Battle of the Logics

Barteld Kooi and Rineke Verbrugge

A concern of the organizers of the workshop on 'Games, Action and Social Software' is that although everyone is keen to use logic for the analysis of key concepts in this area, it is not so clear which logic or which tools from logic to use for investigating games, actions and social software. For this reason, they have organized a discussion session on the theme "Battle of the Logics: Temporal Logic, Dynamic Logic, Game Logic, Logic for Belief Revision . . . Are There too Many?" Participants in the discussion are logicians of four different stripes: a Temporal logician, a Dynamic logician, a Philosophical logician, and a Mathematical logician. As always, the computer scientist is also present. A Multiagent System Designer who uncharacteristically does not know a lot about logic has just been referred to the four logicians for advice.

Multiagent System Designer: As a multiagent system designer, I am usually not that concerned with logic. Of course I use a bit of logic every now and again. A good programmer cannot do without it and should have a good understanding of logic, but I am more concerned with the features and the desired behavior of the system I am designing, than with logical aspects of multiagent systems. Even when I am looking for a logic to support me in the design process, I notice that there is a whole bunch of logics that I could use: temporal logic, dynamic logic, belief-desire-intention logic, which is usually called BDI, and so on. Since there are so many logicians here, I am sure that you could point me in the right direction and tell me what logic I should be using.

Temporal logician: I think you will want to use temporal logic. As Fagin, Halpern, Moses and Vardi have shown in their wonderful book *Reasoning about Knowledge* [88], extensions of temporal logic are best suited to reason about multiagent systems. The approach is very straightforward. As you will acknowledge, a multiagent system can best be represented as a distributed system of computer processors. These processors can be in different *local states* at different times. Together the local states constitute a *global state* of the system. You, as a designer, will have limited the number of global states that are allowed to occur and also which sequences of global states are allowed to occur. We call these sequences of global states *runs* . Formally, a multiagent system can best be thought of as a set of runs. I would even argue that a multiagent system simply *is* the set of runs of the system. We can then interpret a logical language with temporal operators on these runs, and with a little effort we can also interpret epistemic operators on them, and possibly other propositional attitudes. So, temporal logic with such a semantics of interpreted systems is the way to go.

Dynamic logician: Although I agree that temporal logic is a good approach to modeling *some* multiagent systems, I would not advise you to stick to just one logic. As you said, there are lots of approaches. At first this might seem somewhat unfortunate. After all, if logicians cannot even agree on one system for one application, then the whole enterprise must be flawed. However, you can also see this as an advantage. Apparently there are so many aspects of multiagent systems, and so many questions one might ask about social software or intelligent interaction in general, that one logic might not be enough. I have heard Johan van Benthem compare this situation to the mathematics of space, where geometry and topology are not seen as competitors. They are complementary approaches to the mathematics of space [1]. In the same way, different logics for multiagent systems can complement each other, together giving a rich perspective on intelligent interaction.

Temporal logician: Come on. We cannot expect our poor Multiagent System Designer to wade through the entire literature of logics for multiagent systems. This colleague is asking us for some very specific advice, and we should give it. Temporal logic is the best way to go.

Dynamic logician: If you insist, I will join you in your battle of the logics then. Besides temporal logics, our Multiagent System Designer might do well to consider using dynamic logic. In fact, dynamic logic might appeal more to programmers than temporal logic would. After all, the constructs in the dynamic language are very much like a programming language. There is a great textbook by Harel, Kozen and Tiuryn on dynamic logics [111].

Computer Scientist: Why use logics of action when game theory provides all that is needed for analyzing what goes on when rational agents interact? But I suppose the Game theorist has not been invited to this "Battle of the logics"...

Philosophical logician: No, we allow ourselves to be myopic today and to talk about logic, just this once. We have talked about possible uses of logic in game theory on another occasion (see the discourse starting on page 133).

Dynamic logician: Please let me continue my explanation of dynamic logic then. In the language of dynamic logic, there are modal operators corresponding to programs. To inductively build up these programs, there are usually some atomic actions α, as well as tests $(?\phi)$, which correspond to a program that tests whether a formula ϕ is true. Then there is sequential composition $(\pi;\pi')$, which simply says first do π, then do π'. You also have non-deterministic choice $(\pi \cup \pi')$, which allows either to execute π or π'. And last but not least, there is iteration (π^*), which tells you to execute π zero or more times. Sometimes, an intersection operator $\cap$ is also used; it is defined semantically by the intersection of the two accessibility relations.

Temporal logician: Aha, so intersection is a bit like the operator for distributed or implicit knowledge in the logic of knowledge: if two agents would pool all their knowledge, their implicit knowledge is what they would know. For example, if I know that p and you know that p implies q, then we implicitly know that q. The semantics of implicit knowledge is also based on the intersection of the agents' accessibility arrows, just like your intersection operator for programs, and the completeness proof for the resulting logic is a bit tricky because this intersection is not characterizable by modal axioms.

Dynamic logician: Exactly, the same problem also holds for dynamic logic with intersection. Incidentally, the dynamic language might seem somewhat far removed from programming languages, but you can easily represent an 'if p then π else π'' construction as $(?p;\pi) \cup (?\neg p;\pi')$, and a 'while p, do π' construction as $(?p;\pi)^*;?\neg p$. This logic is my personal favorite, and I recommend you to take a look at it, also for specifying and verifying your multiagent systems.

Philosophical logician: You seem to be glossing over a very important philosophical point that is relevant to the current discussion. "Useful" is a *relational* concept. One cannot discuss the usefulness of an object, without specifying the purpose for which it is to be used: One cannot say that a car is useful on its own. However, if one wants to go from A to B, a car might be very useful. It is these simple philosophical points that seem to complicate discussions between non-philosophers all the time. A basic course in philosophy should be mandatory in any academic programme. This would save us a lot of wasted time. So, what would you like to use these logics *for*?

Multiagent System Designer: Well, obviously, to help me design, implement and analyze systems of interacting agents, because that happens to be my job. In fact, I have many applications in mind. Perhaps a suitable logic could help me design a multiagent system by providing a nice logical language to write down a specification of desired system behavior. Perhaps logic could help me verify that the system I have designed indeed has the desired properties I had in mind while making it. Perhaps logic could also warn me not to try to implement impossible systems, such as a decision method for the provability of formulas in predicate logic.

Mathematical logician: Perhaps it would be nice to ask ourselves more generally what logics are supposed to be good for. When you look at the historical roots of formal logic, you see that modern logic arose from a desire to provide a firm foundation for mathematics. Think about the so-called logicist program of Frege and Hilbert's Program, where the aim was to found mathematics on logic. Alas, Hilbert's Program in its original guise failed on two counts: Gödel's incompleteness theorems showed that the supposed foundations, which at that time were represented by Russell and Whitehead's axiomatization of arithmetic in *Principia Mathematica*, were incomplete, and that such systems could not even prove their own consistency. Second, Turing and Church proved that first-order logic is undecidable. Still, Hilbert's Program has proved to be very fruitful for mathematical logic, and nowadays many interesting revisions of Hilbert's Program are around, that try to justify ideal mathematics by restricted means. For example, people like Feferman, Kreisel, Friedman and Simpson have shown that a lot of scientifically applicable mathematics can be based on weak subsystems of analysis, which are reducible to finitary mathematics [211].

Computer Scientist: So, logicians have by no means lost their interest in very precise mathematical proofs. And now we even have computational proof assistants as powerful as Coq, a direct descendant of Automath. Isn't it fascinating that a group of researchers has recently succeeded in constructing and checking a completely formal proof of the four color theorem in Coq [103; 104]?

Philosophical logician: That's all very well, but do not forget that logic started with Aristotle who had a somewhat broader view of logic than just mathematics. He was thinking about argumentation and science in general. If you want go back to the real historical roots of logic, you will see that its purpose is twofold. On the one hand, logic is a normative tool to assess the validity of inferences. On the other hand, due to its precision it clarifies and explicates intuitions one has about complex concepts.

Multiagent System Designer: This is a bit too abstract for me. May we turn to a specific type of logic? In my field I have heard a lot about logics for beliefs, desires and intentions, BDI logics for short. Could you clarify for me what use these logics would have for me?

Temporal logician: BDI logics are simply extensions of branching time temporal logic as it was developed in theoretical computer science to investigate distributed systems. The founders of BDI logics wanted to formalize the concept of an agent. You could view BDI logics as a formalization of work done in philosophy on intentions and planning [187].

Philosophical logician: In philosophy, intention has been the subject of study for years. It seems that people in artificial intelligence are always reinventing the wheel, because this is exactly what Michael Bratman proposed [39; 40]. He said that if one has an intention to do something in the future, one forms a partial plan, that one can fill in along the way. For example, if one wants to go to New York, there are many ways to get there. So, one forms some sort of highly abstract plan: For example one might want to travel by air rather than by sea. Some things one may leave to the very last minute before one starts executing the plan. One may for example decide to take a taxi to the airport, but one would probably not have selected a particular taxi in advance. It is quite clear that intentions and plans are very closely connected.

Temporal logician: Yes, and Bratman's is exactly the work upon which BDI logic is based. So the researchers in artificial intelligence have bought Bratman's very nice wheels instead of reinventing them. You can view BDI logics as providing a specification for the implementation of real agents.

Mathematical logician: So can someone explain the basics, please?

Temporal logician: Just think of the branching time logic CTL, computation tree logic. At each time point, the tree can branch to several successors according to different events taking place, or atomic actions if you like. In the language, you use temporal operators like inevitably (on every branch), optionally (on some branch), eventually (at some future point on the current branch), next, and until. Now Rao and Georgeff's idea was to combine this with operators for beliefs, goals and intentions. In the model, this would lead to a number of time trees, where for example the agent believes a formula at a time in the current tree if that formula holds in all belief-accessible time trees at the corresponding point in time. So, it's just as you would expect. If you want to have obvious axioms in your system that say things like "if you intend something, then you also desire it", this corresponds to a semantic property such as "every desire-accessible world has a subtree that is an intention-accessible world".

Multiagent System Designer: I find it hard to believe that such an abstract logic has anything to do with working systems. If BDI logic really is about this notion of agency, it is a philosophical exercise.

Dynamic logician: Well, BDI logic can be used for planning and that's really relevant for artificial intelligence. BDI logics are on the supply side of social software. They provide social procedures. They do not aim to describe reality, but they are going to help make things reality by providing specifications that can be implemented.

Philosophical logician: Why don't you give us a concrete example?

Temporal logician: A well-known automobile factory has constructed a prototype of a conveyor belt system for car manufacturing [123] based on a BDI architecture. The usual method of manufacturing cars is that a central controller pre-plans the whole production process for a day, but then you lose a lot of time if one of the machines breaks down during the day, as often happens. Instead, Jennings and his colleagues designed a decentralized control system. All machines and all manufactured parts were conceptualized as agents with their own objectives, such as "get myself to the end of the manufacturing line after a specified set of operations has been performed on me". Then constant negotiations among different agents took place, following the Contract Net Protocol, where machines bid for the opportunity to carry out operations on the parts [198]. All these objectives and outcomes of the task allocation protocols were represented as agents' desires and intentions. It turns out that the BDI-based decentralized system is much more flexible and robust in the face of an uncertain dynamic environment than the usual centralized one.

Multiagent System Designer: That's a really neat application, I'm impressed! I will look up that conveyer belt paper and see if I can build upon Jennings' neat work. Still, isn't the notion of agency in BDI logics too much like the notion of a player from game theory to be of practical use?

Dynamic logician: Indeed there are close connections between multiagent systems and game theory, and what goes on in games can also be captured in dynamic logics [17]. I do not want to go too deeply into games here as we have already talked about them elsewhere (in the Chapter starting on page 133), but let me remind you that the notion of agency is one of the key concepts to be analyzed in both fields.

Mathematical logician: I once attended a lecture on BDI logics and remember being surprised by the high number of different aspects that were mixed in one logic. It might have been the case that this was due to the area still being so young, but I would consider it unwise to develop a logic that is so rich in language and semantics. These rich systems might seem attractive when you want to write things down in your logic, but any metalogical result is very hard to obtain. I prefer simple systems with small languages, with which one can obtain beautiful results.

Philosophical logician: Again it seems a question of the purpose of one's enterprise. To me, it seems as though in mathematical logics the axiomatization of a logic comes first. Because one desires a certain elegance of axiomatization, one consequently makes the language very poor. I remember certain mathematical logicians whose preferred fragment of propositional logic only contained implication and absurdity. Of course, this language is truth-functionally complete, and proofs with induction on the language are less cumbersome, but in this way, other philosophically important logical operators are ignored. When I develop a logic, the language simply contains logical operators for those concepts I deem important.

Temporal logician: Maybe you should start with model theory instead of language. Indeed, a logic is used for reasoning about certain structures. You should first capture these structures, then you can decide on the language. Remember that there are lots of ways to represent time. One can have branching time or linear time, or one can have interval based models [86; 23]. Once you fix your models, you can interpret all sorts of languages on them. But the models come first.

Mathematical logician: Hey, what are you all quibbling about? Doesn't every well-trained logician know that once you have a sound and complete system, there is a one-to-one correspondence between syntax and semantics?

Dynamic logician: Of course we all know that, and I guess my colleagues here were just getting a little carried away talking about private tastes. In fact, I also think of the models first, and language and axiomatization later. This is not because I think the models are somehow fundamental. They just give me the best intuitions in developing a logic. I can well imagine that this works differently for other people, though.

Computer Scientist: Actually, there is one perspective from which your quibble between models and deductions is important, and that is feasibility. Halpern and Vardi describe this very well in their paper *Model checking vs. Theorem Proving: A Manifesto* [108]. In good old-fashioned artificial intelligence, an agent's knowledge was represented as a knowledge base, a collection of formulas. An agent was said to know something if it was provable from his knowledge base. But as the fathers of AI, and especially McCarthy, found that first-order logic was the logic for knowledge representation, this meant that the theorem-proving approach led to undecidability. In the model-checking approach, in contrast, you only need to check whether a given formula holds in a database, and that problem takes up memory space only polynomial in the size of your data: it is in PSPACE.

Temporal logician: In my field, there are also such striking cases where model checking is much more efficient than theorem proving. For example, if you want to verify a finite-state program, let's say a communication protocol, with respect to some specification that can be expressed in a branching time logic. Then the theorem-proving way would be that you first completely characterize your protocol by a temporal formula: You just need to describe all possible transitions in all possible global states. Then you need to check whether this description implies your specification. Unfortunately, checking this is not tractable because the validity problem for branching time logic is EXPTIME complete.

Multiagent System Designer: Sorry, guys and girls, all this complexity stuff with PSPACE and EXPTIME complete goes way over my head.

Computer Scientist: OK, so let me fill you in on the four most famous complexity classes. Computer scientists are interested in classifying problems by how much computational resources, like time and memory, they take to solve, as a function of the length of the input of the problem. Problems that take up time polynomial in the length of the input are in the class P, and those problems are usually said to be tractable. For example, think of the problem whether a certain valuation satisfies a given propositional formula - this corresponds to checking a single row in a truth table, which can clearly be done in linear time. The next important class is called NP for non-deterministic polynomial time. This class includes problems that can be described as "guess a polynomially short potential solution, and then check in polynomial time whether this guess indeed forms a solution". A typical example of a problem in NP is satisfiability for propositional formulas, abbreviated as SAT: guess a valuation, and then check in linear time whether it indeed satisfies the formula. Now the interesting thing is that there are no problems in NP that are essentially more difficult than satisfiability: Cook proved already in 1971 that each problem in NP can be easily translated to a suitable instance of SAT. Such problems like SAT that are in NP and are also among the hardest in NP, are called "NP-complete". As you probably know, it is still unknown whether P and NP are really different. If they aren't, that could have serious repercussions for public key cryptography (see the Chapter starting on page 197).

Multiagent System Designer: And if you prove it one way or another, the Clay Mathematics Institute will give you a million dollars, right?

Computer Scientist: Right, but being a Buddhist, the money doesn't interest me much. Let me tell you about two other relevant complexity classes. One is called PSPACE, and contains those problems that can be solved using memory space polynomial in terms of the input. A typical example is the model checking problem for predicate logic that I just mentioned for databases. Another one is the satisfiability problem for the most common modal logics [33, Chapter 6]. Both of these problems are in fact PSPACE-complete, again in the sense that all other problems in PSPACE can be reduced to them. It is immediately clear that NP is included in PSPACE, because short guesses and polynomial checking of them can never take up more than polynomial space. But again, nobody has yet found out whether NP and PSPACE are really different classes.

Multiagent System Designer: Now I wonder how EXPTIME fits into this picture. Surely exponential time is really more difficult than polynomial time?

Computer Scientist: Indeed it is, and that's in fact the only equivalence among the four complexity classes P, NP, PSPACE and EXPTIME that has been disproved. The simple thing we know is that PSPACE is included in EXPTIME - this is done by a nice proof, which you can look up in classical textbooks on complexity theory, such as Papadimitriou's [171]. A typical problem in EXPTIME is the satisfiability problem for propositional dynamic logic, where satisfying a formula that includes the Kleene star operator may require tree models of exponential depth [33, Chapter 6].

Dynamic logician: So, just to sum up all relations: P is included in NP, which is included in PSPACE, which is in turn included in EXPTIME, and the only inequality that has been proved is the one between the two extremes P and EXPTIME.

Multiagent System Designer: Wow, so much is still unknown in complexity theory! Doesn't that mean that computer science has been built on quicksand?

Computer Scientist: You could view it that way, but personally I rather think that these problems are at the heart of computation and show the depth of my subject.

Dynamic logician: I think we can now safely return to our earlier discussion of theorem proving versus model checking for finite-state programs such as communication protocols with respect to specifications that are represented by branching-time formulas.

Temporal logician: Thank you for getting us back to this main branch of our discussion. In the early eighties, Clarke and Emerson found out that you could represent a finite-state program by a Kripke model. The worlds of the Kripke model represent possible global states of your program, and the accessibility relations represent possible transitions. The great thing is that the Kripke model does not get out of hand: it is just about the same size as your program. Now checking whether your program satisfies the specification amounts to checking whether the specification holds at the world in your Kripke model that corresponds to the initial global state. And you can do this in time just linear in the sizes of the specification formula and your Kripke model [108]!

Computer Scientist: Unfortunately I have to temper your enthusiasm about this low complexity a bit. In model checking problems, the model is often assumed to be part of the input, so a seemingly attractive complexity result like "linear time in terms of the size of the input" is sometimes misleading, as these models can be very large in practice [118].

Philosophical logician: I think that the complexity of a logic should also be viewed in relation to its *use*. For example, in the context of cryptography, high complexity is a feature, not a bug.

Multiagent System Designer: I am of course not a logician, but, returning to the question where to start, I think I would start on the language instead of the model. There are just properties that you want to express. I do not want some very small language in which it takes a lot of work to express some basic concepts.

Dynamic logician: Ah, but that is exactly what can get me excited: A very simple language with great expressivity. I agree with our Mathematical logician that having language and semantics that are rich might make metalogical results difficult to obtain. Even worse, it could be that the system "does" things that are incorrect, but you are unaware of this because the system is too complex. I am reminded of what Albert Visser once said in a talk on logic and linguistics: Logicians prefer small and correct theories, and linguists prefer big and incorrect theories. That is to say, when a logician tries to capture an aspect of natural language, he or she develops a dedicated system for a small fragment of natural language. Linguists, on the other hand, try to capture all of natural language in their system. They are bothered that the fragments that the logicians use for describing parts of natural language do not capture *all* of natural language. But their wish to cover everything leads them to adopt theories that contain inconsistencies, which in the eyes of logicians is committing mortal sin after mortal sin.

Mathematical logician: I see what you mean. Just like the linguists you mentioned, computer scientists sometimes seem to construct logics that can express everything you want, but the semantics and axiomatization might be mistaken here and there.

Computer Scientist: I object! No one in his right mind could claim that contributions to conferences such as Logic in Computer Science present shaky semantics and axiomatizations. Use of logic in computer science is often very subtle and we can compete with the best of mathematical logicians in our use of abstract structures like locales, quantales and co-algebras. I think you have ample reason to tone down your arrogance. Mathematical logic has never been viewed as mainstream mathematics, and there has been only little contact between mathematical logic and the natural sciences. On the other hand, you can hardly over-estimate the role of logic in computer science. Just think of the relations between logic and complexity, the use of predicate logic as database query language, the influence of type theory on programming language research, and the use of modal logics in multiagent systems. You should definitely read the classic paper *On the unusual effectiveness of logic in computer science* [106].

Mathematical logician: Ahem, sorry about that.

Multiagent System Designer: To go back to the previous point, I feel sympathetic towards the linguist perspective, and start with all the expressivity that you need. After all, even as a programmer, first you build a system and then you try to debug it.

Computer Scientist: Now wait a minute, that is not the state of the art in software development at all! Nowadays we start with requirements and specifications, we use logic to check these specifications, and we let the implementation process go hand in hand with unit testing and specification-based random testing. For almost all aspects of this process, logic is highly relevant.

Multiagent System Designer: Point taken. What I was worried about is, what good is a logical system that at best will only do part of what I want it to do?

Dynamic logician: I understand your concerns. But in that case, instead of taking the risk of developing an incorrect theory, you can follow a piecemeal approach. You first create dedicated systems that do only part of what you really want and then you extend and combine systems. Initially we only focus on aspects that are interesting for us. Dynamic epistemic logic, for instance, is only about information change [66]. It does not capture anything else. This is also a valid approach.

Philosophical logician: Of course a piecemeal approach assumes that a problem can be thought of and solved in an analytic fashion. This is a very old philosophical discussion, of which I am afraid none of you are aware. One can imagine that there are problems that can only be solved as a whole. That is to say, there can be a need for holism. A system that only deals with some aspects of the problem will in that case always be dreadfully misguided.

Dynamic logician: Can you give an example of this?

Philosophical logician: The notion of obligation is very much connected to the notion of action. One is usually obliged to do something, to take an action. And the obligation is met when a certain action has taken place, and afterwards there may not be an obligation anymore. Therefore a philosophically sound deontic logic needs to be grafted on a logic of action. The work of John Horty is a nice example of this [120].

Dynamic logician: But deontic logic and the logic of action have developed separately. A holistic approach might have been too difficult initially. Moreover, as has recently been shown, the notion of knowledge is also very important for the notion of obligation [170]. Yet, epistemic logic was very fruitfully developed on its own.

Philosophical logician: I will admit it, but in order to grasp a concept in full, one cannot leave out crucial aspects.

Mathematical logician: Even if an analytic approach is possible, it may not be straightforward at all. Given two different logics, it seems highly non-trivial to combine them into one logic.

Temporal logician: It is indeed. When you investigate the complexity of combinations of logics, you might like to turn to general results on the transfer of the complexity of satisfiability problems from single logics to their combinations: isn't a combination of a few PSPACE-complete logics, with some simple interdependency axioms, automatically PSPACE-complete again? However, it turns out that the positive general results that do exist (such as those in Edith Spaan's Ph.D. thesis [216]) apply mainly to minimal combinations, without added interdependencies, of two NP-complete systems, each with a single modality.

Dynamic logician: Even more dangerously, I've heard of some very negative results on the transfer of complexity to combined systems. Listen to this: there are two "very decidable" logics whose combination, even without any interrelation axioms, is undecidable. For the first logic, let's take a weak variant of dynamic logic with two atomic programs, both deterministic. Take the sequential operator ; and the intersection operator $\cap$ as only operators. Satisfiability of formulas is in EXPTIME, just like for propositional dynamic logic itself. For the second logic, take the logic of the global operator A (Always), which just means what you would think, namely that the formula it is applied to is true everywhere throughout the Kripke model. Satisfiability for this logic is in NP. Blackburn and Spaan have shown that the minimal combination of the two logics is not only *not* in EXPTIME, but even undecidable in any finite time. This goes to show that we need to be very careful with any assumptions about generalizations of complexity results to combined systems [34;

33].

Multiagent System Designer: But those are just artificial examples made up to achieve horrendous undecidability results. Let us stick to a realistic multi-modal logic such a BDI combination of the standard logics for beliefs, goals and intentions, and add some reasonable interdependencies, such as the axiom that having an intention implies having the corresponding goal.

Temporal logician: That combination turns out to be PSPACE-complete, so no better or worse than its individual component logics [79].

Computer Scientist: Here I agree with you, the situation is not so bad. You may also be interested in Gabbay's *fibring* as a general approach to combining logics. I will try to tell you roughly what it is, but you all should really read his book about it [97]. Given two logics, let us say linear temporal logic and epistemic logic, the language of their fibring is obtained by combining all atomic symbols and operations from both of them. As for deduction, you suppose that the two given logics have the same type of deductive system (for example, both a Hilbert style one, or both a tableau system). Then in the fibring, you can freely use inference rules from both. If the two original systems were schematic, this means that the inference rules can be applied to formulas including symbols from the "other" language, and fibring them makes sense. The semantics is quite complicated, so you should just look it up in the book, but you could think of a fibred model as a cloud of points. At each point you can extract a model of the first logic and a model of the second one, so in our example you would be able to extract a time line as well as a model of epistemic logic.

Temporal logician: I do not really see the point of combining systems. Why would you want to model everything at the same time? And why should everything be in the language? It might be fine to have temporal models for multiagent systems with knowledge and just interpret an epistemic language on those models. There is nothing wrong with that.

Mathematical logician: I only know of one situation where the models can be uniquely described by the logic: propositional logic with only finitely many propositional variables. In that case the language is not only truth-functionally complete, but also expressive complete in the sense that for every model, there is a formula that is true in exactly that model and in no others. For other logics the models are much richer than the logical language can describe. Modal logics cannot distinguish bisimilar models for instance, but bisimilar models are not isomorphic.

Multiagent System Designer: What are bisimilar models?

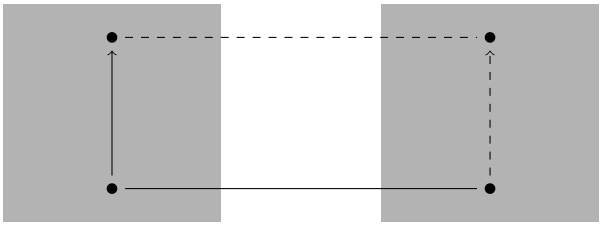
Dynamic logician: Do you know what a Kripke model is?

Multiagent System Designer: Yes.

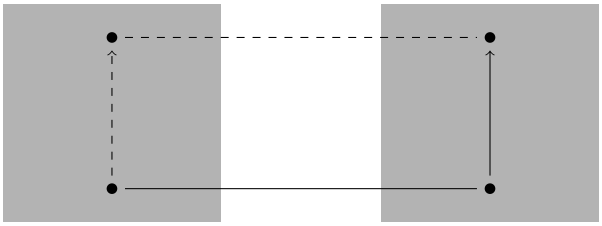
Dynamic logician: Good. A bisimulation is a relation between two Kripke models.

The concept of bisimulation was independently developed in automata theory, modal logic, and non-well founded set theory. Davide Sangiorgi has a nice paper on its history [199]. The idea is that two structures are bisimilar if their "behavior" is somehow the same. In automata theory that means that two automata accept the same language, in modal logic it means that two models satisfy the same formulas and in non-well-founded set theory it means that two sets are identical.

A bisimulation between two Kripke models is a relation between the worlds of the two Kripke models. Such a relation has to satisfy three requirements in order to be a bisimulation. First of all, if two worlds w and w' are linked by the relation, then they have to satisfy the same propositional variables. Secondly, for each accessible world v from w, i.e. by the accessibility relation in the one Kripke model, there is an accessible world v' from w' in the other model such that v and v' are also related. This is called the *forth* condition. Thirdly, for each accessible world v' from w', there is an accessible world v from w such that v and v' are also related. This is the *back* condition. Let me draw you a picture for the forth condition.



We have one model on the left and one on the right. If the two worlds below are linked and there is a world accessibly on the left, then there is a world accessible on the right such that those accessible worlds are also linked. The "if" part is the normal lines, the "then" part are the dashed lines. The picture for the back condition looks like this.



The relation is called a bisimulation because what can be done on the left can be done on the right and simultaneously what can be done on the right can be done on the left. The two models simulate each other simultaneously.

Modal logic cannot distinguish bisimilar models in the sense that they satisfy the same formulas. So in that sense, the models are much richer than the logic can describe.

Multiagent System Designer: I would not like that kind of situation. Why would you want rich models and a poor language? There seems to be something out of balance in that case.

Computer Scientist: Not at all. This is not a *defect*, this is a *virtue*. This is how process theorists look at modal logics. Processes are about choice and sequence, and one and the same process can be pictured in different ways. Take the process X of making a choice between doing a and b, and next performing X again. One can picture this as

$$\begin{tikzcd}
\bullet \arrow[loop above, "b"] \arrow[loop below, "a"'] & \text{or as} & \bullet \arrow[loop below, "a"'] \arrow[r, "b"] & \bullet \arrow[loop above, "b"] \arrow[loop below, "a"']
\end{tikzcd}$$

Both pictures describe the same process. These pictures happen to be bisimilar. So the "real process" is the class of all pictures that are bisimilar to the first picture.

Multiagent System Designer: I see what you mean. But how is it a virtue that different pictures can represent the same process?

Computer Scientist: The notion of process captures the essence of the picture. It tells us what is important and what is not.

Dynamic logician: In modal logic it tells you when two models are not essentially different.

Temporal logician: This depends on the logical language of course. It is quite easy to distinguish the two structures above using first-order logic.

Dynamic logician: It might be better to view this as something of a range of possibilities. On one extreme a feature of the models can be entirely captured such as the propositional case that our Mathematical logician mentioned earlier. At the other end of the spectrum, the extra structure cannot be captured in the language at all. There might be good reasons to be on one side of the spectrum or the other, or somewhere in the middle.

Temporal logician: I don't think we should have a general discussion about this. Why don't you give us a specific example?

Dynamic logician: Indeed, so let's consider the case of AGM-style belief revision versus the dynamic doxastic logic of Segerberg [136]. Although both systems deal with the same phenomena, they have a very different methodology when it comes to something being inside or outside the language. The basic ingredients of belief revision are so-called belief sets. These are simply logically closed sets of either propositional or first-order formulas. Then there are operations on these belief sets that correspond to changes in belief. Dynamic doxastic logic arose out of the idea to internalize these operations in a logic, so to view the operations on belief sets as modal operators in an extended logic.

Philosophical logician: I must say that I really appreciated the original AGM paper [2]. I do not see what extra insight is gained by internalizing belief change operators.

Dynamic logician: In their paper, Leitgeb and Segerberg argue that one of the main advantages is that by putting everything in the language you can nest belief operators and change operators [136]. In this way you can explicitly formalize beliefs about changes, as well as changes of belief. I agree that this is a great advantage. Rather than formalizing belief change, you formalize reasoning about belief change.

Temporal logician: This reminds me of the two different schools in addressing the effects of communication in multiagent systems. There is the famous school of Fagin, Halpern, Moses and Vardi [88] on the one hand, where the semantics are based on interpreted systems. These are in turn based on a temporal structure such as linear or branching time, with added epistemic structure reflecting agents' observational powers. Such interpreted systems work wonders when you want to model processes that arise when a protocol is followed through time. Of course the corresponding language of Epistemic Temporal Logic (ETL) combines epistemic and temporal operators.

The other school is called Dynamic Epistemic Logic (DEL). There, the epistemic events such as public announcements are included in the language. In order to describe communicative processes, you then have to compute so-called product updates in stages, starting from an initial situation.

Computer Scientist: As a computer scientist, I really appreciate the work from the Halpern school: they think as computer scientists and give many examples of how logic can be used to specify and analyze protocols for communicating systems. In fact, one of my favorite papers of all time is Joe Halpern and Lenore Zuck's *A little knowledge goes a long way.* They introduce the concept of a knowledge-based algorithm and give an extremely nice and convincing logical analysis of such computer science classics as the alternating-bit protocol. It seems to me that the DEL examples, which often involve puzzles or simple card games, have much less of a "real life" flavor.

Temporal logician: I keep wondering whether our two schools are really so different as some authors claim.

Dynamic logician: You have timed your question very well indeed. Recent work by Van Benthem, Pacuit, Gerbrandy and others shows that if you look at it the right way, you can find very interesting analogies between ETL and DEL. Rather than reducing one framework to the other, these authors aim to merge them. This program has already led to some interesting results and techniques, such as a new kind of modal correspondence theory which relates properties of DEL protocols to corresponding ETL properties. Also they have proved some completeness theorems for ETL model classes generated by DEL protocols. So instead of remaining rivals, these logicians now use ideas from our DEL school to add fine structure to ETL [24].

Temporal logician: But aren't ETL-style logics much more complex than DEL?

Dynamic logician: To be sure, another article that embodies the temporal-dynamic unification program is aptly named *The tree of knowledge in action: Towards a common perspective* [25]. It explores complexity issues around epistemic logics from both the temporal and the dynamic point of view. At first sight the ETL view on branching time gives rise to models that quickly get out of hand. Especially if the added epistemic structure enables some grid-like structure to be encoded, for example because of properties like Perfect Recall and No Miracles, undecidability may result. Van Benthem and Pacuit go on to use ETL-style methods to investigate the complexity of some DEL-like logics that live close to the edge of undecidability. For example, the result by Miller and Moss that the dynamic epistemic logic of public announcement with program iterations is undecidable may be contrasted with the fact that adding temporal "past" operators to DEL does not destroy decidability. Also in this case, methods from one camp are fruitfully used to chart the complexity of logics from the other camp.

Mathematical logician: As the paper by Van Benthem and Pacuit suggests, it seems that there is still a certain amount of strife between temporal and dynamic camps, though. I remember that Johan van Benthem compared this situation with the start of computability theory. There were several approaches, from recursive functions through lambda calculus to Turing machines. Rather than bicker and argue about which approach was the best, the logicians at the time proved that these definitions were equivalent and embraced Church's Thesis that every effectively calculable function is general recursive [25].

Dynamic logician: And rather than weakening their own position, their joint forces strengthened the field enormously because it turned out that the notion of computability is quite stable. Indeed, "seeing differences may make for short-term gains, seeing analogies leads to a long-term common cause" [25].

Philosophical logician: I do not think such a grand unification can ever be achieved in the case of logics for intelligent interaction. There are simply too many systems and they seem quite incomparable. Moreover I think it is nonsensical to aim to achieve unification. It will only give us a few extra theorems, but no better understanding of the concepts involved.

Temporal logician: There is a danger if one never compares systems. A lot of time will be wasted if different people work on essentially the same problem, because they are blind to the fact that the systems they use are essentially equivalent. The book by John Horty *Agency and Deontic Logic* for instance uses branching time temporal logic and I think his approach fits in very nicely with the temporal logics that are used in computer science [120]. It would be very useful if the people working on logics for "seeing to it that" and the people working on computation tree logic would compare notes.

Dynamic logician: I've heard that Broersen, Herzig and Troquard have started doing so: they found some nice first results on the connection between Alternating-time temporal logic and STIT logic [41].

Temporal logician: Indeed, we have hardly discussed Alternating-time temporal logic or ATL today, but I think it is also a worthwhile approach for specifying multiagent systems [118]. If temporal logics tell you *when* you will be happy, and dynamic logic can express *how* it is done, ATL can speak about *who* will achieve this state for you. This seems to be quite an essential aspect when you are interested in intelligent interaction.

Multiagent System Designer: I wish there were a map of the logics of intelligent interaction, showing what the connections between different approaches are and charting in what ways some of them are equivalent. Then I would, depending on the purpose, be able to use one of those systems off the shelf.

Dynamic logician: Let a thousand flowers and trees of knowledge bloom in the logical landscape! Our task is to be both gardeners and cartographers, so that everyone can find his way.