightarrow \langle y \rangle (Tr((Woman(y) \wedge Loves(x, y)))) \\ & [x](x:Man \rightarrow \langle y \rangle (Tr(Woman(y)) \wedge Tr(Loves(x, y)))) \\ & [x](x:Man \rightarrow \langle y \rangle (y:Woman \wedge x:\langle Loves \rangle y)) \end{aligned}$$

The Other Translation

- Of course, we can do the translation in the other direction as well.
- We only need to realize that the semantic definition of all the operators we introduced can be defined in first-order logic.

$\mathcal{M}, w \models \langle R \rangle \varphi$ iff there is w' s.t. wRw' and $\mathcal{M}, w' \models \varphi$

$$Tr_w(\langle R \rangle \varphi) = \exists w'. (R(w, w') \wedge Tr_{w'}(\varphi))$$

- The w in Tr_w keeps track of where we are evaluating the formula in the model.

The Other Translation

- Let's see the details. Assume that we have a formula in the $\langle R \rangle$ language extended with constants, and the $:$ and $\langle n \rangle$ operators. We will (recursively) define an equivalent first order formula:

$$\begin{aligned} Tr_x(p) &= P(x) \\ Tr_x(n_i) &= (n_i = x) \\ Tr_x(\neg\varphi) &= \neg Tr_x(\varphi) \\ Tr_x(\varphi \wedge \psi) &= Tr_x(\varphi) \wedge Tr_x(\psi) \\ Tr_x(\langle R \rangle \varphi) &= \exists x.(R(x, y) \wedge Tr_y(\varphi)) \text{ for } y \text{ a new variable} \\ Tr_x(n:\varphi) &= Tr_n(\varphi) \\ Tr_x(\langle n \rangle \varphi) &= \exists n. Tr_x(\varphi) \end{aligned}$$

What we Covered Today

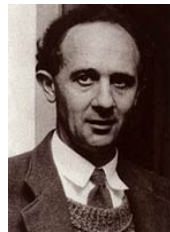
- In a way, the reason for today's talk was to show that there is nothing *special* about first order logic.
- It can be obtained in a natural way, following the ideas that we introduced in previous lectures.
- People has told me
I understand PL, but I would never get how FOL works.
NONSENSE!!
- As we saw today, they are not as different. You only need to look at them from the right perspective.
- If you really understand how one works, you already know how the other does.

Relevant Bibliography

- As Patrick mentioned in the first lecture one of the first *polytheistic logicians* was Arthur Prior.
- Prior is the father of Tense Logic, a logic that include the operators F and P to talk about the future and the past.
- He was a strong advocate of the *bottom up* way of viewing first-order logic that we presented today.



Prior, Arthur (1967). *Chapter V.6 of Past, Present and Future*. Clarendon Press, Oxford.



The Next Lecture

We Like it Complete and Compact
(and We have a Soft Spot for Löwenheim-Skolem)

Logics for Computation

Lecture #8: We like it Complete and Compact (and have a Soft Spot for Löwenheim-Skolem)

Carlos Areces and Patrick Blackburn
{carlos.areces,patrick.blackburn}@loria.fr

INRIA Nancy Grand Est
Nancy, France

ESSLLI 2008 - Hamburg - Germany

The story up till now

- ▶ We put together various bits-and pieces we have played with (names, $:$, diamonds) added the $\langle x \rangle$ and $[x]$ operators, and reached the most expressive language we have seen so far.
- ▶ The approach might seem somewhat shake-and-bake (or as the French would probably put it: "Bricolage!"), but we arrived at somewhere very important.
- ▶ For we reached "first-order logic", the logic often considered to "classical logic", and one of the most distinctive and important spots on the logical landscape. For our $\langle x \rangle$ and $[x]$ are really just the familiar \exists and \forall quantifiers.

Areces & Blackburn: Logics for Computation

INRIA Nancy Grand Est

Not called classical logic for nothing

- ▶ Some logicians (notably Quine) view first-order logic as the be-all and end-all of logic.
- ▶ We disagree with this viewpoint, but there is no getting away from a very stubborn fact — first-order logic certainly is *special*.
- ▶ Being "first-order logic" is not a matter of notation or symbolism. It's about something much deeper. It's about finally being able to get to grips with each and every element in the domains of our models.
- ▶ And as we shall learn today and tomorrow, this has some deep consequences. Today we're going to take a (rather abstract) look at inference and expressivity in first-order logic.

Areces & Blackburn: Logics for Computation

INRIA Nancy Grand Est

In today's lecture

- ▶ First we'll look at inference. We'll briefly discuss what we have to do to get a tableaux system for the quantifiers, and then discuss the concepts of *soundness* and *completeness*. We'll avoid computational issues — though we will observe that model checking, the simplest of our inference tasks, is starting to get computationally difficult.
- ▶ But then we turn to the heart of the lecture: expressivity. We have clearly gained a lot of expressivity, but interesting gaps remain. We will discuss two results, the *Compactness Theorem* and the *Löwenheim-Skolem theorem(s)* that pin down the crucial expressive limitations of first-order logic. Indeed, as we shall learn, these results actually *characterize* first-order logic.

Areces & Blackburn: Logics for Computation

INRIA Nancy Grand Est

To infinity and beyond... ?!



The distinction between finiteness and the different degrees of infinity will play an important role in the later part of this lecture. Will we get to infinity? Will we get beyond? Or is first-order logic really just a toy?

Areces & Blackburn: Logics for Computation

INRIA Nancy Grand Est

Example: Theory of linear order

We now have a lot of expressive power — enough to state some interesting theories. Here, for example, is a simple theory of linear order:

- ▶ Axiom Irr: $[x](x : \neg \langle R \rangle x)$
- ▶ Axiom Tran: $[x][y](x : \langle R \rangle \langle R \rangle y \rightarrow x : \langle R \rangle y)$
- ▶ Axiom Lin: $[x][y](x : y \vee x : \langle R \rangle y \vee y : \langle R \rangle x)$

So if we had a proof system for our first-order language, we could prove some non-trivial theorems:

$$\models \text{Irr} \wedge \text{Tran} \wedge \text{Lin} \rightarrow [x][y](x : \langle R \rangle y \rightarrow y : \neg \langle R \rangle x).$$

That is (recall Lecture 1) we would be able to handle Euclid-style reasoning.

Areces & Blackburn: Logics for Computation

INRIA Nancy Grand Est

Two Tableaux Rules

Here are the two key tableau rules we need:

$$\frac{s : \langle x \rangle \varphi}{s : \varphi[x \leftarrow n]} ((x)) \quad \frac{s : \neg \langle x \rangle \varphi}{s : \varphi[x \leftarrow o]} (\neg(x))$$

(where n is an *new* name) (where n is an *old* name)

Actually, we need other rules too — as yet we have given no rules for $:$ or for *names* — but we won't bother with these here.

Instead we ask again the question raised yesterday: how to we know that what we're doing is right? And what does "right" mean anyway?

Areces & Blackburn: Logics for Computation

INRIA Nancy Grand Est

Proof theory

- ▶ Proof theory is the *syntactic* approach to logic.
- ▶ It attempts to define collections of rules and/or axioms that enable us to generate new formulas from old. That is, it attempts to pin down the notion of inference syntactically.
- ▶ Given some proof system P , we write $\vdash_P \phi$ to indicate that a formula ϕ is provable in the the proof system. (Incidentally, $\not\vdash_P \phi$ means that ϕ is *not* provable in proof system P .)

Areces & Blackburn: Logics for Computation

INRIA Nancy Grand Est

Many types of proof system

- ▶ Natural deduction
- ▶ Hilbert-style system (often called axiomatic systems)
- ▶ Sequent calculus
- ▶ Tableaux systems
- ▶ Resolution

Why so many different proof systems?

- ▶ Well, one of the most important may simply be that logicians love to play with such systems — and every logician has his or her own favourite pet system!
- ▶ A more serious reason is: different proof systems are typically good for different purposes.
- ▶ In particular, some systems (notably tableau and resolution) are particularly suitable for computational purposes.

But what does all this have to do with semantics and inference?

- ▶ Note: **nothing** we have said so far about proof systems makes any connection with the model-theoretic ideas previously introduced.
- ▶ All we have done is talk about provability and vaguely said that we want to “generate” formulas syntactically. What does this have to do with relational structures and semantics?
- ▶ Answer: we insist on working with proof systems with two special properties, namely **soundness** and **completeness**.

Soundness

- ▶ Recall that we write $\models \phi$ to indicate that the formula ϕ is valid (that is, satisfied in all models under all assignments).
- ▶ Recall that we write $\vdash_P \phi$ to indicate that ϕ is provable in proof system P .
- ▶ We say that a proof system P is **sound** if and only if

$$\vdash_P \phi \text{ implies } \models \phi$$

Explanation

- ▶ That is, soundness means that syntactic provability implies semantic validity.
- ▶ To put it another way: P does not produce garbage.
- ▶ And another: P is “safe”.
- ▶ Needless to say, all the standard proof systems are sound.

Remark

- ▶ Soundness is typically an easy property to prove.
- ▶ Proofs typically have some kind of inductive structure. One shows that if the first part of proof is true in a model, then the rules only let us generate formulas that are also true in a model.

Completeness

- ▶ Recall that we write $\models \phi$ to indicate that the formula ϕ is valid (that is, satisfied in all models under all assignments).
- ▶ Recall that we write $\vdash_P \phi$ to indicate that ϕ is provable in proof system P .
- ▶ We say that a proof system P is **complete** if and only if

$$\models \phi \text{ implies } \vdash_P \phi$$

Explanation

- ▶ That is, completeness means that our proof system is strong enough to prove everything that is provable.
- ▶ To put it another way: **if some formula really is true in all models, then our proof system P really is powerful enough to generate it.**
- ▶ And another: no valid formula is out of reach of our proof system.
- ▶ The standard proof systems are complete.

Remark

- ▶ Completeness is a **much** deeper property than soundness, and is a lot more difficult to prove.
- ▶ It is typically proved by contraposition. **We show that if some formula is not provable ($\nvdash \phi$) then ϕ is not valid ($\not\models \phi$).** This is done by building a model for $\neg\phi$.
- ▶ And our first-order language is very expressive — so it can describe some pretty intricate models. And they certainly won't all be trees!

Soundness and completeness together

- ▶ Recall: proof system P is sound if and only if
$$\vdash_P \phi \text{ implies } \models \phi$$
- ▶ Proof system P is complete if and only if
$$\models \phi \text{ implies } \vdash_P \phi$$
- ▶ So if a proof system is both sound and complete (which is what we want) we have that:
$$\models \phi \text{ if and only if } \vdash_P \phi$$
- ▶ **That is, syntactic provability and semantic validity coincide.**

A remark on first-order model checking

- ▶ I've carefully avoided saying anything about computational issues — because there is a **lot** to say and that is the topic of tomorrow's talk! We've brought a lot of expressivity — and we're going to have to pay for it!
- ▶ But things are getting computationally complex. To see this, we remark that even the (usually easy) model checking task is getting tough.
- ▶ In fact the model checking task for first-order logic is PSPACE-complete! That is, it is as tough as validity/satisfiability checking for the diamond language!
- ▶ To see why, reflect on what we have to do to check a formula of the form $[x](y)[z](w) \dots [v](x)\phi$.

Expressivity of first-order logic

- ▶ We turn now to the theme of **expressivity of first-order logic**.
- ▶ Our discussion revolves around two famous results: the **Compactness Theorem**, and the **Löwenheim Skolem Theorem(s)**. I'm going to be state (but won't prove) these results and discuss some of their consequences.
- ▶ As we said at the start of the lecture, our discussion will have a lot to do with **finiteness**, and the **different grades of infinity**. So before going any further let's remind ourselves what an infinite set is ...

Finite and Infinite

- ▶ The most fundamental infinite set in set theory is the natural numbers \mathbf{N} . This is the set
$$\{0, 1, 2, 3, 4, 5, \dots\}$$
- ▶ A set S is **infinite** if there is one-to-one (injective) function from \mathbf{N} to S .
- ▶ A set is **finite** if it is not infinite. That is, a set is finite if it is not possible to define such a function. This amounts to saying that that it is the same size as some finite set
$$\{1, 2, 3, 4, 5, \dots, n\}$$

Such a set, incidentally, is called an **initial segment** of the natural numbers.

Infinite Axiom Sets!



- ▶ We already seen a simple set of axioms, namely the three axioms that defined the theory of linear order.
- ▶ But note: **any finite set of axioms can be replaced by one single axiom** — for we simply need to form their conjunction!
- ▶ So, very early in the history of logic, logicians started to consider the consequences of working with infinite sets of axioms. After all, why not? **Remember: Logicians are the Masters of the Universe!**
- ▶ Now, using infinite sets of axioms **does** give us more power ...

An infinite set of axioms defining infinity!

- ▶ Try to write down a **single sentence** of first-order logic that is true in every infinite model (and false on every finite one). But don't try too hard, for I promise that you won't find one...
- ▶ But it's easy to write down an **infinite** set of sentences with this property.
- ▶ Let $AtLeast_2$ be $\langle x \rangle \langle y \rangle (x : \neg y)$.
- ▶ Let $AtLeast_3$ be $\langle x \rangle \langle y \rangle \langle z \rangle (x : \neg y \wedge x : \neg z \wedge y : \neg z)$.
- ▶ Following this pattern, we can write down sentences $AtLeast_4$, $AtLeast_5$, $AtLeast_6$, $AtLeast_7$, ..., $AtLeast_{100}$, ...
- ▶ Let \mathbf{INF} be the set of all these sentences. Then we have that

$\mathcal{M} \models \mathbf{INF}$ iff \mathcal{M} is an infinite model.

Compactness Theorem

Compactness Theorem: Let Σ be an infinite set of first-order sentences. If every finite subset of Σ can be made true (in some model or other) then there is at least one model that simultaneously makes every sentence in Σ true.

We can paraphrase this as follows. To check whether an infinite set of first-order sentences Σ , has a model, we **don't** need to try and find a model that make all the sentences in Σ true at once. If it enough to show that any finite subset of Σ can be made true in some model. For if we can show this, the Compactness Theorem guarantees that there is some model that makes all the (infinitely many) sentences in Σ true all at once.

How do you prove the Compactness Theorem?

Many proofs are known, but two are worth mentioning...

- ▶ Actually, one can prove the Compactness Theorem for first-order logic more-or-less simultaneously with the Completeness Theorem. More precisely, completeness for first-order logic is a reasonably simple extension of compactness.
- ▶ And there is another (in a sense more revealing) proof. The ultraproduct construction lets us “multiply together” the finite models into one big model.

A powerful theorem

The Compactness Theorem is central to mathematical model theory:

- ▶ A powerful theorem — and a two-sided one.
- ▶ On the positive side, it allows us to build many interesting and unexpected model (such as non-standard models of arithmetic).
- ▶ And it has negative uses too — it can also show that we cannot define certain things. *And that's the kind use we will put it now.*

Finiteness is not first-order definable

The question we will ask is the following:

*Is there a **single sentence** of first-order logic that defines finiteness?*

That is, is there a single first order sentence (let's call it **fin** such that:

$$\mathcal{M} \models \mathbf{fin} \text{ iff } \mathcal{M} \text{ is finite}$$

As we shall see, the answer is **no**.

The argument

- ▶ Suppose there is such a sentence **fin**. We are going to show (with the help of the Compactness Theorem) that this assumption leads to a contradiction.
- ▶ Consider the set of sentences $\mathbf{INF} \cup \mathbf{fin}$
- ▶ That is: $\{\mathbf{AtLeast}_1, \mathbf{AtLeast}_2, \mathbf{AtLeast}_3, \mathbf{AtLeast}_4, \dots\} \cup \mathbf{fin}$
- ▶ **Claim:** every finite subset of this has a model. *Why is this?*

And now we have a problem

- ▶ Since every finite subset of $\mathbf{INF} \cup \mathbf{fin}$ is true, by the Compactness Theorem, there is some model \mathcal{M} that simultaneously makes every sentence in $\mathbf{INF} \cup \mathbf{fin}$
- ▶ Hence, as \mathcal{M} makes **INF** true, \mathcal{M} is infinite.
- ▶ But as \mathcal{M} also makes **fin** true, so \mathcal{M} is finite!
- ▶ *Oooooooooooooops!!!!!!!!!!!!*
- ▶ From this contradiction we deduce that **fin** does not exist. That is, there is no sentence of first-order logic that expresses the concept of finiteness.

Countably infinite and bigger...

- ▶ A set S is **countably infinite** if there is one-to-one (injective) and onto (surjective) function from \mathbf{N} to S .
- ▶ Countable infinite sets are the smallest infinite sets. Examples of countable infinite sets include: \mathbf{N} , \mathbf{E} (the even numbers), \mathbf{Z} (the integers), and \mathbf{Q} (the fractions).
- ▶ The set of real numbers \mathbf{R} is **not** countably infinite — it is bigger.
- ▶ In general, the power set $\mathcal{P}(S)$ of S is bigger than S .
- ▶ **The universe of sets is huge. It contains infinitely many ever-bigger grades of infinity.**

Löwenheim-Skolem Theorems

The Löwenheim Skolem Theorems tell us that first-order logic is completely blind to all these distinctions:

Upward Löwenheim-Skolem Theorem: *If a set of sentences Σ has an infinite model, it has infinite models of all larger infinite cardinalities.*

Downward Löwenheim-Skolem Theorem: *If a set of sentences Σ has an infinite model, it has infinite models of all lower infinite cardinalities. In particular, it always has a countable model.*

It follows that is not possible to fully describe the real numbers \mathbf{R} (that is, describe them up to isomorphism) using first-order logic. Any description will have **many** models (indeed models of every infinite cardinality) and a model that (being countable) is too small!

Lindström's Theorem

- ▶ There is a field of logic called **abstract model theory** which works with very general and abstract definitions of what logics are.
- ▶ There is a celebrated result from abstract model theorem called Lindström's Theorem which tells us that first-order logic is the **only** logic for which both the Compactness and the Downward Löwenheim-Skolem Theorems hold.
- ▶ That is, if you invent a very strange logical formalism, but can prove that it has these two properties, then you have invented a logic with exactly the same expressive power as first-order logic. You “new” logic, when you get right down to it, is in the business of quantifying over individuals.
- ▶ **And is there really anything else out there to quantify over....?**

What we Covered Today

- ▶ Brief remarks on inference for the $\langle x \rangle$ language: how to build a tableaux system, and why even model checking is starting to get hard (PSPACE-complete).
- ▶ Soundness and completeness and why they are important.
- ▶ Expressivity for the $\langle x \rangle$ language: Compactness and the Löwenheim Skolem Theorem.
- ▶ We also learned that “first-order logic” occupies a fundamental place in the expressivity hierarchy. Being “first-order” is not about this notation or that notation; it’s something more fundamental.
- ▶ But there remains a big gap in our discussion: computability.

Relevant Bibliography



Georg Cantor (1845-1918) created modern set theory, which is the setting for virtually all of modern mathematics. As David Hilbert, the famous mathematician and logician once remarked, “No one shall expel us from the Paradise that Cantor has created.”

Relevant Bibliography



The Completeness Theorem for first-order logic was proved by Kurt Gödel (1906–1978) in his 1928 doctoral thesis. As it turned out, however, this was merely the first of many great results that he was to prove.

Relevant Bibliography



The modern form of the Compactness Theorem seems to trace back to the work of the Russian mathematician Anatoly Maltsev (1909–1967).

Relevant Bibliography



The downward Löwenheim-Skolem Theorem was first proved by Leopold Löwenheim (1878–1957; top picture) in 1915. In 1920 his proof was greatly simplified and generalised by Thoralf Skolem (1887–1963).

The Next Lecture

Computing the Uncomputable

Logics for Computation

Lecture #9: Putting Tiles in an Infinite Bathroom

Carlos Areces and Patrick Blackburn
{carlos.areces,patrick.blackburn}@loria.fr

INRIA Nancy Grand Est
Nancy, France

ESSLLI 2008 - Hamburg - Germany

The Story so Far

- ▶ We are working with first-order logic.
- ▶ We discussed **soundness** and **completeness**
Intuitively, they are the way to **synchronize** the semantics of a logic with an inference method like tableaux.
They are the way to show that **we got it right**.
- ▶ We also discussed some properties like **Compactness** and **Löwenheim-Skolem**.
- ▶ We saw that these properties actually **characterize first order logic**: Lindström Theorem.
- ▶ We've come a long way. Think that on Monday we were in propositional logic!

Areces & Blackburn: Logics for Computation

INRIA Nancy Grand Est

What do we do Today

- ▶ We'll talk about the satisfiability problem of first-order logic.
- ▶ We will argue that the problem is **undecidable**
 - ▶ That is, there is no algorithm that can answer for any formula of first order logic, whether the formula has a model or not.
- ▶ Actually, we are going to show how to **tile and infinite bathroom**.

Areces & Blackburn: Logics for Computation

INRIA Nancy Grand Est

Tableaux for First Order Logic

- ▶ Patrick introduced these two rules yesterday:

$$\frac{s:\langle x \rangle \varphi}{s:\varphi[x \leftarrow n]} (\langle x \rangle) \quad \frac{s:\neg \langle x \rangle \varphi}{s:\neg \varphi[x \leftarrow o]} (\neg \langle x \rangle)$$

(where **n** is a **new name**) (where **o** is an **old name**)

- ▶ How comes that these tableaux do not terminate?
- ▶ As we already mentioned,
The SAT problem of FO is undecidable.
- ▶ Hence, we cannot expect any sound and complete tableaux to also terminate.

Areces & Blackburn: Logics for Computation

INRIA Nancy Grand Est

(Un)Decidability

How can we prove that problem X is **undecidable**?
One way is

- ▶ Ask somebody (more intelligent than us) to prove that some problem Y is undecidable
- ▶ Prove that if X would be decidable then Y would be decidable, giving a codification of Y into X.

The **halting problem** of Turing machines is the standard example of an undecidable problem. The behaviour of a Turing machine, and the predicate that says that a given Turing machine stops on all inputs can be expressed in FO.

Areces & Blackburn: Logics for Computation

INRIA Nancy Grand Est

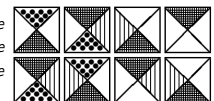
Tiling Problems

- ▶ A **tiling problem** is a kind of jigsaw puzzle
- ▶ a **tile** T is a 1×1 square, fixed in orientation, with a fixed **color** in each side
- ▶ for example, here we have six different kinds of tiles:



- ▶ a simple tiling problem, could be:

Is it possible to place tiles of the kind we show above on a grid of 2×4 , in such a way that we cover the entire grid and that adjacent tiles have the same color on neighboring sides?



Areces & Blackburn: Logics for Computation

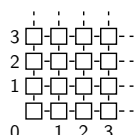
INRIA Nancy Grand Est

Tiling Problems

- ▶ The general form of a tiling problem
Given a finite number of kinds of tiles \mathcal{T} , can we cover a given part of $\mathbb{Z} \times \mathbb{Z}$ in such a way that adjacent tiles have the same color on the neighboring sides?
- ▶ In some cases, it is also possible to impose certain conditions on what is considered a correct tiling.

- ▶ Covering $\mathbb{N} \times \mathbb{N}$

- ▶ **tiling $\mathbb{N} \times \mathbb{N}$** : Given a finite set of tiles \mathcal{T} , can \mathcal{T} cover $\mathbb{N} \times \mathbb{N}$?
- ▶ this problem is undecidable (It is equivalent to the halting problem of Turing machines)

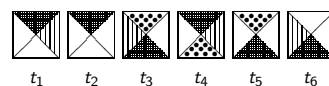


Areces & Blackburn: Logics for Computation

INRIA Nancy Grand Est

Representing Tiling Problems

- ▶ Notice that every finite set of tiles $\mathcal{T} = \{t_1, \dots, t_k\}$ can be represented as **two binary relations** H , and V .
We put $H(t_i, t_j)$ when the right side of t_i coincides with the left side of t_j , and similarly for V .
- ▶ For example, for



we will have

$$H = \{(t_1, t_3), (t_1, t_6), (t_2, t_4), (t_2, t_5), (t_2, t_1), \dots\}$$
$$V = \{(t_1, t_3), (t_1, t_5), (t_1, t_6), (t_2, t_3), (t_2, t_5), \dots\}$$

Areces & Blackburn: Logics for Computation

INRIA Nancy Grand Est

Coding a Tiling of the Grid in FO

- **Theorem:** The problem of deciding whether a given FO formula is satisfiable is undecidable.

Let \rightarrow and \uparrow be relations, and t_1, \dots, t_n be propositional symbols.

Let φ be the conjunction of the formulas:

1) Total: $[x](y)(x:(R)y) \text{ for } R \in \{\rightarrow, \uparrow\} [x](y)(x:(R)y)$ for $R \in \{\rightarrow, \uparrow\}$

2) Functional: $[x][y][z](x:(R)y \wedge x:(R)z \rightarrow y:z) \text{ for } R \in \{\rightarrow, \uparrow\} [x][y][z](x:(R)y \wedge x:(R)z \rightarrow y:z)$ for $R \in \{\rightarrow, \uparrow\}$

3) Commuting: $[x][y](x:(\rightarrow)(\uparrow)y \leftrightarrow x:(\uparrow)(\rightarrow)y) [x][y](x:(\rightarrow)(\uparrow)y \leftrightarrow x:(\uparrow)(\rightarrow)y)$

4) Tiled: $[x](x:t_1 \vee \dots \vee x:t_n) [x](x:t_1 \vee \dots \vee x:t_n),$

5) But not Twice: $[x](x:t_i \rightarrow x:\neg t_j) \text{ for } i \neq j [x](x:t_i \rightarrow x:\neg t_j) \text{ for } i \neq j,$

6) Horizontal Match: $[x][y]((x:t_i \wedge x:(\rightarrow)y) \rightarrow$

Relevant Bibliography I

Undecidable Problems

- Alonzo Church invented **lambda calculus** and propose it as a **model for computation**.
- He showed then that the problem of satisfiability of first-order logic could not be coded in the lambda calculus. Hence it was **uncomputable**.
- Here is a list of some of Church's doctoral students: A. Anderson, P. Andrews, M. Davis, L. Henkin, S. Kleene, M. Rabin, B. Rosser, D. Scott, R. Smullyan, and A. Turing.



Church, Alonzo (1956). *Introduction to Mathematical Logic*. The Princeton University Press.

Relevant Bibliography II

Undecidable Problems

- Alan Turing invented **Turing Machines** and **Computer Science**.
- He showed that the halting problem could not be decided by a Turing Machine.
- And that the behavior of a Turing Machine can easily be described in first-order logic, providing an alternative proof that the satisfiability problem of first-order logic is undecidable.



Turing, Alan (1936), *On Computable Numbers, with an Application to the Entscheidungsproblem*. Proceedings of the London Mathematical Society, Series 2, Vol.42, pp 230–265, http://www.thocp.net/biographies/papers/turing_oncomputablenumbers_1936.pdf

Logics for Computation

Lecture #10: Where do We Go from Here?

Carlos Areces and Patrick Blackburn
{carlos.areces,patrick.blackburn}@loria.fr

INRIA Nancy Grand Est
Nancy, France

ESSLLI 2008 - Hamburg - Germany

The Story up to Now

- ▶ In the last three lectures we have discussed a very strong logic namely **first-order logic** (developed using the **Arthur Prior style notation** $\langle x \rangle$ and $[x]$) from the perspective of inference, expressivity, and computation.
- ▶ As we have seen, it is deductively natural, highly expressive (albeit with some interesting limitations), and undecidable.
- ▶ The question now, of course, is where (if anywhere) do we go from here ...?
- ▶ The answer is — **higher-order logic**, and in particular, **second order logic**.

Areces & Blackburn: Logics for Computation

INRIA Nancy Grand Est

What's that?

- ▶ Well, what is that? Aren't we already quantifying over everything that there is in our models?
- ▶ The answer is **no**. There's a lot more sitting out there in our models, patiently waiting to be quantified.
- ▶ Sure, we're already quantifying over the individuals — but there are **higher-order** entities there too, such as **sets of individuals**, and **relations**.
- ▶ And these logics certainly **do** offer increased expressivity...

Areces & Blackburn: Logics for Computation

INRIA Nancy Grand Est

Transitive closure

- ▶ In Lecture 6 we met the concept of the reflexive transitive closure of a relation.
- ▶ There are two (equivalent) ways of defining reflexive transitive closure.
 - ▶ As the smallest reflexive and transitive relation S (on the domain D) containing an arbitrary relation R ; or
 - ▶ As the relation T on D defined by xTy iff there is a finite sequence of elements of D such that $x = d_0$ and

$$d_0 R' d_1, d_1 R' d_2, \dots, d_{n-1} R' d_n, \text{ and } d_n R'_y$$

where $dR'e$ means that dRe or $d = e$.

- ▶ Let's try defining this concept in our shiny new $\langle x \rangle [x]$ language ...

Areces & Blackburn: Logics for Computation

INRIA Nancy Grand Est

Let's try...

Let X and Y be binary relations. It's easy to insist that X is reflexive:

$$\text{Ref}(X) \stackrel{\text{def}}{=} [n](n : \langle X \rangle n).$$

And it's easy to say that X is transitive:

$$\text{Tran}(X) \stackrel{\text{def}}{=} [n](n : \langle X \rangle \langle X \rangle n \rightarrow \langle X \rangle n)$$

And to say that X is a subrelation of Y

$$X \subseteq Y \stackrel{\text{def}}{=} [n](n : \langle X \rangle n \rightarrow \langle Y \rangle n).$$

Areces & Blackburn: Logics for Computation

INRIA Nancy Grand Est

Here's a first try...

So let's put all this together to define transitive reflexive closure:

$$\begin{aligned} \text{Tran}^*(R, S) &\stackrel{\text{def}}{=} \text{Ref}(S) \\ &\quad \wedge \text{Tran}(S) \\ &\quad \wedge R \subseteq S \\ &\quad \wedge S \text{ is the smallest such subrelation of } R \end{aligned}$$

Oh dear...!

Areces & Blackburn: Logics for Computation

INRIA Nancy Grand Est

No way José!

- ▶ Try as you might, you won't be able to do this
- ▶ And we can prove this using the Compactness Theorem
- ▶ $\{\neg p, [R]\neg p, [R][R]\neg p, [R][R][R]\neg p, \dots, (R^*)p\}$
- ▶ Every finite subset has a model. Hence (by Compactness) so does the whole thing. But this is impossible.
- ▶ Hence we can define R^* .

Areces & Blackburn: Logics for Computation

INRIA Nancy Grand Est

So extend the language

- ▶ As we learned in Lecture 6, we're free to extend the language.
- ▶ Now of course, we could just add the $\langle R^* \rangle$ operator — but that was just one example of something we couldn't do.
- ▶ Let's give ourselves the power to quantify over two types of higher order entities: **properties** and **binary relations**.

Areces & Blackburn: Logics for Computation

INRIA Nancy Grand Est

A second order language

- ▶ $\langle p \rangle \varphi$, and $[p] \varphi$ express existential and universal quantification over properties.
- ▶ $\langle R \rangle \varphi$, and $[R] \varphi$ express existential and universal quantification over relations.
- ▶ Semantics? Simply extend what we did in first-order case.

Now we can define reflexive transitive closure. . .

$$\begin{aligned} Tran^*(R, S) \quad =_{def} \quad & Ref(S) \\ & \wedge Tran(S) \\ & \wedge R \subseteq S \\ & \wedge [X](Ref(X) \wedge Tran(X) \wedge R \subseteq X \rightarrow S \subseteq X). \end{aligned}$$

What's the Price

- ▶ Loss of Completeness (for standard models)
- ▶ Loss of Compactness. After all:
 $\{\neg p, [R]\neg p, [R][R]\neg p, [R][R][R]\neg p, \dots, \langle R^* \rangle p\}$
is now an example of a set in which each finite subset has a model, and the complete set doesn't.
- ▶ Loss of Löwenheim Skolem. (It is easy to define the natural number \mathbf{N} and the integers \mathbf{Z} up to isomorphism.)

Tradeoff: expressivity versus computation and inference

- ▶ Which brings us back to the fundamental trade-off, expressivity versus inference/tractability.
- ▶ We've bought serious expressivity — and have lost everything else.

What we covered in the course

- ▶ We've been essentially looking at a menu of logics.
- ▶ But the menu was designed by a Master Chef (Tarski!); the meal is built around the crucial ingredient of relational structures.
- ▶ Relational structures tell us why logic is applicable in semantics (natural language metaphysics) and computer science.
- ▶ Back to a logicist position, but not in traditional sense.
- ▶ Monotheist — but not in terms of logic, rather, in terms of semantics.

Relevant Bibliography

And, hanging over it all, the brooding specter of Rudolf Carnap and Hans Reichenbach, the Vienna Circle of Philosophy and the rise of symbolic logic. A muddy world, in which he did not care to involve himself. From: *Galactic Pot-Healer*, by Philip K. Dick, 1969.

