COMPUTATION AS SOCIAL AGENCY: WHAT AND HOW [1]

Johan van Benthem, Amsterdam & Stanford, http://staff.science.uva.nl/~johan

1 Views of computing, from 'what' to 'how'

This Turing Year has been the occasion for lively debates about the nature of computing. Are we on the threshold of new styles of computation that transcend the limitations of the established paradigm? Let us briefly recall three classical themes of the golden age when Turing and his generation made computation a subject for mathematical inquiry, and hand in hand with that, for practical development. First of all, by analyzing the bare basics of mechanical computing, Turing defined a *Universal Machine* that can compute the result of any algorithm on any matching input, when both are presented in suitably encoded form. This notion then supported the subsequent development of Recursion Theory, bringing to light both the basic structures and powers of effective computation, but also its limitations as exemplified in the undecidability of the Halting Problem. On the basis of this and other, equivalent models proposed at the time, *Church Thesis* then claimed that all effectively computable functions over the natural numbers (a canonical domain that can mimic non-numerical computation by various encodings going back to Gödel and others), coincide with the 'recursive functions', that can be computed on Turing machines. As the power of this paradigm became clear, it was suggested in the famous *Turing Test* that computation might well emulate human cognition, to the extent that in conversation, humans would not be able to tell whether they are interacting with another human or a machine.

Now 80 years later, computer science and information technology have transformed the world of both machines and humans in sometimes wholly unpredictable ways. Given the experience obtained over this period, can we spring the bounds of the classical age, and compute a larger class of functions/problems after all? There are interesting current debates in the US and Europe on this theme, with proposals ranging from using infinite machines to letting the physical universe do the computing in its own ways (Loewe et al., Barry Cooper). I am not going to enter these debates here, except for one basic comment. It seems important to make a distinction here between two issues:

---

[1] This discussion paper is a version of a talk presented at the Manchester ASL Logic Colloquium in the Turing Year 2012, the ESSLLI Summer School in Opole, and several presentations after that.

(a) *What* can we compute, and (b) *how* can we compute it?

Somewhat apodictically, my view is this. I see no evidence in current debates that we can compute more than before, forcing us to extend the calibration class of recursive functions. But then, this 'What' question is not of great interest. Of much greater interest is a 'How' question, not addressed by Church's Thesis, namely, what are natural styles of computing? Or if you insist on 'what' questions after all: do not ask what is *computable*, but what is *computing,* viewed as a kind of process producing characteristic forms of *behavior*.

Right from its start, the history of computer science has shown an outburst of ideas on these themes, and this paper will be about one of these: computation as social agency. My discussion will have a logical slant, reflecting my own work in this area (van Benthem 2009, 2011), and I am not claiming to represent public opinion in computer science.

2 Computer science as a hot spring of ideas

Before I start with my own theme, here is some very quick background that not all of my fellow logicians interested in the foundations of computing seem aware of.

Logic and fine-structuring views of computing Turing machines have opaque programs telling the machine in complete detail what to do in machine code, making heavy use of that old enemy of perspicuity called 'go to' instructions (Dijkstra 1968). Real computer science took off when higher programming languages were invented, that represent higher-level ideas on the sort of computation taking place. One can think of programs in such languages as 'algorithms' that describe the essence of some computational task at some more suitable abstraction level. Different programming languages have given a wealth of perspectives on this, often drawing on traditions in logic.

For instance, imperative programs like those of Algol or C+ may be viewed as a 'dynamified' version of logical formulas in standard formalisms like predicate logic, telling the machine what sort of results to achieve given certain preconditions. Such systems lend themselves well to model-theoretic semantics in the usual logical style (first-order, modal, or otherwise), witness the development of Hoare Calculus and Dynamic Logic. On the other hand, there are functional programming languages like LISP or Haskell, akin to systems of lambda calculus and type theory, closer to the proof-theoretic and category-theoretic traditions in logic. And of course, there are many other styles that do not fall simply into this dichotomy, including object-oriented programs, logic programs, and so on.

The semantics for this large family programming languages have provided a wealth of matching process models that offer many deep answers to the issue of how we compute.

Distributed computation and process theory One major challenge around 1980 was a theoretical reflection on the practice of distributed computing emerging at the time. One major development here, moving up the abstraction level beyond programming languages, was the invention of *Process Algebra* (Milner, Baeten, Bergstra & Smolka), an abstract view of processes as graphs modulo some suitable notion of structural behavioral invariance, often some variant of bisimulation. While it is true to say that no consensus ahs emerged on one canonical model of distributed computation, comparable in breadth of allegiance to Turing machines, a deep process theory did emerge with many features that have no counterpart in the classical theory of sequential computing (van Emde Boas). Abstract process theories are still emerging in the foundations of computation. A noticeable new development has been the birth of *co-algebra*, as a theory of computing on infinite streams, tied to fixed-point logics and category-theoretic methods (Venema 2007). My point here is very modest: thinking about the foundational hows of computation is a productive line of thought that shows no signs of abating yet. [2] For instance, in the last 15 years, a striking model for multi-agent distributed computing has been the introduction of *game models* (Abramsky, and in another paradigm Graedel, Thomas & Wilke), leading to new encounters between computer science, logic, and game theory (van Benthem 2013).

3 Computation and social agency

It is often said that Turing took the human out of the term 'computer', extracting only the abstract procedures behind their pencil-and-paper activity. In that light, the Turing Test then added insult to injury, since the computer thus defined might then even dispel the mystery of our other intelligent tasks just as well. And even without grand reductionist aims, it is undeniably true that computational models have proved of immense value in studying what might be considered typical human activities, such as conversation as a form of computed information flow driven by natural language functioning as a programming language (van Benthem 1996). I will mention more examples below.

[2] My subsidiary point is addressed more to some of my fellow logicians, who sometimes think that no deep insights worth our august attention have ever come out of actual computer science. My point to them is simply that there is 'life after Turing': deep thinkers on the foundations of computation such as McCarthy, Dijkstra, Hoare, Pnueli, Milner, Pratt or Abramsky have a lot to teach us.

But there is an opposite stream as well. Much of the history of computer science can also be seen as a case of the 'humans striking back'. Here is a mild instance of this phenomenon. Around 1980, Halpern and others started the TARK tradition of enlisting the delicate yet powerful understanding that we have of human agents with knowledge and social activity in support of modeling complex distributed protocols, how they function, what they do and do not achieve, and what might go wrong with them (Fagin et al. 1995). Now this may be a case of using metaphors, but the resulting revival of epistemic logic, broadly conceived, has had widespread repercussions in several disciplines. Likewise, and even much earlier, the development of *Artificial Intelligence*, though perceived by some as a reductionist replacement exercise, in fact gradually made computer scientists (and others) aware of the amazing subtlety of human behavior and skills in problem solving, knowledge acquisition, and social interaction. A wealth of logical systems arose out of this that also started influencing other areas far beyond computer science, including linguistics and philosophy. And finally, just consider the realities of computing today. What we see all around us are complex societies of humans and machines engaged in new interactive styles of behavior, nowadays even driven by conscious design of 'gaming', and it might even be thought that the real challenge today is understanding what makes this reality tick, rather than abstruse discussions about computing at infinity or in the Milky Way.

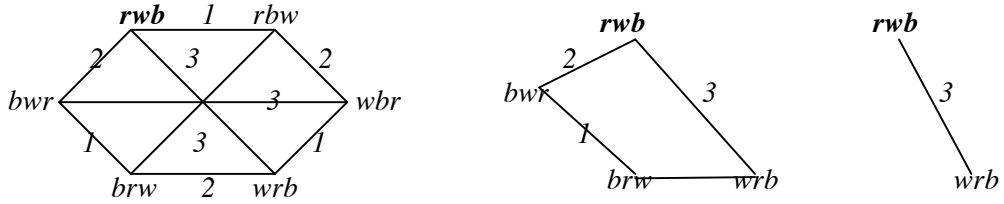
4 Conquering daily life: conversation as computation

Computational models are widely used in the study of human agency today, so much so, that the area of multi-agent systems combines features of computer science, epistemology, and cognitive science (Leyton-Brown & Shoham). Of the vast literature, I only mention one current strand as an illustration, viz. 'dynamic-epistemic logics' (van Benthem 2011).

Conversation and information update Simple games are a good setting for studying communication. Three cards "red", *w* "white", and "blue" are given to three children: ***1*** gets red, ***2*** white, and ***3*** blue. Each child sees his own card, not the others. Now ***2*** asks ***1*** "Do you have the blue card?", and the truthful answer comes: "No". Who knows what now? Here is what seems the correct reasoning. If the question is genuine, player *1* will know the cards after it was asked. After the answer, player *2* knows, too, while *3* still does not.

But there is also knowledge about others involved. At the end, all players know that *1* and *2*, but not *3*, have learnt the cards, and this is even 'common knowledge' between them. [3]

The Cards scenario involves a computational process of state change, whose basic actions are updates shrinking a current range. In the diagrams below, indexed lines indicate an uncertainty for the relevant agents. Informational events then shrink this range stepwise:



The first step is for the presupposition of the question, the second for the answer. In the final model to the right, both players *1* and *2* know the cards, but *3* does not, even though he can see that, in both of his remaining eventualities, *1, 2* have no uncertainties left. ■

The geometry of the diagram encodes both knowledge about the facts and knowledge about others: such as *3*'s knowing that the others know the cards. The latter kind is crucial to social scenarios, holding behavior in place. Indeed, at the end of the scenario, everything described has become *common knowledge* in the group *{1, 2, 3}*. [4]

Dynamic logics of communication 'Dynamic epistemic' logics describing this information flow, and changes in what agents know from state to state, have been found on the analogy of program logics in computer science. First, what agents know about the facts, or each other, at any given state is described by a standard language of *epistemic logic*:

$$p \mid \neg\varphi \mid \varphi \wedge \psi \mid K_i\varphi$$

while the corresponding epistemic models were tuples

$$\boldsymbol{M} = (W, \{\sim_i \mid i \in G\}, V)$$

with a set of relevant worlds W, accessibility relations $\sim_I$ and a propositional valuation V for atomic facts. Knowledge is then defined as having semantic information: [5]

---

[3] This way of understanding the scenario presupposes that questions are sincere – as seems reasonable with children. But our methods also cover the possibly insincere scenario.

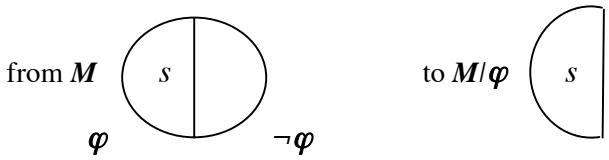
[4] Cf. Fagin, Halpern, Moses & Vardi 1995 for all these notions in games and computation.

[5] These epistemic models encode 'semantic information', a widespread notion in science, though other logical views of information exist (van Benthem & Martinez 2008).

$\boldsymbol{M}, s \models K_i\varphi$ iff for all worlds $t \sim_i s$: $\boldsymbol{M}, t \models \varphi$

Common knowledge $\boldsymbol{M}, s \models C_G\varphi$ is defined as φ's being true for all t reachable from s by finite sequences of $\sim_i$ steps. If necessary, we distinguish an *actual world* in the model.

Update as model change The key idea is now that informational action is model change. The simplest case is a *public announcement* $!\varphi$ of hard information: learning with total reliability that φ is the case eliminates all current worlds with P false:



We call this *hard information* for its irrevocable character: counter-examples are removed.

This dynamics typically involves truth value change for complex formulas. While an atom p stays true after update (the physical base facts do not change under communication), complex epistemic assertions may change their truth values: before the update $!p$, I did not know that p, afterwards I do. As with imperative programs, this may result in order dependence. A sequence $!\neg Kp; !p$ makes sense, but the permuted $!p ; !\neg Kp$ is contradictory.

Public announcement logic The dynamic logic *PAL* arises by extending the epistemic language with a dynamic modality for public announcements, interpreted as follows:

$\boldsymbol{M}, s \models [!\varphi]\psi$ iff *if* $\boldsymbol{M}, s \models \varphi$, *then* $\boldsymbol{M}|\varphi, s \models \psi$

The system of public announcement logic *PAL* can be axiomatized completely by combining a standard logic for the static epistemic base plus a *recursion law* for knowledge that holds after update, the basic 'recursion equation' of the system:

$$[!\varphi]K_i\psi \quad \leftrightarrow \quad \varphi \rightarrow K_i(\varphi \rightarrow [!\varphi]\psi)$$

Dynamics of other events Similar systems exist for updating agents' *beliefs*, defined in terms of truth in the *most plausible* epistemically accessible worlds. Here the variety of dynamic events increases. Beliefs can change under hard information, but also under *soft* information, where $\neg\varphi$–worlds are not eliminated, but made less plausible than φ–worlds. And similar methods again work for events modifying agents' *preferences* (Liu 2011).

Time and program structure Single events are just atomic actions that bunch together to form meaningful larger scenarios. Again, computational ideas are essential. Action and

communication involve complex programs with operations of sequential composition ***;***, guarded choice *IF THEN ELSE*, and iteration *WHILE DO*. Even parallel composition ***||*** occurs when people act or speak simultaneously. Here is a well-known illustration:

The Muddy Children "After playing outside, two of a group of three children have mud on their foreheads. They can only see the others, and do not know their own status. Now the Father says: "At least one of you is dirty". He then asks: "Does anyone know if he is dirty?" The Children always answer truthfully. What will happen?" As questions and answers repeat, nobody knows in the first round. But in the next round, each muddy child reasons thus: "If I were clean, the one dirty child I see would have seen only clean children, and so she would have known that she was dirty at once. But she did not. So I am dirty, too." This scenario falls within the above update setting, but we do not elaborate here. ■

Clearly, there is a program here involving sequence, guarded choice and iteration:

!"At least one of you is dirty" ; WHILE not know your status
DO (IF not know THEN "say don't know" ELSE "say know")

Temporal limit behavior Another interesting feature is the limit behavior in the puzzle, leading to a stable endpoint where updates have no further effect and agents' knowledge is in equilibrium. In particular, the children have common knowledge of their status in the limit model $\#(\mathbf{M}, \varphi)$ reached by the iterated updates $!\varphi$ of their ignorance assertion. So in the end, this statement 'refutes itself'. In other scenarios, like game solution procedures, the statement announced is 'self-fulfilling', becoming common knowledge in the limit. [6]

Limit features of computation over time can be studied in sophisticated fixed-point logics, but one simple case is just propositional dynamic logic *PDL* of basic imperative programs. The resulting setting has vast computational power (Miller & Moss 2005): the logic *PAL* with Kleene iteration of updates is Π^1_1-complete. Van Benthem 2007 has a positive interpretation of high complexity results for logics like these, namely that

"conversation has universal computing power":

any significant computational problem can be realized as one of conversation planning.

While this looks attractive as an observation about conversation as a paradigm for computation in general, there is a catch. The high complexity resides in the logic of reason-

[6] Limit features of belief revisions over time underlie formal learning theory: Gierasimczuk 2010.

ning about conversation, but as discussed in van Benthem 2011, conversational algorithms themselves might have low complexity as far as computational procedures go. [7]

Our examples and glimpses of wider implications may have shown how computational notions and techniques arise all the way in a basic human activity like conversation. For many further examples of 'communication as computation', we refer to the cited literature.

5 Daily life strikes back: computation as conversation

Let us now reverse the perspective. Starting around 1980, Halpern and his colleagues in what is now sometimes called the TARK community have shown how human metaphors of knowledge and social interaction, if made precise in logical terms, can be a powerful tool for specifying and proving properties of complex protocols for multi-agent systems. The book Fagin et al. 1995 is a landmark of the resulting program. But the borderline between a metaphor and the real thing may be thin. Increasingly, there seems to be a viable view that computing *is itself* a form of social behavior, mixing action and information much as humans do. Correspondingly, the same formal objects that act as programs for machines are also 'plans for humans. The two essential basic features of human agency then enter our understanding of computation: one it the *knowledge* of agents (perhaps also other attitudes, such as their beliefs and preferences), and the other their social *interaction*. In what follows, we take up these two themes separately, showing how they enter our view of computing in natural ways. Our major tool for highlighting these phenomena will be transformations from standard algorithms to knowledge-based social procedures.

6 Epistemizing computational tasks

This section is about what we call the phenomenon of *epistemization*, the introduction of agents' knowledge at various parts in basic computational tasks.

Epistemizing algorithmic tasks Consider the key planning problem of Graph Reachability (*GR*). Given a graph $\boldsymbol{G}$ with points x, y, is there a chain of arrows from x to y? *GR* can be solved in *Ptime* in the size of $\boldsymbol{G}$: a quadratic-time algorithm finds a path (Papadimitriou 1994). The same holds for reachability of a point in $\boldsymbol{G}$ satisfying a goal condition φ. The solution algorithm performs two related tasks: determining if a path exists at all,

[7] However, this complexity may go down on extended *protocol models* (van Benthem, Gerbrandy, Hoshi & Pacuit 2009) that constrain the admissible sequences of updates in each world.

and giving a concrete way or plan for getting from x to y. We will now consider various natural ways of introducing knowledge and information in this setting.

Knowing you made it Suppose an agent is trying to reach a goal region defined by φ, with only limited observation of the terrain. The graph $\boldsymbol{G}$ is now a model $(G, R, \sim)$ with accessibility arrows, but also the earlier epistemic uncertainty links between nodes. It is natural to ask for a plan that will lead you to a point *that you know to be in the goal region* φ. Brafman, Latombe & Shoham 1993 analyze a robot whose sensors do not tell her exactly where she is. They then add a *knowledge test* to the task, inspecting current nodes to see if we are definitely in the goal region: $K\varphi$. Given the *P-time* complexity of model checking for epistemic logic (Chapter 2), the new search task remains *P-time*. [8]

Epistemizing social tasks Many algorithmic tasks themselves come from social scenarios, witness the area of computational social choice (Endriss & Lang). Here, too, epistemization makes sense. Think of the basic computational task of *merging orderings*. In social choice theory, preferences of individual agents are to be merged into a preference order for the group as a whole. This way of phrasing started with Arrow's Theorem stating that no social choice procedure exists that satisfies some basic postulates of unanimity, monotonicity, context independence, and especially, absence of a 'dictator', an individual whose preference ranking always coincides with that of the group. These specifications are completely non-epistemic, which is somewhat surprising, since much of what we consider essential about democratic decision making has to do with privacy, and what agents may know or not know. But, there is even a mismatch between the usual base conditions and how they are interpreted intuitively in terms of agency. The existence of a dictator is problematic if we think of an individual who can abuse her powers: but for that, she should *know* that she is a dictator – and perhaps, others should also know (or not know) this. Thus, epistemic rethinking of the very scenario of social choice seems in order, and it is not even clear what a knowledge-based version of the basic theory would look like.

Other algorithmic tasks where similar points can be made occur in *game theory*. Indeed, the move from games of perfect information to games with imperfect information (Osborne & Rubinstein 1994) may be considered a case of epistemization in our sense.

---

[8] A general model for epistemic robots relying on possibly limited or defective sensors is proposed in Su 200x. This model has already led to new 'evidence models' for human agency (van Benthem & Pacuit 2011) that are more fine-grained than the standard epistemic models of this paper.

Two aspects of epistemization Our examples show two different aspects of introducing knowledge. One is that the *specifications* of what an algorithmic task is to achieve may come to involve knowledge, like saying we must know we are at the goal, This does not necessarily mean that the algorithm itself has to be epistemic. Many social algorithms are purely physical, such as folding ballot slips, though they do have epistemic effects.

The second step, then, makes the *algorithms themselves* contain knowledge aspects. One obvious place where this happens is test conditions for conditional action. We find it obvious that a computer 'checks' in its registers whether, say, $x = 1$, before performing an *IF THEN* task: truth and knowledge are easily confused here. But for more complex conditions, such as 'the battery is functioning', we can only perform *IF THEN* instructions if we *know* which condition holds. And there may be yet more subtle aspects of knowledge involved. Turing 193x says a machine should know which symbol it is reading, and even which state it is in: a rather human form of introspection. [9]

Epistemic programs Algorithms with conditions that we know to be true or false look like human plans. One format for epistemizing standard algorithms is the *knowledge programs* of Fagin et al. 1995, making actions dependent on conditions like "the agent knows φ" that can always be decided given epistemic introspection. [10] The language of the programs now also explicitly contains our earlier epistemic operators. Knowledge programs make sense in epistemic planning (Bolander et al. 2012), and also as definitions for uniform strategies in imperfect information games (van Benthem 2013).

A related way of epistemizing programs is offered by the earlier dynamic epistemic logics. Public announcements are closely related to 'test actions' that remove all epistemic uncertainty links between φ–worlds and $\neg\varphi$–worlds. The behavior of test actions, and that of many other ubiquitous informational actions, such as questions and answers, can be described in exactly the same logical style as before.

Further aspects of epistemization But once we entangle algorithms and knowledge, many further issues emerge, going beyond just opening a 'knowledge parameter' here and there.

---

[9] Turing himself thinks that these epistemic properties are guaranteed by having only *finite* sets of symbols and states. This is not the notion of knowledge used in this paper, since it seems to refer more to perceptual discrimination (cf. Williamson on the latter notion in epistemology).

[10] Some of the surprising cognitive algorithms in Gigerenzer & Todd 1999 have this flavor.

For a start, epistemic specifications or programs essentially refer to some agent performing the task, and then, the nature of those agents becomes a factor.

Different types of agent Epistemic algorithms may work for one type of agent but not for another. The literature on dynamic epistemic logic has mainly focused on agents with *Perfect Recall* who remember everything they knew at earlier stages of the process, and who also learn from observation only. But equally important are agents with bounded memory, such as finite automata. Various assumptions of this kind will be reflected in the epistemic logic of action. For instance, Perfect Recall holds for an agent iff the following commutation law for action and knowledge governs its behavior:

$$K[a]\varphi \rightarrow [a]K\varphi$$

This says that, if we know beforehand what an action is going to achieve, we will know its effect afterwards. This is crucial to consciously following a plan, though it can fail in other circumstances. [11] Other axioms that can be written in this language govern the behavior of finite automata. Clearly, such assumptions about agents influence what we can expect an epistemic algorithm to achieve – but I am not aware of any general theory.

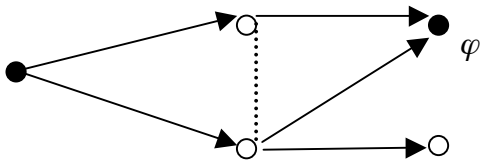
Know-how, and knowing a program So far we followed the mainstream of epistemic logic in letting knowledge apply to propositions. But our setting suggests a richer view. In addition to propositional *knowing-that*, there is *know-how,* embodied in algorithms, plans and procedures. Knowing how is the subject of much of our learning, perhaps even more than knowing that. And this know-how is related to an important notion in our natural language, that of knowing an *object*. In our setting, one obvious instance of this is what it means to 'know a program'. There seems to be no unique answer as to what this means, but here is a tie with propositional knowledge that seems relevant.

Suppose that we have an epistemic program or plan, knowing it seems to involve at least some clear grasp of its execution, not just being lucky. Should the agent know the plan to be successful: beforehand, and at all stages of its execution? There are two aspects to this mastery (see van Benthem 2013 for discussion and formal results). Suppose that the agent has an epistemic plan: does it follow that she knows its effects? It is easy to see that this is not always so, and hence we might use this as a stronger requirement on epistemized

[11] I may know that entering this enticing bar will lead to my swift moral downfall, but once I am inside, all such thoughts may have left my mind.

algorithms than we have imposed so far. But also, suppose that the agent knows now what the plan will achieve, will this knowledge persist over time as the plan is being followed?

Example For an illustration, recall the earlier problem of epistemized graph reachability. Let the agent at the root of the following graph trying to reach a φ-point:



The dotted line says that the agent cannot tell the two intermediate positions apart. A plan that reaches the goal is *Up; Across*. But after the first action, the agent no longer knows where she is, and whether moving *Across* or *Up* will reach the φ-point. ■

Much more can be said about when intermediate knowledge of effects does hold, but we merely cite one result from van Benthem 2013: agents with Perfect Recall have intermediate knowledge of effects for all knowledge programs in the earlier sense.

From knowing to understanding In recent discussions, more stringent requirements have come up concerning knowing a program, sometimes under the heading of *understanding* what one is doing. In addition to propositional knowledge of effects of a plan, or parts of it, another key feature is 'robustness': counterfactually knowing the effects of a plan under changed circumstances, or the ability to modify it as needed. [12] And there are yet other tests of understanding a subject, such as a 'talent for zoom': being able to describe a plan at different levels of detail, moving up or down between grain levels as needed. [13]

Epistemization in general We will not explore these issues further here, except to note that epistemizing algorithms seems to open up a rich and interesting area of investigation. Perhaps the first issue on the agenda here should be to *define epistemization* as a general transformation, or a family of these, on traditional algorithms and specifications, whose properties can then be studied as such. The next general issue would be what happens

[12] Counterfactual robustness under a natural range of deviant circumstances is also well-known from the philosophical literature on definitions of knowledge: see Nozick, Holliday 2012. In that literature, knowledge gets tied to policies for *belief revision* – and it is an intriguing thought that really understanding a program or algorithm might also have to do with agents' *beliefs* about it.

[13] Similar issues arise in analyzing what it means for someone to understand a formal *proof*, and useful intuitions might be drawn from our experience with mathematical practice.

when we systematically epistemize major existing process theories of computation, such as process algebra or game semantics (Bergstra et al., Abramsky). [14] There are a few bits and pieces in the literature, but I am not aware of general results in this spirit. [15]

Finally, it should be pointed out that epistemization is a more general phenomenon than just adding epistemic logic to the world of algorithms. Epistemic logic is one way of modeling knowledge, based, as we saw, on the notion of semantic information. However, various other views of information make sense for logic and computation, including more fine-grained syntactic accounts of information structure as code (van Benthem & Martinez 2007). The issues that we have raised in this section would still make sense then.

7 Interaction and games

The second essential feature of social agency that we mentioned earlier was multi-agent interaction. The typical paradigm for multi-agent action with many-mind knowledge are games, and what we will do know is look at a 'social transformation' of algorithmic tasks that might be called *gamification* (van Benthem 200x).

Multi agent scenarios and knowledge games Reaching a goal and knowing you are there naturally comes with social variants where, say, *others* should not know where you are. In the 'Moscow Puzzle' (van Ditmarsch 2003), two players must inform each other about the cards they have without letting a third party know the solution. More general *knowledge games* of this sort have been studied in Agotnes & van Ditmarsch 2011, Minica 2010. One can think of these as extended semantic explorations of a given epistemic model, assigning different roles to different parties to model more interesting features of inquiry.

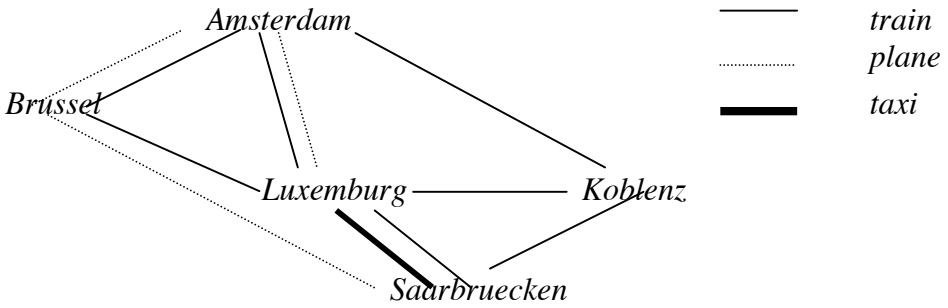
Reachability and sabotage Turning algorithms into games involves prying things apart with roles for different agents. Early examples are *logic games* in the style of Lorenzen, Ehrenfeucht, or Hintikka (cf. the survey in van Benthem 2013), where traditional logical notions now involve a split between a player for truth (proof, analogy, …) versus a player for falsity (countermodel, difference, …). The strategic game-theoretic powers of players in such games provide a more fine-structured analysis of many classical logical notions.

---

[14] Adding epistemic action to process algebra fits its emphasis on communication channels. Explicit epistemics also makes sense with game semantics for programming languages (Abramsky 2008).

[15] The earlier dynamic epistemic logics seem relevant to this enterprise, and so does the literature on computational complexity of epistemic action logics, cf. Halpern & Vardi 1989.

The sabotage game For a more purely algorithmic example, consider again the earlier Graph Reachability, now in a different scenario with two agents. The following picture gives a travel network between two European capitals of logic and computation:



It is easy to plan trips either way. But what if transportation breaks down, and a malevolent Demon can cancel connections, anywhere in the network? At every stage of our trip, let the Demon first take out one connection, while Traveler then follows a remaining link. This turns a one-agent planning problem into a two-player *sabotage game*. Simple game-theoretic reasoning shows that, from Saarbruecken, a German Traveler still has a winning strategy, but in Amsterdam, Demon has the winning strategy against the Dutch Traveler. [16]

Sabotage, logic, and complexity The above suggests a transformation for any algorithmic task to a *sabotage game* with obstructing players. This raises general questions. First, there is logic (van Benthem 2005). One can design languages for these games and players' strategies in terms of 'sabotage modalities' on models with accessibility relations R:

$$\mathbf{M}, s \models \langle\!-\!\rangle \varphi \text{ iff there is a link } (s, t) \text{ in } R \text{ such that } \mathbf{M}[R := R - \{(s, t)\}], s \models \varphi$$

In these unusual modal logics, models change in the process of evaluation, and indeed, one can show that sabotage modal logic, though axiomatizable, is *undecidable*: somehow the computational content of the logic has increased from standard modal logic. Next, there is computational complexity (Rohde 2005). For sabotaged Graph Reachability, the solution complexity of the game jumps from *P-time* for modal model checking to *Pspace*-completeness. This takes a polynomial amount of memory space, like Go or Chess. [17]

---

[16] These games also have interesting interpretations in terms of *learning*, where a Teacher tries to trap a Student into a certain state of knowledge by blocking all escape routes.

[17] Ron van der Meyden, p.c., has pointed out that, while the sabotage game gamifies the original reachability *task*, there is still an additional issue of how the game solution procedure gamifies the original *algorithm* solving the task. Much remains to be understood at this second level.

Still, the game need not always be more complex than the original algorithmic task.

Catch me if you can Now consider another game variant of *GR*. Obstruction could also mean that someone tries to stop me en route: "Starting from an initial position *(**G**, x, y)* with me at *x* and you at *y*, I move first, then you, and so on. I win if I reach my goal region in some finite number of moves without meeting you. You win in all other cases." This game, too, models realistic situations, such as avoiding some people at some receptions. The difference with the Sabotage game is that the graph remains fixed during the game. Sevenster 2006 proves that its computational complexity stays in *P-time*.

Adding knowledge and observation again But it also makes sense to combine all these games with our earlier epistemizations. For instance, sabotage as practiced in warfare involves limited observation and partial knowledge. If we turn algorithms into games of *imperfect information*, solution complexity may increase even further. Jones 1978 gives a classic complexity jump in such a search task. Sevenster 2006 studies a broader array of epistemized gamified algorithms, linked with the '*IF* logic' of Hintikka-Sandu 1997.

Gamification in general The general program behind these examples would be a theory of gamifying algorithmic tasks, and the study of their strategic properties as related to their earlier process properties. We mentioned knowledge games and sabotage games as specific instances – but as we have said, many further examples of successful gamification exist in logic and computer science. [18]A general understanding of this phenomenon might profit from current contacts between logic, computer science, and game theory.

What and how again Some fundamental issues that will play here are related to the central topics of van Benthem 2013. One of these is the transition from logics of programs to *logics of strategies*. But also, an earlier issue that we raised at the beginning of this paper returns. A fundamental question in the logical foundations of game theory is *when two games are the same*. Answers to this question embody a view of a game as an interactive process, and hence, they embody a view of social computation. One persistent intuition here has been the possibility of simulating strategies in other games inside the current one,

---

[18] One should also mention the practical uses of gamification in the world of computer games (**reference**), which seem to have developed very similar aims independently.

sometimes even in the brutal form of copying the same moves. But this computational idea is at the same time an intriguing intuition about the glue of social behavior. [19]

While these issues have pure versions, eventually, we want to look at epistemized ones. This brings us to the theory of *imperfect information games* (Osborne & Rubinstein 1994, Perea 2012) This meshes well with the dynamic epistemic logics that we have mentioned earlier, since they invite explicit analysis of the informational actions taking place during the game. [20] But there is also another dimension to this. Imperfect information games have bona fide solutions in terms of *Nash equilibria in mixed strategies*, letting players play moves with certain probabilities. Thus, perhaps surprisingly, epistemization and gamification may need foundations in terms of mixtures of logic and *probability theory*. [21]

8 Foundations: the three pillars of computation once more

What does the more social perspective on computation sketched here tell us about the original grand questions about computation? We will go in reverse order.

As for the *Turing Test,* the issue of mimicking, or even replacing, humans by machines seems tedious and, despite some unholy attractions, ultimately uninteresting. Given how the world of computation has developed in reality, the real challenge today is understanding the diverse *mixed societies of computers and humans* that have sprung up all around us, and that have vastly increased the behavioral repertoire of humans (and machines).

More tenable today is the original *Church Thesis*. Given the close entanglement of social computation and our use of classical techniques of analysis in logic and complexity theory, we see no need to doubt its 'What' answer: the recursive functions seem fine as the extensional view of what can be computed. But this may be the less interesting question eventually, if one's aim is to understand computation. As we said before, what we really want to understand is the 'How" question of what constitutes computational behavior. And if we take the social perspective of information and interaction outlined here seriously, then some very fundamental questions are on the table: when are two social processes the

[19] Maybe game-theoretic notions of equivalence also have to depend on the *types of agent* playing the games, with their ways of reasoning based on combining belief and preference.

[20] Van Benthem 2013 develops this theme at length under the heading of *'Theory of Play'*.

[21] For quite different epistemic aspects of playing games, in terms of required knowledge of strategies, see van Benthem (to appear, Abramsky paper).

same, and how do we factor in the essential role of the agents involved in them? What we really need is a convincing foundational theory of social behavior, and maybe the focus on computation of this paper will be a good way of making progress here. [22]

Finally, let us return to Turing's original contribution. The *Universal Machine* was, and remains, a crucial device for making our thinking about computation sharp, and allowing, for the first time in history, precise mathematical results on the power and limitations of what is computable. Can there be a similar universal format for the behavior produced by social computation in the sense of this paper? We may need a 'new Turing' for this, but my guess is that the answer will come in the form of an abstract conceptual analysis of what is really means to be a *game* – beyond the details of current game theory.

9 References (to be completed)

Johan van Benthem, *2008*, 'Computation as Conversation', in B. Cooper et al., eds., *New Paradigms, Changing Conceptions of What is Computable*, Springer Science, New York, 35–58. *2011*, *Logical Dynamics of Information and Interaction*, Cambridge University Press, Cambridge. *2013*, *Logic in Games*, The MIT Press, Cambridge Mass. [With Pieter Adriaans], *2008*, *Handbook of the Philosophy of Information*, Elsevier, Amsterdam.

[22] Admittedly, the lack of convergence to a unique view even in the more restricted area of *concurrency* may be a source of worry here. But see Abramsky 2013 for some positive answers.