

# Using Counterfactuals in Knowledge-Based Programming\*

Joseph Y. Halpern<sup>†</sup>  
Cornell University  
Dept. of Computer Science  
Ithaca, NY 14853  
halpern@cs.cornell.edu

Yoram Moses  
Department of Electrical Engineering  
Technion—Israel Institute of Technology  
32000 Haifa, Israel  
moses@ee.technion.ac.il

April 11, 2004

*Knowledge-based programs*, first introduced by Halpern and Fagin [HF89] and further developed by Fagin, Halpern, Moses, and Vardi [FHMV95, FHMV97], are intended to provide a high-level framework for the design and specification of protocols. The idea is that, in knowledge-based programs, there are explicit tests for knowledge. Thus, a knowledge-based program might have the form

**if**  $K(x = 0)$  **then**  $y := y + 1$  **else** skip,

where  $K(x = 0)$  should be read as “you know  $x = 0$ ” and skip is the action of doing nothing. We can informally view this knowledge-based program as saying “if you know that  $x = 0$ , then set  $y$  to  $y + 1$  (otherwise do nothing)”.

Knowledge-based programs are an attempt to capture the intuition that what an agent does depends on what it knows. They have been used successfully in papers such as [DM90, Had87, HMW01, HZ92, ML90, Maz90, MT88, NT93] both to help in the design of new protocols and to clarify the understanding of existing protocols. However, as we show here, there are cases when, used naively, knowledge-based programs

---

\*A preliminary version of this paper appeared in the Proceedings of the Seventh Conference on Theoretical Aspects of Rationality and Knowledge (TARK), 1998. The full version can be found at <http://www.cs.cornell.edu/home/halpern> and will appear in *Distributed Computing*.

<sup>†</sup>Work supported in part by NSF under grant IRI-96-25901, IIS-0090145, and CTC-0208535, by the Air Force Office of Scientific Research under grant F49620-96-1-0323 and F48620-02-1-0101, and by ONR under grants N00014-00-1-03-41, N00014-01-1-0795, and by the DoD Multidisciplinary University Research Initiative (MURI) program administered by the ONR under grant N00014-01-1-0795.

exhibit some quite counterintuitive behavior. We then show how this can be overcome by the use of *counterfactuals* [Lew73, Sta68]. In this introduction, we discuss these issues informally, leaving the formal details to later sections of the paper.

Some counterintuitive aspects of knowledge-based programs can be understood by considering the *bit-transmission problem* from [FHMV95]. In this problem, there are two processes, a *sender*  $S$  and a *receiver*  $R$ , that communicate over a communication line. The sender starts with one bit (either 0 or 1) that it wants to communicate to the receiver. The communication line may be faulty and lose messages in either direction in any given round. That is, there is no guarantee that a message sent by either  $S$  or  $R$  will be received. Because of the uncertainty regarding possible message loss,  $S$  sends the bit to  $R$  in every round, until  $S$  receives an *ack* message from  $R$  acknowledging receipt of the bit.  $R$  starts sending the *ack* message in the round after it receives the bit, and continues to send it repeatedly from then on. The sender  $S$  can be viewed as running the program  $BT_S$ :

**if *recack* then skip else sendbit,**

where *recack* is a proposition that is true if  $S$  has already received an *ack* message from  $R$  and false otherwise, while *sendbit* is the action of sending the bit.<sup>1</sup> Note that  $BT_S$  is a *standard* program—it does not have tests for knowledge. We can capture some of the intuitions behind this program by using knowledge. The sender  $S$  keeps sending the bit until an acknowledgment is received from the receiver  $R$ . Thus, another way to describe the sender’s behavior is to say that  $S$  keeps sending the bit until it *knows* that the bit was received by  $R$ . This behavior can be characterized by the knowledge-based program  $BT'_S$ :

**if  $K_S(\textit{recbit})$  then skip else sendbit,**

where *recbit* is a proposition that is true once  $R$  has received the bit. The advantage of this program over the standard program  $BT_S$  is that it abstracts away the mechanism by which  $S$  learns that the bit was received by  $R$ . For example, if messages from  $S$  to  $R$  are guaranteed to be delivered in the same round in which they are sent, then  $S$  knows that  $R$  received the bit even if  $S$  does not receive an acknowledgment.

We might hope to improve this even further. Consider a system where all messages sent are guaranteed to be delivered, but rather than arriving in one round, they spend exactly five rounds in transit. In such a system, a sender using  $BT_S$  will send the bit 10 times, because it will take 10 rounds to get the receiver’s acknowledgment after the original message is sent. The program  $BT'_S$  is somewhat better; using it  $S$  sends the bit only five times, since after the fifth round,  $S$  will know that  $R$  got his first message. Nevertheless, this seems wasteful. Given that messages are guaranteed to be delivered, it clearly suffices for the sender to send the bit once. Intuitively, the sender should be able to stop sending the message as soon as it knows that the receiver will *eventually* receive a copy of the message; the sender should not have to wait until the receiver *actually* receives it.

It seems that there should be no problem handling this using knowledge-based programs. Let  $\diamond$  be the standard “eventually” operator from temporal logic [MP92];  $\diamond\varphi$

---

<sup>1</sup>Running such a program amounts to performing the statement repeatedly forever.

means that  $\varphi$  is eventually true, and let  $\square$  be its dual, “always”. Now the following knowledge-based program  $\text{BT}_S^*$  for the sender should capture exactly what is required:

**if  $K_S(\diamond \text{recbit})$  then skip else sendbit.**

Unfortunately,  $\text{BT}_S^*$  does not capture our intuitions here. To understand why, consider the sender  $S$ . Should it send the bit in the first round? According to  $\text{BT}_S^*$ , the sender  $S$  should send the bit if  $S$  does not know that  $R$  will eventually receive the bit. But if  $S$  sends the bit, then  $S$  knows that  $R$  will eventually receive it (since messages are guaranteed to be delivered in 5 rounds). Thus,  $S$  should not send the bit. Similar arguments show that  $S$  should not send the bit at any round. On the other hand, if  $S$  never sends the bit, then  $R$  will never receive it and thus  $S$  *should* send the bit! It follows that according to  $\text{BT}_S^*$ ,  $S$  should send the bit exactly if it will never send the bit. Obviously, there is no way  $S$  can follow such a program. Put another way, this program cannot be implemented by a standard program at all. This is certainly not the behavior we would intuitively have expected of  $\text{BT}_S^*$ .<sup>2</sup>

One approach to dealing with this problem is to change the semantics of knowledge-based programs. Inherent in the semantics of knowledge-based programs is the fact that an agent knows what standard protocol she is following. Thus, if the sender is guaranteed to send a message in round two, then she knows at time one that the message will be sent in the following round. Moreover, if communication is reliable, she also knows the message will later be received. If we weaken the semantics of knowledge sufficiently, then this problem disappears. (See [EMM98] for an approach to dealing with the problem addressed in this paper along these lines.) However, it is not yet clear how to make this change and still maintain the attractive features of knowledge-based programs that we discussed earlier.

In this paper we consider another approach to dealing with the problem, based on counterfactuals. Our claim is that the program  $\text{BT}_S^*$  does not adequately capture our intuitions. Rather than saying that  $S$  should stop sending if  $S$  knows that  $R$  will eventually receive the bit, we should, instead, say that  $S$  should stop sending if it knows that *even if  $S$  does not send another message  $R$  will eventually receive the bit*.

How should we capture this? Let  $do(i, a)$  be the formula that is true at a point  $(r, m)$  if process  $i$  performs  $a$  in the next round.<sup>3</sup> The most obvious way to capture “(even) if  $S$  does not send a message then  $R$  will eventually receive the bit” uses standard implication, also known as *material implication* or *material conditional* in philosophical logic:  $do(S, \text{skip}) \Rightarrow \text{recbit}$ . This leads to a program such as  $\text{BT}_S^{\Rightarrow}$ :

**if  $K_S(do(S, \text{skip}) \Rightarrow \diamond \text{recbit})$  then skip else sendbit.**

Unfortunately, this program does not solve our problems. It too is not implementable by a standard program. To see why, suppose that there is some point in the execution of this protocol where  $S$  sends a message. At this point  $S$  knows it is sending a message, so  $S$  knows that  $do(S, \text{skip})$  is false. Thus,  $S$  knows that  $do(S, \text{skip}) \Rightarrow$

<sup>2</sup>While intuitions may, of course, vary, some evidence of the counterintuitive behavior of this program is that it was used in a draft of [FHMV95]; it was several months before we realized its problematic nature.

<sup>3</sup>We assume that round  $m$  takes place between time  $m - 1$  and  $m$ . Thus, the next round after  $(r, m)$  is round  $m + 1$ , which takes place between  $(r, m)$  and  $(r, m + 1)$ .

$\Diamond recbit$  holds. As a result,  $K_S(do(S, skip) \Rightarrow \Diamond recbit)$  is true, so that the test in  $BT_S^{\Rightarrow}$  succeeds. Thus, according to  $BT_S^{\Rightarrow}$ , the sender  $S$  should *not* send a message at this point. On the other hand, if  $S$  *never* sends a message according to the protocol (under any circumstance), then  $S$  knows that it will never send a message (since, after all,  $S$  knows how the protocol works). But in this case,  $S$  knows that the receiver will never receive the bit, so the test fails. Thus, according to  $BT_S^{\Rightarrow}$ , the sender  $S$  should send the message as its first action, this time contradicting the assumption that the message is never sent. Nothing that  $S$  can do is consistent with this program.

The problem here is the use of material implication ( $\Rightarrow$ ). Our intuitions are better captured by using counterfactual implication, which we denote by  $>$ . A statement such as  $\varphi > \psi$  is read “if  $\varphi$  then  $\psi$ ”, just like  $\varphi \Rightarrow \psi$ . However, the semantics of  $>$  is very different from that of  $\Rightarrow$ . The idea, which goes back to Stalnaker [Sta68] and Lewis [Lew73] is that a statement such as  $\varphi > \psi$  is true at a world  $w$  if in the worlds “closest to” or “most like”  $w$  where  $\varphi$  is true,  $\psi$  is also true. This attempts to capture the intuition that the counterfactual statement  $\varphi > \psi$  stands for “if  $\varphi$  were the case, then  $\psi$  would hold”. For example, suppose that we have a wet match and we make a statement such as “if the match were dry then it would light”. Using  $\Rightarrow$ , this statement is trivially true, since the antecedent is false. However, with  $>$ , the situation is not so obvious. We must consider the worlds most like the actual world where the match is in fact dry and decide whether it would light in those worlds. If we think the match is defective for some reason, then even if it were dry, it would not light.

A central issue in the application of counterfactual reasoning to a concrete problem is that we need to specify what the “closest worlds” are. The philosophical literature does not give us any guidance on this point. We present some general approaches for doing so, motivated by our interest in modeling counterfactual reasoning about what would happen if an agent were to deviate from the protocol it is following. We believe that this example can inform similar applications of counterfactual reasoning in other contexts.

There is a subtle technical point that needs to be addressed in order to use counterfactuals in knowledge-based programs. Traditionally, we talk about a knowledge-based program  $Pg_{kb}$  being implemented by a protocol  $P$ . This is the case when the behavior prescribed by  $P$  is in accordance with what  $Pg_{kb}$  specifies. To determine whether  $P$  implements  $Pg_{kb}$ , the knowledge tests (tests for the truth of formulas of the form  $K_i\varphi$ ) in  $Pg_{kb}$  are evaluated with respect to the points appearing in the set of runs of  $P$ . In this system, all the agents know that the properties of  $P$  (e.g. facts like process 1 always sending an acknowledgment after receiving a message from process 2) hold in all runs. But this set of runs does not account for what may happen if (counter to fact) some agents were to deviate from  $P$ . In counterfactual reasoning, we need to evaluate formulas with respect to a larger set of runs that allows for such deviations.

We deal with this problem by evaluating counterfactuals with respect to a system consisting of all possible runs (not just the ones generated by  $P$ ). While working with this larger system enables us to reason about counterfactuals, processes no longer know the properties of  $P$  in this system, since it includes many runs not in  $P$ . In order to deal with this, we add a notion of likelihood to the system using what are called *ranking functions* [Spo88]. Runs generated by  $P$  get rank 0; all other runs get higher rank. (Lower ranks imply greater likelihood.) Ranks let us define a standard notion

of *belief*. Although a process does not *know* that the properties of  $P$  hold, it *believes* that they do. Moreover, when restricted to the set of runs of the original protocol  $P$ , this notion of belief satisfies the knowledge axiom  $B_i\varphi \Rightarrow \varphi$ , and coincides with the notion of knowledge we had in the original system. Thus, when the original protocol is followed, our notion of belief acts essentially like knowledge.

Using the counterfactual operator and this interpretation for belief, we get the program  $\text{BT}_S^>$ :

**if**  $B_S(\text{do}(S, \text{skip}) > \Diamond \text{recbit})$  **then skip else sendbit.**

We show that using counterfactuals in this way has the desired effect here. If message delivery is guaranteed, then after the message has been sent once, under what seems to be the most reasonable interpretation of “the closest world” where the message is not sent, the sender believes that the bit will eventually be received. In particular, in contexts where messages are delivered in five rounds, using  $\text{BT}_S^>$ , the sender will send one message.

As we said, one advantage of  $\text{BT}'_S$  over the standard program  $\text{BT}_S$  is that it abstracts away the mechanism by which  $S$  learns that the bit was received by  $R$ . We can abstract even further. The reason that  $S$  keeps sending the bit to  $R$  is that  $S$  wants  $R$  to know the value of the bit. Thus, intuitively,  $S$  should keep sending the bit until it knows that  $R$  knows its value. Let  $K_R(\text{bit})$  be an abbreviation for  $K_R(\text{bit} = 0) \vee K_R(\text{bit} = 1)$ , so  $K_R(\text{bit})$  is true precisely if  $R$  knows the value of the bit. The sender’s behavior can be characterized by the following knowledge-based program,  $\text{BT}_S^K$ :

**if**  $K_S K_R(\text{bit})$  **then skip else sendbit.**

Clearly when a message stating the value of the bit reaches the receiver,  $K_R(\text{bit})$  holds. But it also holds in other circumstances. If, for example, the  $K_S K_R(\text{bit})$  holds initially, then there is no need to send anything.

As above, it seems more efficient for the sender to stop sending when he knows that the receiver will *eventually* know the value of the bit. This suggests using the following program:

**if**  $K_S(\text{do}(S, \text{skip}) \Rightarrow \Diamond K_R(\text{bit}))$  **then skip else sendbit.**

However, the same reasoning as in the case of  $\text{BT}^>$  shows that this program is not implementable. And, again, using belief and counterfactuals, we can get a program  $\text{BT}_S^{\Diamond B}$  that does work, and uses fewer messages than  $\text{BT}_S^>$ . In fact, the following program does the job:

**if**  $B_S(\text{do}(S, \text{skip}) > \Diamond B_R(\text{bit}))$  **then skip else sendbit,**

except that now we have to take  $B_R(\text{bit})$  to be an abbreviation for  $(\text{bit} = 0 \wedge B_R(\text{bit} = 0)) \vee (\text{bit} = 1 \wedge B_R(\text{bit} = 1))$ . Note that  $K_R(\text{bit})$ , which was defined to be  $K_R(\text{bit} = 0) \vee K_R(\text{bit} = 1)$ , is logically equivalent to  $(\text{bit} = 0 \wedge K_R(\text{bit} = 0)) \vee (\text{bit} = 1 \wedge K_R(\text{bit} = 1))$ , since  $K_R\varphi \Rightarrow \varphi$  is valid for any formula  $\varphi$ . But, in general,  $B_R\varphi \Rightarrow \varphi$  is not valid, so adding the additional conjuncts in the case of belief makes what turns out to be quite an important difference. Intuitively,  $B_R(\text{bit})$  says that  $R$  has correct beliefs about the value of the bit.

In the full paper (which will appear in *Distributed Computing* and is available at <http://www.cs.cornell.edu/home/halpern/papers/tark98.pdf>) we provide a formal model of counterfactuals, and formally analyze the programs  $BT_S^>$  and  $BT_S^{\diamond B}$  in this model, showing that they have the appropriate properties.

## References

- [DM90] C. Dwork and Y. Moses. Knowledge and common knowledge in a Byzantine environment: crash failures. *Information and Computation*, 88(2):156–186, 1990.
- [EMM98] K. Engelhardt, R. van der Meyden, and Y. Moses. Knowledge and the logic of local propositions. In *Theoretical Aspects of Rationality and Knowledge: Proc. Seventh Conference (TARK 1998)*, pages 29–41. 1998.
- [FHMV95] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning about Knowledge*. MIT Press, Cambridge, Mass., 1995.
- [FHMV97] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. Knowledge-based programs. *Distributed Computing*, 10(4):199–225, 1997.
- [Had87] V. Hadzilacos. A knowledge-theoretic analysis of atomic commitment protocols. In *Proc. 6th ACM Symp. on Principles of Database Systems*, pages 129–134, 1987.
- [HF89] J. Y. Halpern and R. Fagin. Modelling knowledge and action in distributed systems. *Distributed Computing*, 3(4):159–179, 1989. A preliminary version appeared in *Proc. 4th ACM Symposium on Principles of Distributed Computing*, 1985, with the title “A formal model of knowledge, action, and communication in distributed systems: preliminary report”.
- [HMW01] J. Y. Halpern, Y. Moses, and O. Waarts. A characterization of eventual Byzantine agreement. *SIAM Journal on Computing*, 31(3):838–865, 2001.
- [HZ92] J. Y. Halpern and L. D. Zuck. A little knowledge goes a long way: knowledge-based derivations and correctness proofs for a family of protocols. *Journal of the ACM*, 39(3):449–478, 1992.
- [Lew73] D. K. Lewis. *Counterfactuals*. Harvard University Press, Cambridge, Mass., 1973.
- [Maz90] M. S. Mazer. A link between knowledge and communication in faulty distributed systems. In *Theoretical Aspects of Reasoning about Knowledge: Proc. Third Conference*, pages 289–304. 1990.
- [ML90] M. S. Mazer and F. H. Lochovsky. Analyzing distributed commitment by reasoning about knowledge. Technical Report CRL 90/10, DEC-CRL, 1990.

- [MP92] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, Berlin/New York, 1992.
- [MT88] Y. Moses and M. R. Tuttle. Programming simultaneous actions using common knowledge. *Algorithmica*, 3:121–169, 1988.
- [NT93] G. Neiger and S. Toueg. Simulating real-time clocks and common knowledge in distributed systems. *Journal of the ACM*, 40(2):334–367, 1993.
- [Spo88] W. Spohn. Ordinal conditional functions: a dynamic theory of epistemic states. In W. Harper and B. Skyrms, editors, *Causation in Decision, Belief Change, and Statistics*, volume 2, pages 105–134. Reidel, Dordrecht, Netherlands, 1988.
- [Sta68] R. C. Stalnaker. A theory of conditionals. In N. Rescher, editor, *Studies in Logical Theory*, American Philosophical Quarterly Monograph Series, No. 2, pages 98–112. Blackwell, Oxford, U.K., 1968. Also appears in W. L. Harper, R. C. Stalnaker and G. Pearce (Eds.), *Ifs*. Dordrecht, Netherlands: Reidel, 1981.