Computational Modal Logics

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

INRIA Nancy Grand Est
Nancy, France

2009 - Copenhagen - Denmark

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

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

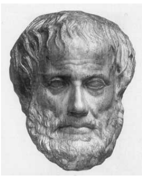
$$\forall x(\text{Human}(x) \rightarrow \text{Mortal}(x))$$

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. For example, his famous 'Buffy' syllogism:



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 with 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.

Logics 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.

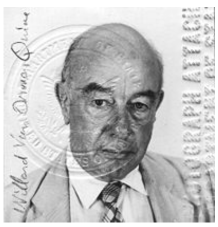
Logic or Logics?

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 thing 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 investigates 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 or Arthur Prior).

Polytheism in the 21st century

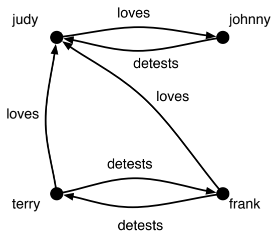
- But polytheism gradually became the dominant thread as time went by.
- Why? Because logic spread everywhere. Computer scientists used it. Early artificial intelligence relied on it. It cropped up in economic and cognitive science. And it became a corner stone of natural language semantics.
- However 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 our 'world') it is natural to play with different ways of talking about it.

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 though 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:

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

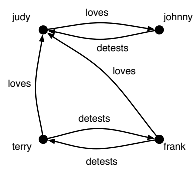
Sometimes we work with simpler forms. For example the following

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

we would usually write as:

$$\langle D, R \rangle$$

Another look at our first relational structure



$$\langle D, \{\text{loves}, \text{detests}\}, \emptyset, C \rangle$$

$D = \{ju, jo, te, fr\}$

$\text{loves} = \{(ju, jo), (te, ju), (fr, ju)\}$

$\text{detests} = \{(jo, ju), (te, fr), (fr, te)\}$

$C = \{judy \mapsto ju, johnny \mapsto jo, terry \mapsto te, frank \mapsto fr\}$

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.

A general and important modelling tool

- All common mathematical structures can be though 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 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: can computers help with such and such logic? If so how, and how much?

Inference tasks

The semantic perspective give us a good way to think about inference tasks. We will mention the following:

- Model Checking: 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 formulas true (or satisfied) in the model?
- Satisfiability Checking: Given a certain logical description φ, does there exist a model where the description is satisfied?
- Model Building: Given a certain logical description φ, exhibit a model where that description is true.
- Validity Checking: Given a logical description φ, is it true (or satisfied) in all models?

The Model Checking task

- As we shall learn, this is the simplest.
- 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 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 and Model Building

- A nice way to think of these problems is in terms of constraints. We have some description, and we ask ourselves 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.
- Why? After all, we've already mentions that $p \vee \neg p$ and $p \rightarrow p$ are not going to set too many pulses racing...
- The answer is: checking if φ is valid is boring. But checking if φ follows from Γ ($\Gamma \models \varphi$) is usually interesting. E.g.:

$$\{(p \vee q)\} \models p \quad ??? \quad \textit{No}$$
$$\{(p \vee q), \neg q\} \models p \quad ??? \quad \textit{Yes}$$
$$((p \vee q) \wedge \neg q) \rightarrow p \text{ is valid}$$

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 too 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 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.

Propositional Logic: Syntax

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

Propositional symbols:	$p, q, r, p_1, p_2, p_3, \ldots$
Logical symbols:	$\top, \bot, \neg, \vee, \wedge, \rightarrow, \leftrightarrow$
Grouping symbols:	$($, $)$

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

Propositional Logic: Syntax

We then say that $\top$, $\bot$ 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 $\neg\psi$, $(\varphi \vee \psi)$, $(\varphi \wedge \psi)$, $(\varphi \rightarrow \psi)$, and $(\varphi \leftrightarrow \psi)$.
- 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 $(p \wedge q) \rightarrow r$.

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.
- 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.

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

Truth Tables

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

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

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

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.

- $\bot$ is $\neg\top$
- $\top$ is $p \vee \neg p$.
- $\varphi \to \psi$ 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 $\{\to, \bot\}$ 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 already mentioned:

- If a model V makes a formula φ true we say V satisfies φ, or φ is true in V, 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 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

$$\begin{array}{rcl@{\qquad}rcl}
P & \equiv & \text{invite Peru} & \neg P & \equiv & \text{exclude Peru} \\
Q & \equiv & \text{invite Qatar} & \neg Q & \equiv & \text{exclude Qatar} \\
R & \equiv & \text{invite Romania} & \neg R & \equiv & \text{exclude Romania}
\end{array}$$

- The problem can be formalized as

$$\begin{array}{rcl}
\text{prince: } P \vee \neg Q & \equiv & \text{invite Peru or exclude Qatar (or both)} \\
\text{queen: } Q \vee R & \equiv & \text{invite Qatar or Romania (or both)} \\
\text{king: } \neg R \vee \neg P & \equiv & \text{snub Romania or Peru (or both)}
\end{array}$$

- 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?

$$\frac{\text{prince: } P \vee \neg Q \qquad \text{queen: } Q \vee R}{P \vee R}$$

- 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	T	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 ...

Expressivity

- As we remarked earlier, it is possible to think about the semantics of PL in terms of relational structures.
- 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 formula.
- Can we do better that $O(2^n)$ steps? Can other methods do satisfiability/validity checking more efficiently.
- Sadly, it seems the answer is no.

Computational Modal Logics

Carlos Areces   Patrick Blackburn

{carlos.areces,patrick.blackburn}@loria.fr

INRIA Nancy Grand Est
Nancy, France

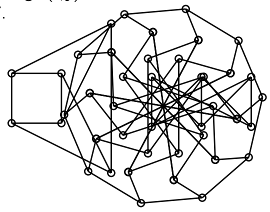
2009 - Copenhagen - Denmark

More than Diplomacy

- We saw 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!

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 assing colors to nodes in N s.t.
  - All nodes are colored with one of the k colors.
  - For every edge $(i, j) \in N$, the color of i is different from the color of j.
- An Example:



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: $p_{i1} \vee \ldots \vee p_{ik}$, for $1 \leq i \leq n$
2. Each node has no more than one color: $\neg p_{il} \vee \neg p_{im}$, 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 $1 \leq l \leq k$

Graph Coloring: Complexity

- Using the encoding in the previos page we efectively 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-deterministiec polynomial problems).
  - Assuming that, we just proved that PL-SAT is also NP-complete.

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 formula. 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 PL

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

Rules for $\neg$ and $\wedge$

$$\frac{(\varphi \wedge \psi)}{\begin{array}{c}\varphi\\ \psi\end{array}}\ (\wedge)$$

$$\frac{\neg(\varphi \wedge \psi)}{\neg\varphi \qquad \neg\psi}\ (\neg\wedge)$$

$$\frac{\neg\neg\varphi}{\varphi}\ (\neg\neg)$$

$$\begin{array}{c}
(\neg(p \wedge q) \wedge \neg\neg r) \wedge p\\
\neg(p \wedge q) \wedge \neg\neg r\\
p\\
\neg(p \wedge q)\\
\neg\neg r\\
r\\
\begin{array}{cc}\neg p \qquad\qquad & \qquad\qquad \neg q\\ \textit{Contradiction!!!} \qquad\qquad & \qquad\qquad \textit{Model}\end{array}
\end{array}$$

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 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 $\rightarrow$

$$\frac{\varphi \rightarrow \psi}{\neg\varphi \quad\quad \psi}\ (\rightarrow)$$

$$\frac{\neg(\varphi \rightarrow \psi)}{\begin{array}{c}\varphi\\ \neg\psi\end{array}}\ (\neg\rightarrow)$$

$$\begin{array}{c} \neg((p \wedge q) \rightarrow p) \\ p \wedge q \\ \neg p \\ p \\ q \\ \textit{Contradiction!!!} \end{array}$$

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.

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

Moving into Clausal Form

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

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

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

- Using the following equivalences:

$$\begin{array}{rcl} (\neg(p \vee q)) & \leadsto & (\neg p \wedge \neg q) \\ (\neg(p \wedge q)) & \leadsto & (\neg p \vee \neg q) \\ (\neg\neg p) & \leadsto & p \\ (p \vee (q \wedge r)) & \leadsto & ((p \vee q) \wedge (p \vee r)) \end{array}$$

The clause set associated to

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

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:

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

Conjunctive Normal Form

- This conversion to CNF can lead to exponentially big formulas. Consider

$$(p_1 \wedge q_1) \vee (p_2 \wedge q_2) \vee \cdots \vee (p_n \wedge q_n).$$

- In CNF we get a formula:

$$(p_1 \vee \cdots \vee p_{n-1} \vee p_n) \wedge (p_1 \vee \cdots \vee p_{n-1} \vee q_n) \wedge \cdots \wedge (q_1 \vee \cdots \vee q_{n-1} \vee q_n).$$

- Which has 2^n clauses: each clause contains either p_i or q_i.
- We can obtain formulas en CNF which are only polynomially bigger than the original formula. But they are only equisatisfiable to the input and not equivalent.

CNF: Using New Propositional Symbols

- Consider again

$$\varphi = (p_1 \wedge q_1) \vee (p_2 \wedge q_2) \vee \cdots \vee (p_n \wedge q_n).$$

- We can write φ' as:

$$(r_1 \vee \cdots \vee r_n) \wedge (\neg r_1 \vee p_1) \wedge (\neg r_1 \vee q_1) \wedge \cdots \wedge (\neg r_n \vee p_n) \wedge (\neg r_n \vee q_n).$$

- A model satisfies φ' if at least one of the new variables r_i is true. If r_i is true, then p_i and q_i are true: Every model that satisfies the translation also satisfies φ.
- On the other hand, if we have a model for φ then it makes some p_i and q_i true. We can get a model for φ' by seting r_i true.

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(Σ)
if Σ={} then return SAT                // (SAT)
if {} ∈ Σ then return UNSAT            // (UNSAT)
if Σ has unit clause {l}
   then DP(Σ[{l=true}])                // (Unit Pr.)
Choose literal l and
   if DP(Σ[{l=true}]) return SAT
      then return SAT
      else return DP(Σ[{l=false}])     // (Split)
```


Examples

$$\neg(\neg(p \vee q) \vee (\neg\neg q \vee (p \vee q))) \ {-}\text{CNF}{\rightarrow}\ \{\{p, q\}, \{\neg q\}, \{\neg p\}\}$$

$$\{\{P, \neg Q\}, \{Q, R\}, \{\neg R, \neg P\}\}$$

Davis-Putnam: The (Split) Rule

- The (`Split`) rule is non-deterministic: Which literal do we chose?
  - MOM's heuristics: pick the literal that occurs most often in the minimal size clauses (with ties broken at random or following a fix order). This method is hard to beat for speed and simplicity.
  - Jeroslow-Wang's heuristics: estimate the contribution each literal is likely to make to satisfying the clause set and pick the best

$$\mathrm{score}(l) = \sum_{c \in \Sigma \ \& \ l \in c} 2^{-|c|}$$

- SATZ, one of the best available implementations of DP, uses a heuristics aimed at maximizing unit propagation: generate candidate set of branching literals, perform unit propagation, choose the literal leading to the smallest simplified clause set

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,

$$\begin{aligned} 2^{100} &= 1.267.650.000.000.000.000.000.000.000.000 \\ 1.696^{100} &= 87.616.270.000.000.000.000.000 \\ 1.618^{100} &= 790.408.700.000.000.000.000 \end{aligned}$$

- DP can reliably solve problems with up to 500 variables
- Sadly real world applications easily go into the thousands of variables (remember coloring: #nodes × #colors).
- But this 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 specialized coloring algorithms.

What we covered up to now

- 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 eficient decision methods for PL (able to cope with problems with hundreds of propositional symbols, but our codings easily get into the thousands).
- Still, no matter how nicely we paint them, 1-point relational structures are boooooooooring.

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

- The life of "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
- *An introduction to good old fashioned model theory*, by Harold Simmons, `http://www.cs.man.ac.uk/~hsimmons/BOOKS/books.html`

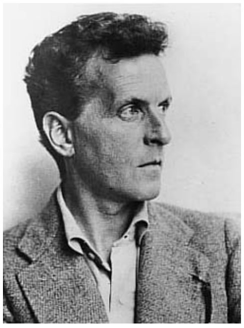
Relevant Bibliography

- 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 since the Aristotelian syllogistic.

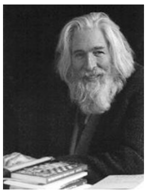
1815–1864

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.



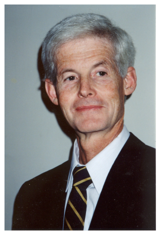
Relevant Bibliography



- Tableau were originally introduced by the Dutch logician Willem Beth.
- The particular 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.

Relevant Bibliography

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

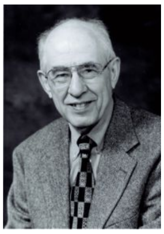


- That is, any problem in NP can be reduced in polynomial time to PL SAT.
- In plain English: if we find a cheap way of solving PL SAT, we'll also have a cheap way of solving a hell of a lot more. (Coda: probably there is no cheap way. Too good to be true. But still, it has not been shown. P $\stackrel{?}{=}$ 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

The Davis Putnam Algorithm



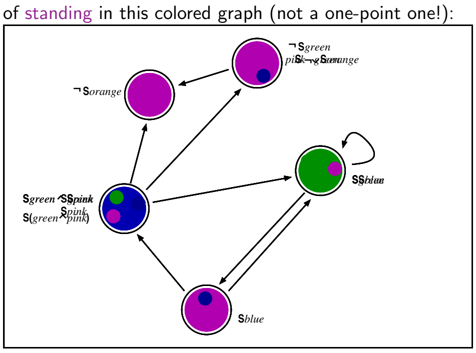
- It was developed by Martin Davis and Hilary Putnam.
- It was then improved (with the split rule) by Martin Davis, George Logemann and Donald Loveland. The correct name is DPLL.
- Davis' Web page: `http://www.cs.nyu.edu/cs/faculty/davism/`

Davis, Martin; Putnam, Hillary (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.

Simple Structures / Simple Languages

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



Computational Modal Logics

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

INRIA Nancy Grand Est
Nancy, France

2009 - Copenhagen - Denmark

The Story So Far

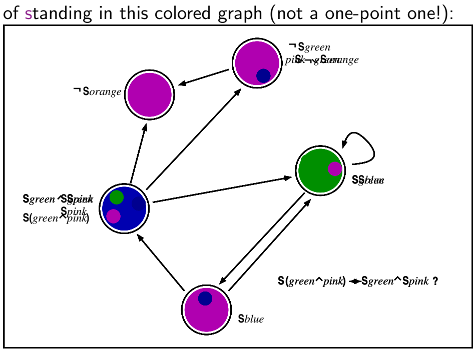
- We looked at SAT-solving (that is, model building) for propositional calculus in some detail.
- In particular, we discussed the Davis-Putnam algorithm.
- We also briefly met the concept of an NP-complete problem.
- In a nutshell, what we learned was this: although PL looks very simple, it is actually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

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 ...

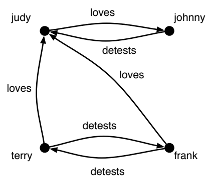
Simple Structures / Simple Languages

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



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.

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:

$$\langle R \rangle \qquad [R]$$

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

$$\Diamond \qquad \Box$$

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

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:

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

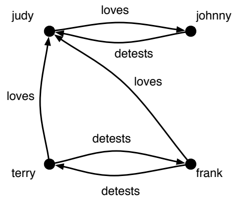
We don't need both boxes and diamonds

$\Box\varphi$ is $\neg\Diamond\neg\varphi$.

$\Diamond\varphi$ is $\neg\Box\neg\varphi$.

So, like WallE, we can choose either the diamond or the box — but we choose the diamond!

Waterloo Sunset ...

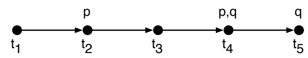


$$\langle \text{LOVES} \rangle \top \wedge \langle \text{DETESTS} \rangle \langle \text{LOVES} \rangle \top$$

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.



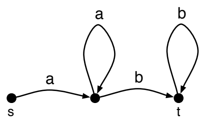
- $\Diamond q$ is true at t_1, t_2, t_3 and t_4.
- $\Diamond(p \wedge q)$ is satisfied at points t_1, t_2 and t_3 (because all these points are to the left of t_4 where both p and q are true together) but it is not satisfied at t_4 and t_5.
- Note that for any formula φ whatsoever, $\Box\varphi$ is satisfied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

Example

All formulas of the form

$$\langle a \rangle \cdots \langle a \rangle \langle b \rangle \cdots \langle b \rangle t$$

(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:

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

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
$$con(\varphi) \times nodes(\mathcal{M}) \times nodes(\mathcal{M}),$$
where $con(\varphi)$ is the number of connectives in φ, and $nodes(\mathcal{M})$ is the number of nodes in $\mathcal{M}$.
- To see this, note that $con(\varphi)$ tells us how many rounds of labeling we need to perform, one of the $nodes(\mathcal{M})$ 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

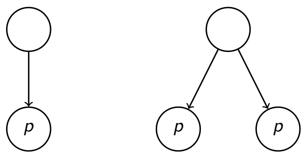
- Although this looks simple, this kind of model checking is important.
- It is possible to strengthen the diamond language in various ways to make it excellent for talking about graph structures representing hardware chips and much else besides.
- The above algorithm can (and have been) extended to 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.

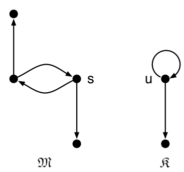
Bisimulations (formal definition)

Here is th 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 Rwv, 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 Rwv.

Bisimilar or not?

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

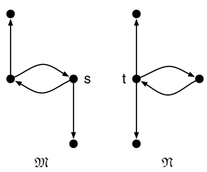


If they are bisimilar, what is the bisimulation? If they are not bisimilar, what is a formula that distinguishes them?

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?

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

Computational Modal Logics

Carlos Areces   Patrick Blackburn

Stand-up Logicians

{carlos.areces,patrick.blackburn}@loria.fr

INRIA Nancy Grand Est

Nancy, France

2009 - Copenhagen - Denmark

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?

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:

$$\frac{s{:}(\varphi \wedge \psi)}{\begin{array}{c} s{:}\varphi \\ s{:}\psi \end{array}}\ (\wedge) \qquad \frac{s{:}\neg(\varphi \wedge \psi)}{s{:}\neg\varphi \quad \mid \quad s{:}\neg\psi}\ (\neg\wedge) \qquad \frac{s{:}\neg\neg\varphi}{s{:}\varphi}\ (\neg\neg)$$

- Pretty neat: 3 rules for an NP-complete problem!
- But now we want to deal with more than a single point.
- The solution is: labels!
- They will help us keep track of what is going on in each point in our model.

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

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

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

$$\frac{s{:}\langle R \rangle \varphi}{\begin{array}{c} sRt \\ t{:}\varphi \end{array}}\ (\langle R \rangle)$$

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.

$$\frac{\begin{array}{c} s{:}\neg\langle R \rangle \varphi \\ sRt \end{array}}{t{:}\neg\varphi}\ (\neg\langle R \rangle)$$

The Complete Cast, plus an Example

$$\frac{s{:}(\varphi \wedge \psi)}{\begin{array}{c} s{:}\varphi \\ s{:}\psi \end{array}} \qquad \frac{s{:}\langle R \rangle \varphi}{\begin{array}{c} sRt \\ t{:}\varphi \end{array}} \text{ for } t \text{ a new label}$$

$$\frac{s{:}\neg(\varphi \wedge \psi)}{s{:}\neg\varphi \quad \mid \quad s{:}\neg\psi} \qquad \frac{\begin{array}{c} s{:}\neg\langle R \rangle \varphi \\ sRt \end{array}}{t{:}\neg\varphi}$$

$$\frac{s{:}\neg\neg\varphi}{s{:}\varphi}$$

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

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?

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?

$$\frac{s{:}(\varphi \wedge \psi)}{\begin{array}{c} s{:}\varphi \\ s{:}\psi \end{array}} \qquad \frac{s{:}\langle R \rangle \varphi}{\begin{array}{c} sRt \\ t{:}\varphi \end{array}} \text{ for } t \text{ a new label}$$

$$\frac{s{:}\neg(\varphi \wedge \psi)}{s{:}\neg\varphi \quad \mid \quad s{:}\neg\psi} \qquad \frac{\begin{array}{c} s{:}\neg\langle R \rangle \varphi \\ sRt \end{array}}{t{:}\neg\varphi}$$

$$\frac{s{:}\neg\neg\varphi}{s{:}\varphi}$$

- Did we get it right?
  - Did we get it right in the PL case, to start with?! Consider the rule:

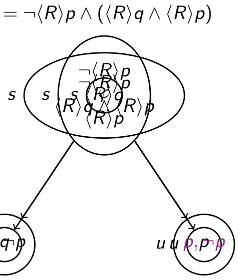
$$\frac{s{:}\neg(\varphi \wedge \psi)}{s{:}\neg\varphi \quad \mid \quad \begin{array}{c} s{:}\varphi \\ s{:}\neg\psi \end{array}}$$

- What can we learn from the calculus?
  - Something about models!

Tree Models

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

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



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

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.

$$\frac{s{:}(\varphi \wedge \psi)}{\begin{array}{c} s{:}\varphi \\ s{:}\psi \end{array}} \qquad \frac{s{:}\langle R \rangle \varphi}{\begin{array}{c} sRt \\ t{:}\varphi \end{array}} \text{ for } t \text{ a new label}$$

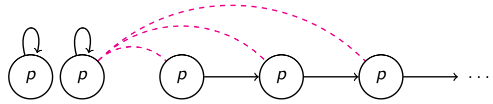
$$\frac{s{:}\neg(\varphi \wedge \psi)}{s{:}\neg\varphi \quad \mid \quad s{:}\neg\psi} \qquad \frac{\begin{array}{c} s{:}\neg\langle R \rangle \varphi \\ sRt \end{array}}{t{:}\neg\varphi}$$

$$\frac{s{:}\neg\neg\varphi}{s{:}\varphi}$$

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

One is the Same as Infinite

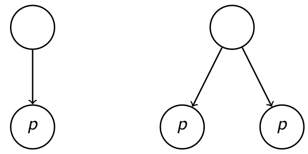
- We said that we cannot say infinite in the $\langle R \rangle$ language.
- Let's see this in more detail. Consider the model:



- 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.

One is the Same as Two

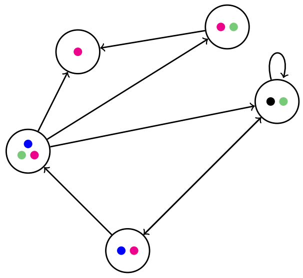
- But let's consider a simpler example (after all, infinite is quite a big number).
- The $\langle R \rangle$ language cannot distinguish between one and two!!!



Learning to Count

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

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



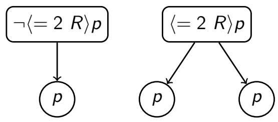
Alice in Wonderland

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.

Extending the Language

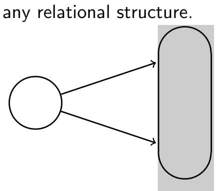
- What other things we cannot say in the $\langle R \rangle$ language?
- Plenty:
  1. In that particular node.
  2. Everywhere in the model.
  3. In a finite number of steps.

  ...
- Luckily, as Humpty Dumpty says, we are the masters, and we can desing the language that better pleases us.
- Let's get to work...

Names for Points

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

$\langle R \rangle$(pink $\wedge$ n)

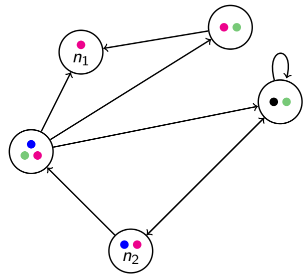


- Looks useful...
- We can introduce names into our language (you probably know them as constants).

Names for Points

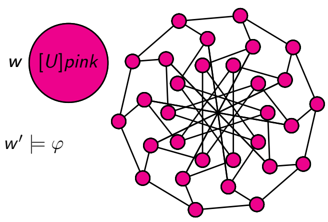
- When we allow names in our language, our models will look like this:
- We have already used something like names. Anybody remembers when?
- Tableaux for the $\langle R \rangle$ language!
- If we also introduce the :-operator we can write things like

n_1:pink
n_2:$\langle R \rangle$pink
n_2:$\langle R \rangle\langle R \rangle\langle R \rangle n_2$



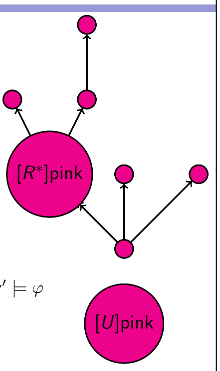
Everywhere in the model

- 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 $\mathcal{M}, w \models [U]$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:
    $p \vee \langle R \rangle p \vee \langle R \rangle \langle R \rangle p \vee \ldots$
- 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)



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 discusing 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 hight complexity (EXPTIME-complete).
- By chosing the right expressivity for a given application we will pay the exact price required.

Checking our Stock

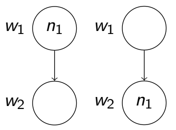
- Suppose we want to define a modal language equivalent to First Order Logic
  (with equality and constants, but without function symbols).
- During the previous lectures we introduced a number of operators.
  - Propositional symbols $\surd$
  - The boolean operators $\wedge$, $\neg$ $\surd$
  - The $\langle R \rangle$ operator $\surd$
  - Constants $\surd$
  - The : operator $\surd$
  - The counting operators $\langle = n\ R \rangle$ $\times$
  - The universal operator $[U]$ Close, but no cigar!
  - The reflexive and transitive closure operator $\langle R^* \rangle$ $\times$
- Which one can we use?

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.

A Detour: Renaming Points

- I will define a litle 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.



$\mathcal{M} \Rightarrow \mathcal{M}[n_1{:=}w_2]$

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 some $w', \mathcal{M}, w' \models \varphi$
- Actually, using $\langle n \rangle$ and : together we can define $[U]$:

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

To infinity and beyond...?!

Just how expressive is the language we just defined?

- Does it have the tree model property?
- Does it have the finite model property?

Is it really a MACHO language, or is it really just a toy? Let's see...

Example: Coding Infinity

This language hast tons of expressive power:

- Irr: $[x](x : \neg \langle R \rangle x)$
- Tran: $[x][y](x : \langle R \rangle \langle R \rangle y \rightarrow x : \langle R \rangle y)$
- Ser: $[x]\langle y \rangle (x : \langle R \rangle y)$

Theorem: If $M \models$ Irr $\wedge$ Tran $\wedge$ Ser then M is infinite.

Daddy, Daddy, It's broken!!! ☹

If the language can say 'infinite,' it means that we won't be able to know when to stop when searching for models for a formula.

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

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

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 $\neg\langle X\rangle\neg$

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

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.

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

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

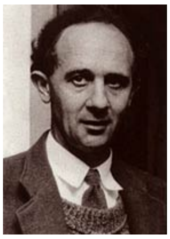
- The w in Tr_w keeps track of where we are evaluating the formula in the model.

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 the : and $\langle n\rangle$ operators.
  We will (recursively) define an equivalent first order formula:

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

Relevant Bibliography



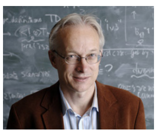
- The first polytheistic logicians was Arthur Prior.
- Prior is the father of Tense Logic, a logic that include the operators F and P to talk about the future and the past.
- He was a strong advocate of the bottom up way of viewing first-order logic that we presented today.

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

Relevant Bibliography

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



- Unraveling, the procedure to turn arbitrary models into trees, was introduced by Segerberg.
- Segerberg's Web page: `http://www.phil.ucalgary.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.

Relevant Bibliography

- 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 the languages we discussed today have been extensively studied by you-know-who.
  Patrick'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, Unversity of Amsterdam.