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

ESSLLI 2008 - Hamburg - Germany

The Story So Far

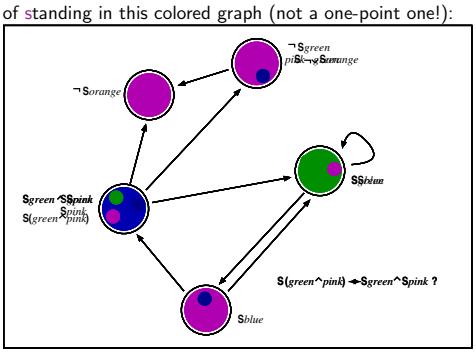
- We looked at SAT-solving (that is, model building) for propositional calculus in some detail.
- In particular, we discussed the Davis-Putnam algorithm.
- We also briefly met the concept of an NP-complete problem.
- In a nutshell, what we learned was this: although PL looks very simple, it is actually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

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 . . .

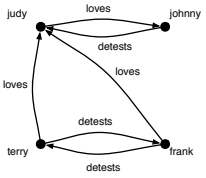
Simple Structures / Simple Languages

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:

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

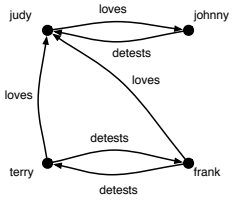
We don't need both boxes and diamonds

$\Box\varphi$ 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 . . .

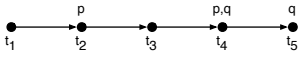


$\langle \text{LOVES} \rangle \top \wedge \langle \text{DETESTS} \rangle \langle \text{LOVES} \rangle \top$

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.



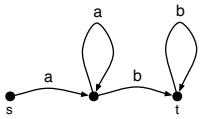
- $\Diamond q$ is true at t_1, t_2, t_3 and t_4.
- $\Diamond(p \wedge q)$ is satisfied at points t_1, t_2 and t_3 (because all these points are to the left of t_4 where both p and q are true together) but it is not satisfied at t_4 and t_5.
- Note that for any formula φ whatsoever, $\Box\varphi$ is satisfied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

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

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

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

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

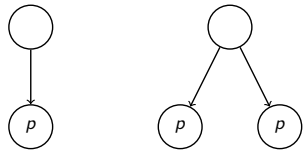
- Although this looks simple, this kind of model checking is important.
- It is possible to strengthen the diamond language in various ways to make it excellent for talking about graph structures representing hardware chips and much else besides.
- The above algorithm can (and have been) extended to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

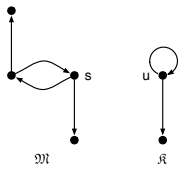
Atomic harmony: w and w' satisfy the same proposition symbols,

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)?

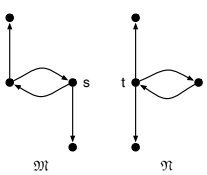


If they are bisimilar, what is the bisimulation? If they are not bisimilar, what is a formula that distinguishes them?

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.

What we Covered Today

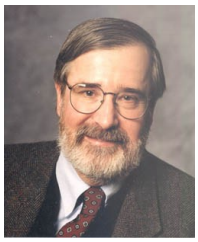
- We motivated the diamond language, defined its syntax and semantics, and gave example of the concepts of satisfaction at a point in a model, and of validity.
- We showed that model checking with diamonds is a simple task.
- We discussed expressivity, introducing and illustrating the concept of bisimulation.
- 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.

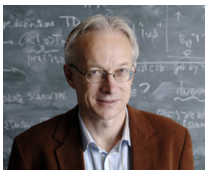
Relevant Bibliography II

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

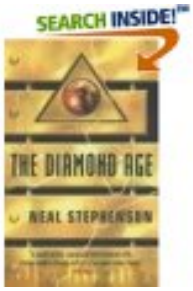
He wears a crown of diamonds — Johan van Benthem, the king of bisimulation. The researcher who, more than any other, made people see how general and useful diamond languages and their relatives actually are, and who is largely responsible for creating the LLI community.

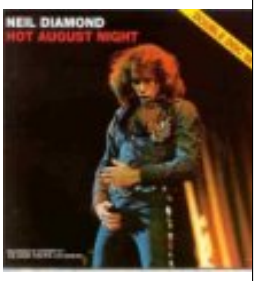


Relevant Bibliography

- 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