Working Paper Series
ISSN 1170-487X

A Logic for specifying and reasoning about cooperative environments

by Steve Reeves

Working Paper 95/27

August 1995

Department of Computer Science
The University of Waikato
Private Bag 3105
Hamilton, New Zealand

A Logic for Specifying and Reasoning about Cooperative Environments

Steve Reeves

Department of Computer Science
University of Waikato
HAMILTON
New Zealand

stever@waikato.ac.nz

Abstract

In this paper we describe the current progress of an attempt to develop a logic which will allow us to specify required properties of systems which typically consist of a single interactive program being used, probably simultaneously, by several agents, usually people.

The logic is a development of ideas from modal logic and their more recent developments to describe computation. Since modal logic (and its extensions) are still relatively new to most people we give introductions to these logics in this paper, assuming only a familiarity with classical first-order logic and some proof theory.

We also give an account of some of the sorts of situations that we want to specify.

Finally, we consider what work will be needed in the future, building on what we present here, in order to achieve our goal of providing a language in which to specify and reason about systems intended to support co-operative working.

Keywords Formal Specification, Computer supported co-operative working, Modal Logic

1 Introduction

The aim is to develop a logic (i.e. a language and some notion of truth) which will allow us to describe and then reason about systems with several agents able to perform several actions on some shared program.

The final aim, not yet achieved, is to be able to specify and reason about co-operative working, idealized as several agents using a piece of software at the same time. So, their co-operation is manifested in their shared use of a piece of software.

The work presented here is in its initial stages—it is built on solid (and well known) technical grounds—but we need to try out the language and logic as it is described to see what improvements need to be made.

The work on modal logic is well-known and standard; the extensions are the sorts of things that several people have developed, in different guises—Harel's dynamic logic in Harel (1979) and the FOREST group (who were looking at concurrent systems mainly and concentrating on ways to structure specification of such systems in the logic) at Imperial College (using the name modal action logic) are examples, in Ryan, Fiadero and Maibaum (1991). More recently, Duke and Harrison at York have started to apply many of these ideas to interactive systems, though not with the 'action, agent' modalities we consider below—they stick to action modalities, see Duke and Harrison (1995).

The 'action, agent' modalities are, technically, straightforward once the idea of indexing the modalities has been seen. Even so, their use to describe interaction and the possible use of epistemic logic to relate agents are certainly new, though I've no doubt others are thinking about this, given the current fashion for so-called computer support for co-operative working (CSCW). Perhaps, though, the mix of logic and CSCW is not so common.

However, this mixing is a necessary thing if we are to have any hope of software engineering. We need precise, unambiguous and sound ways of describing and reasoning about systems—only on that basis can an implementer, say, know that they have correctly implemented a system, relative to some specification.

2 Background

2.1 Modal Logic

The presentation here is based on the one in Reeves and Clarke (1990).

Modal logic is an extension of classical propositional logic, though we will extend it to first-order logic later. The syntax is extended by adding the modalities $\Box$ and $\Diamond$ to the alphabet and the rule

$$S ::= \Box S \mid \Diamond S$$

to the grammar, where S is a sentence. For any sentence S, the sentence $\Box S$ is understood as (the formal semantics is given a bit later) "S is necessarily true"; $\Diamond S$ is understood as "S is possibly true".

These extra modes of truth (extending the dichotomy of first-order logic), something being necessarily or possibly true, rather than just true, give rise to the name 'modal logic'. (The development of this logic can be told as an interesting story and also raises many technical and philosophical issues that are still not settled today, around 65 years after (modern) modal logic was invented).

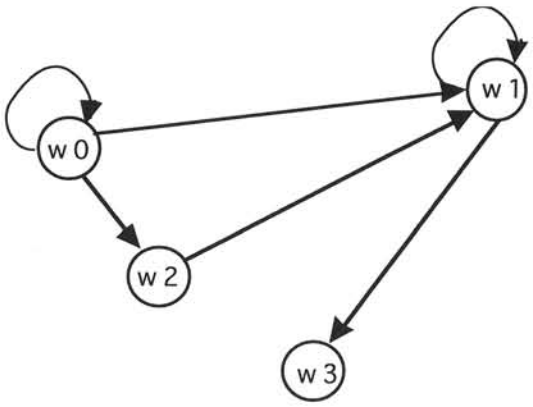
Having extended the syntax, we now have to extend the semantics to match. We use the (now usual) possible-world semantics developed by Kripke in around 1963.

The basis is a set P of possible worlds. These are objects whose only required property is that they are not logically impossible, i.e. not inconsistent. They don't have to be actual or even conceivable worlds, as long as they are not impossible.

Based on P we also have a binary relation $R \subseteq P \times P$ which is called an accessibility relation. It tells us which worlds are accessible from which other worlds. So, given a current world w we can find all the worlds v accessible from w; they are $\{v \mid R(w,v)\}$.

The data P and R go together to form a frame (of reference) $\langle P,R \rangle$.

It is often useful to draw a picture of a given frame—for example



where the arrows occur between v and v' just when R(v,v') for v, v' ∈ P.

Given a frame we can then define the meaning of the usual logical connectives and the modalities. To do this we give truth conditions for non-atomic sentences in terms of their syntactic parts. We use the notation $\models_w S$ to mean that sentence S is true in world w. In the end, for the atomic sentences (i.e. those with no connectives or modalities in) we have to be told explicitly whether they are true or false in a given possible world. For a set L of atomic sentences there is a valuation $v : P \times L \to \{\text{true, false}\}$ so that for each world w and each atomic sentence A, v(w,A) is true or false.

Given all this we have, for any sentences S and T

$\models_w S \wedge T$ iff $\models_w S$ and $\models_w T$

$\models_w S \vee T$ iff $\models_w S$ or $\models_w T$

$\models_w S \to T$ iff not $\models_w S$ or $\models_w T$

$\models_w \neg S$ iff not $\models_w S$

$\models_w \Box S$ iff for all $v \in P$ such that $R(w,v)$ we have $\models_v S$

$\models_w \Diamond S$ iff there is some $v \in P$ such that $R(w,v)$ and $\models_v S$

So, to say S is necessarily true in some world w is to say that S is true in any world we can get to from w. To say S is possibly true is to say that there is at least one world we can get to from w where S is true. Note that it follows from these definitions that any occurrence of $\Diamond S$ can be replaced by $\neg\Box\neg\Box S$, for any sentence S.

We are now in a position to prove things since we have a language and a notion of truth. We can develop proof rules (and implement a proof assistant to help us use the rules correctly) or we can develop theorem-provers which use the notion of truth above to try to decide whether conjectured propositions in our language are true. Both these activities are current research goals in their own rights and we'll not consider them further here.

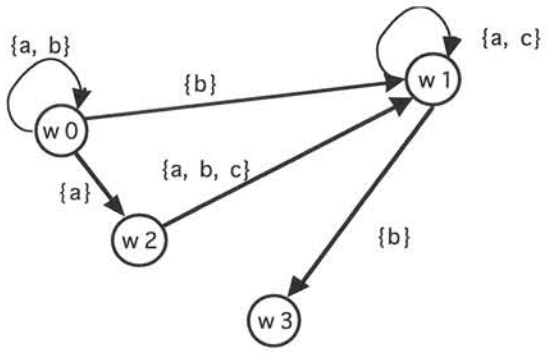
What we will do is to extend the language once more, getting closer to our aim.

2.2 Modal Logic extended

We extend the syntax of the language by indexing the modalities with actions. (As with all of this development, we are currently keeping things very general, so this idea of action will be left uninterpreted, though the name should suggest what we expect to find as an action.) We assume that there is a set Act of actions and write [a]S to mean 'after action a has successfully been completed, S is true'.

In order to give a semantics to this we need a more sophisticated frame. Instead of having a single accessibility relation, the frame now has a family of them, indexed by elements of Act. The frame is now $\langle P, \{R_i\}_{i \in Act} \rangle$ where $R_i \subseteq P \times P$ for $i \in Act$.

A picture of this might be



Here a, b, c ∈ Act. An arrow between w and v (both elements of P) is marked with i ∈ Act iff $R_i(w,v)$.

The semantics for the non-modal parts of the language extends with no changes. For the indexed modality above we have

$\models_w [a]S$ iff for all $v \in P$ such that $R_a(w,v)$ we have $\models_v S$

which is to say that [a]S is true in world w if and only if in all worlds v accessible due to the action a we have S true in v.

2.3 Quantifying

Our first step in introducing the quantifiers is the simplest since it only quantifies over individuals, not over the actions that index the modalities.

We assume that there is a set of individuals over which we want to quantify.

First, the frame is extended to a triple by adding a function $I : P \rightarrow 2^D$, where D is the set of individuals (which in our case is going to contain things like numbers, characters, lists, buttons etc.). This function I simply tells us which individuals exist in which worlds, i.e. given some world w, I(w) is the set of all the individuals that exist there.

The language is extended to include the quantifiers and variables as usual.

Given this, we have the following

$\models_w \forall xS$ iff $\models_w S(x/b)$ for all $b \in I(w)$

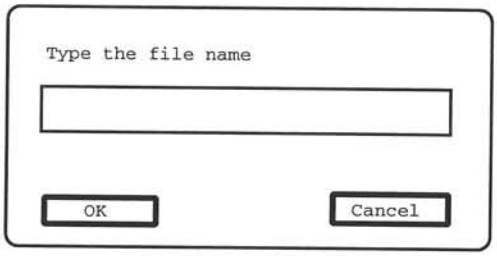
(where S(x/b) is S with all free occurrences of x replaced by b).

3 Application

Having described the language and logic we'll now look at some simple examples of its use.

Since we're interested in trying to describe interactive systems of a certain sort (ultimately those where several people interact with a program) we'll concentrate our examples on that sort of thing. Clearly, the framework above is very general and could be used in many specific cases. To make it more particular, we now consider the set of possible worlds to be a set of states in which the execution of the program that is being shared can be.

Consider the program when it displays a dialogue box:



Here are two things we might want to have as part of the statement of the specification of the program that, on execution, displays this box:

> Pressing "cancel" opens no more files

We could represent this by

$\models$ select("Cancel") →
  ∀l(open(l) → [push]open(l))

assuming that we have other facts that say that the predicate 'select' is true just when its argument is the label on the area within which the mouse cursor is currently positioned, 'open' is a predicate which is true just when its argument is a list of the names of all the files currently open and 'push' is the action of pressing the mouse button.

> Pressing "OK" keeps open all the files that were open and in addition opens the one whose name was typed in the text field if it is the name of some existing file

given by

$\models$ select("OK") →
  ∀x(text(x) → name(x) →
    ∀l(open(l) → [push]open(x:l)))

assuming the definition as above and also that 'text' is a predicate which is true just when its argument is a piece of text (a string, perhaps), 'name' is a predicate which is true just when its argument is the name of a file which exists in the current environment and ':' adds an element to the front of a list.

So far, the logic that we have described is fairly standard and has been developed by many people (Goldblatt (1987) is a standard text on this). The application to which we are putting it is new, though has similarities, in that it talks about interactive systems, to the work of Duke and Harrison (1995).

4 Further extensions to the language

We still aren't quite where we want to be. The language as it stands does not allow us to talk explicitly about any agents that might be around—so far we can give properties of parts of the system (as in the examples above) but we cannot yet denote the agents which cause, say, a button to be pressed or some text to be typed.

Given our aim of specifying and reasoning about systems in which several agents co-operate in using, this is clearly important.

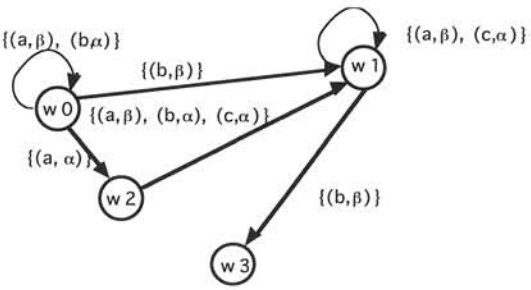
In order to do this we extend the language further by first extending the syntax to

$$[a,\alpha]S$$

where a is an action, just as before, and α is an agent, an element of the set Agt. The intended meaning is that when action a is successfully completed by agent α then S is true.

To give the semantics we extend the idea used in the previous extension and index the family of accessibility relations with action and agent, so the frame is now $\langle P, \{R_{i,\iota}\}_{i \in Act, \iota \in Agt}, I\rangle$ where $R_{i,\iota} \subseteq P \times P$ for $i \in Act$, $\iota \in Agt$. Now, the accessibility relation tells us what worlds are accessible from what other worlds if a is done by α.

A picture of the frame now would be



As before, all the non-modal parts of the language extend without change and we have

$\models_w [a,\alpha]S$ iff for all $v \in P$ such that $R_{a,\alpha}(w,v)$ we have $\models_v S$

We can also extend the language to allow both actions and agents to be quantified over. First, we have to extend the function I. Currently it maps worlds to sets of individuals. If we are to quantify over elements of Act and Agt we need I to be indexed on the sort of set it should deliver. So, we have

$I_{ind} : P \to 2^D$, as I was before

$I_{act} : P \to 2^{Act}$, takes a possible world to the actions which can be done there, so

$$I_{act}(w) = \{ a \mid \exists \alpha \in Act.\ \exists v \in P.\ R_{a,\alpha}(w,v)\}$$

$I_{agt} : P \to 2^{Agt}$, takes a possible world to the agents that can perform there, so

$$I_{agt}(w) = \{ \alpha \mid \exists a \in Agt.\ \exists v \in P.\ R_{a,\alpha}(w,v)\}$$

Then we can define the quantifiers over actions and agents in the obvious way:

$\models_w \forall x S$ iff for all $a \in I_{act}(w)$ we have $\models_w S(x/a)$

$\models_w \forall \delta S$ iff for all $\alpha \in I_{agt}(w)$ we have $\models_w S(\delta/\alpha)$

and a frame is now the triple

$$\langle P, \{R_{i,\iota}\}_{i \in Act, \iota \in Agt}, \langle I_{ind}, I_{act}, I_{agt}\rangle\rangle.$$

5 Deontics

When we consider systems with several agents we usually need to be able to talk about whether actions are allowed at certain stages, for example we want to ensure that only one agent at a time can press a button.

Logics which talk about permission and obligation are known as deontic logics. There are many ways to define such a logic (indeed using a modal logic where $\Box$ and $\Diamond$ stand for obligation and permission is a time-honoured area of research). Here, since we already have some of the framework in place we can use a fairly simple mechanism to allow some deontic features to appear in the logic.

First, we have two binary predicates can and must, whose first arguments are from Act and second from Agt. The intended meaning is that can(a,α) is true just when α is permitted to carry out action a; must(a,α) is true just when α is obliged to carry out a.

Formally, we give the semantics by first having two functions

$$\Pi : Act \times Agt \to 2^P$$

$$O : Act \times Agt \to 2^P$$

Π, given action a and agent α, has a value the set of possible worlds in which α is permitted to carry out a. Similarly, O has as value the set of possible worlds in which α is obliged to carry out a.

Given this we then have

$\models_w can(a,\alpha)$ iff $w \in \Pi(a,\alpha)$

$\models_w$ must(a,α) iff w $\in$ O(a,α)

6 More Applications

A first example considers describing a system after initialization (perhaps after the 'init' button has been pressed):

$\models \forall\delta$[init,δ]Initialized

$\models$ Initialized $\leftrightarrow$ ($\forall$v(var(v) $\rightarrow$ v = 0) $\wedge$
$\forall x\forall\alpha$(can(x, α) $\wedge$ .....
)

Next, some facts which describe how buttons work:

$\forall$x(button(x) $\rightarrow$
$\forall\alpha(\exists\beta(\alpha \neq \beta \wedge$ can(click(x),β)) $\rightarrow$
$\neg$can(click(x),α)))

for any button x, for any agent α, if there is an agent β different from α and β is permitted to click x then no α can

$\forall$x(button(x) $\rightarrow$
$\forall\alpha$((can(click(x),α) $\wedge$ enabled(x)) $\rightarrow$
[click(x),α]$\neg$enabled(x)))

for any button x, for any agent α, if α is permitted to click the enabled button x then if α clicks x successfully, x will then be disabled

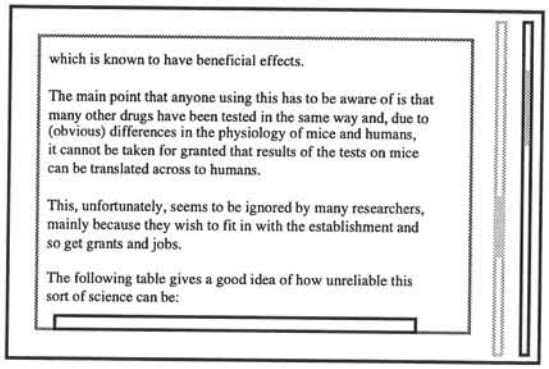
Finally, consider the dialogue box from above:

$\forall\alpha\forall$b($\exists\beta(\alpha \neq \beta \wedge$ button(b) $\wedge$ select(β,b)) $\rightarrow$
$\neg$select(α,b))

for any agent α and button b, if there is an agent β different from α and β has selected b then no other agent α can select b

7 A Larger Example

Imagine that we have a group editor. What properties might we want to specify for it? One technique that has been experimented with is that of multiple scrollbars. They can be used (by the owner) in the usual ways to move through a file. They can also be used (by everyone else) as a way of seeing where in a file other people are working. For example:



In this picture, the scrollbar to the far right is mine, so the dark rectangle within it corresponds to the piece of text that I am viewing (and which is shown). The other scrollbar belongs to the other user (there will be one for each user, so there are only two in this case) and, similarly, shows where in the file they are working. This darkened rectangle corresponds to the predicate 'viewing' below.

Let scrollbar(s,α) be true when s is the scollbar belonging to α. The basic fact about ownership of a scrollbar is given by

$\models \forall\alpha\forall$s(scrollbar(s,$\alpha$) $\leftrightarrow$
(can(up(s),α) $\wedge$ can(down(s),α)) (SB1)

Let viewing(α,top,bottom) be true when α can see lines numbered top to bottom.

We can give properties of the actions up and down by

$\models \forall s\forall\alpha\forall t\forall$b((can(down(s),$\alpha$) $\wedge$
viewing(α,t,b))
$\rightarrow$ [down(s),α]viewing(α,t+10,b+10))
(DN1)

$\models \forall s\forall\alpha\forall t\forall$b((can(up(s),$\alpha$) $\wedge$ viewing(α,t,b))
$\rightarrow$ [up(s),α]viewing(α,t-10,b-10))
(UP1)

Now let Steve launch the group editor on the file document X. We have the most basic fact that Steve is 'present' and working on the file, recorded by

$\models_{w0}$ present(Steve)

for some world w0. Following our discussion above we also want, as a basic, fact, that

$\models \forall\alpha$(present(α) $\rightarrow \exists$sscrollbar(s,α))

from which it follows that

$\models_{w0}$ scrollbar(s1,Steve)

for some scrollbar called s1.

We also expect the scrollbar to be positioned so that a standard amount (say the first 20 lines) can be seen:

$\models_{w0}$ viewing(Steve, 0, 20)

We can show that

$\models_{w0}$ can(down(s1), Steve)

SB1 gives us

$\models_{w0} \forall\alpha\forall s(\text{scrollbar}(s,\alpha) \leftrightarrow (\text{can}(\text{up}(s),\alpha) \wedge \text{can}(\text{down}(s),\alpha)))$

so

$\models_{w0}$ scrollbar(s1,Steve) $\leftrightarrow$ (can(up(s1),Steve) $\wedge$ can(down(s1),Steve))

so since

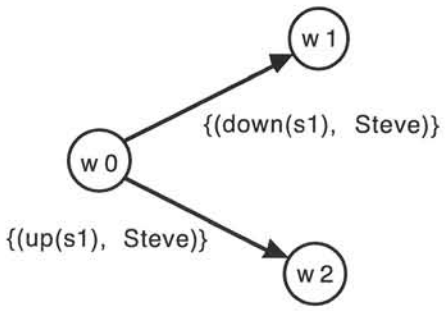
$\models_{w0}$ scrollbar(s1,Steve)

we have

$\models_{w0}$ can(up(s1),Steve) $\wedge$ can(down(s1), Steve)

and hence the required conclusion.

The frame we have been using looks, in part, like



and we also know that

w0 $\in \Pi$(down(s1), Steve)

and

w0 $\in \Pi$(up(s1), Steve)

Steve now pushes the down-arrow on his scrollbar, which is possible since

$\models_{w0}$ can(down(s1), Steve)

and

$\models_{w0}$ viewing(Steve, 0, 20)

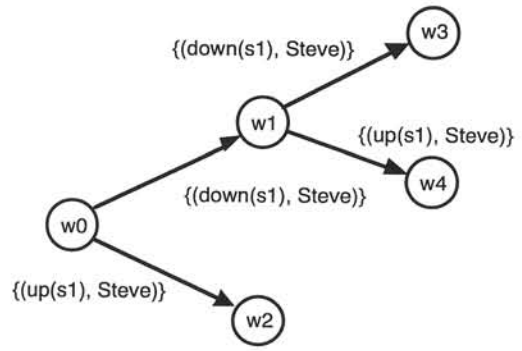
so we have, from DN1, that

$\models_{w0}$ [down(s1),Steve]viewing(Steve,10,30)

and so

$\models_{w1}$ viewing(Steve, 10, 30)

Considering DN1 again shows us that the frame looks like



and Steve could again press his down arrow, giving

$\models_{w3}$ viewing(Steve, 20 40)

If someone else now joins in we have

$\models_{w1}$ scrollbar(s2,Chris)

and

$\models_{w1}$ viewing(Chris,0,20)

and the frame will now look like figure one.

We would also expect to have other general facts, like

$\forall\alpha\forall s\forall a((\text{scrollbar}(s,\alpha) \wedge a \neq \text{close}) \rightarrow [a,\alpha]\text{scrollbar}(s,\alpha))$

$\forall\alpha\forall s(\text{can}(\text{close},\alpha) \leftrightarrow \text{scrollbar}(s,\alpha))$

$\forall\alpha(\text{can}(\text{close},\alpha) \rightarrow [\text{close},\alpha]\neg\exists s\ \text{scrollbar}(s,\alpha))$

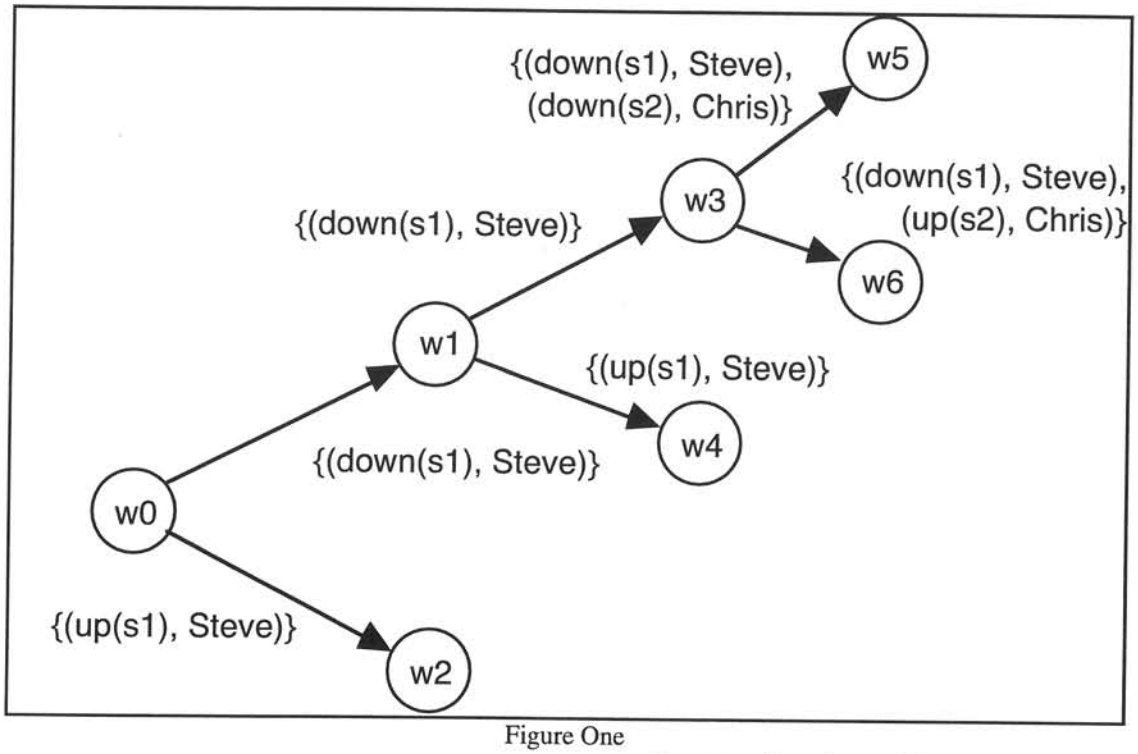


Figure One

8 People or Processes?

The examples above show that the logic we have is an expressive and, with practice, convenient way of describing how the program reacts to people using the actions that it supports. However, this is an almost entirely program-centred point-of-view and can be seen as nothing more than a specialization of the idea that people are just other programs and so the whole example reduces to a network of concurrent computations.

What, then, makes the above example (and all CSCW examples) more than 'just' a network of concurrent processes? To answer this question, we turn to the current 'buzz-phrase' for this area of work—computer supported cooperative working. The point is that people engage in this sort of work because they wish to cooperate over some task. When cooperating, people clearly need to form models of their colleagues' knowledge of the task. They do this partly through their own basic knowledge but they also use knowledge they derive from the program itself; for example, the existence of a scrollbar tells Steve that Chris is now working on this document and the position of Chris's view becomes known to Steve when he looks at the position of the shaded part of Chris's scrollbar.

So, we contend that knowledge is what people have that makes them different from the program they are sharing. (Of course, they also have intelligence, motivation, intention and other properties that differentiate them from the program, but knowledge seems to be enough for now.)

Our plan, then, is to add a way of describing knowledge within the specification of the system. To do this we first need a language and logic for knowledge.

In order to add the ability to talk about knowledge in our specifications we can carry-on in the modal tradition and use the classic example of S4.

This is a logic which, in terms of its semantics, simply has a transitive and reflexive accessibility relation over a set of possible worlds that model states of knowledge. Necessity is interpreted as 'is known', so a sentence of the form $\Box S$ is read as 'S is known'.

It turns out, remarkably, that the various properties of the accessibility relation for modal logics can be exactly captured by a few simple axioms. In the case of S4 the following is a sound and complete axiomatization which exactly expresses the semantics which are captured by a reflexive and transitive accessibility relation:

Any axiom of classical propositional logic

$\vdash \Box S \rightarrow S$

$\vdash \Box S \rightarrow \Box\Box S$

$\vdash \Box(S \rightarrow T) \rightarrow (\Box S \rightarrow \Box T)$

(where S and T are any sentences) together with the rules modus ponens and, again where S is any sentence,

$$\frac{\vdash S}{\vdash \Box S} \text{ (K)}$$

The first axiom is there so that we have classical propositional logic to build on. The second characterizes an important property of knowledge: if S is known then S is true. Contrast this with belief, where something can be believed without it being true. We view knowledge as 'true belief'.

The third axiom says that if something is known that it is known that it is known. The last says that if you know S implies T and you know S then you know T. The rule K says that the axioms are known.

Since we want to specify situations in which there are several people working together, we clearly need to be able to attribute what is known to the knower, rather than having a big pool of knowledge with no differentiation as to who knows what.

To do this we can use the same device as we did for incorporating actions and agents, namely we index the modality which means indexing the accessibility relation that moves us between states of knowledge. In order to make clear that we are dealing with a different set of possible worlds here we use the notation K_α, where α is some agent, instead of $[\alpha]$.

In order that we can reason about what the agents know about the system, we must allow the atomic sentences which are true in each state underlying the action and agent logic to be available for use in the knowledge logic.

Note that each possible world or computational state in the action and agent logic has associated with it a set of atomic sentences which describe it, e.g. scrollbar(s1, Steve. It is these sentences which form the link between the two logics and the basis for the knowledge logic since it is the propositions described by these sentences which are the elements of the possible worlds in the knowledge logic.

Given this new language we can now go on to specify some properties of our editor which make is clear that we are talking about a system where people are cooperating.

A very basic requirement would be that

$$\forall\alpha(\text{present}(\alpha) \rightarrow \forall\beta(\text{present}(\beta) \rightarrow K_\beta\text{present}(\alpha)))$$

i.e. that for any people present, they know who else is present. Another would be that

$$\forall\alpha\forall\beta\forall x\forall y(K_\alpha\text{viewing}(\beta,x,y))$$

i.e. everyone can tell where in the file anyone else is looking.

The designer's job is to specify a system where these sorts of requirements are met, i.e. they must specify that certain facts hold so that the above requirements are provable.

So, we now have two logics, connected by the facts that appear in each state of the underlying computation. This pair of logics and the associated language is what we view as the specification language and logic for specifying and reasoning about a CSCW system.

9 Problems

There are several problems that need to be resolved.

Since we have blithely allowed equality into our logic, we have the standard problem of naming things. The problem arises when some object is known by two names and the classic example is of the planet Venus, which is known both as 'The morning star' and 'The evening star'. If we say that some person α knows 'The morning star' has some property P, i.e.

$$K_\alpha P(\text{The morning star})$$

then it is not necessarily the case that

$$K_\alpha P(\text{The evening star}) \qquad (*)$$

(since they may not know about the alias) but since

$$\text{The morning star} = \text{The evening star}$$

our logic leads to the conclusion (*) just by replacement of equals for equals.

It may be that since everything we are likely to name will have just one name (the one that the program gives it internally), this problem will not arise for us, but it seems prudent to be aware of its potential.

We have a similar problem that arises because certain facts (for example, what size agent α's window is currently) will be known to everyone, according to our logic, because of the rule K and the fact that we need these basic sorts of fact to provide a connection between the two logics.

An obvious solution here is to denote certain facts as private to each agent so that they don't unnecessarily find their way into everyone's knowledge.

One possible solution to many of our current problems, including those mentioned above, is to move to a language and logic like that proposed by Moore (1985) in the context of a logic for reasoning about knowledge and action in planning systems.

He views the sentence $\vdash_w K_\alpha P$ as meaning that P is true in all worlds that are compatible with α's knowledge in w. An action, then, is something which modifies knowledge (and so affects what possible-worlds are compatible).

His logic solves many questions, like the Venus one, particularly elegantly and so is very

attractive. However, it is not clear how it might be adapted to work in our case since we want have states of a computation as the basis for both our logics and Moore's notion of compatibility does not seem easily to cover this. We need to do more work on this though.

Another place to look for solutions to some of our technical problems is the property theory of Turner (1990).

10 Future work

Having introduced a language and a logic for it for it, there are several obvious next steps. A few are:

get more examples of, and reflect further on, CSCW-style interactions and see how well they can be formalized in the language; use this experience to correct mistakes and make the language more suitable for this task;

develop some rules of proof which correctly capture the notion of truth that we have given here (these will probably be natural deduction style rules, in keeping with now commonly accepted practice);

embody the proof rules in a program which will act as a proof assistant, i.e. a program which will only allow the rules to be used correctly (so anything proved is known to be correctly proved) and which will allow the recording, naming and storage of definitions and theorems, which allow better structuring and presentation of proofs;

embody the semantics rules above in a tableau-based theorem-prover, say, which will then allow us to answer questions about the consistency of descriptions of the sorts of examples we've given here.

More philosophical questions for the future include:

should we, or do we need to, consider allowing explicit temporal notions into the language;

some ideas like trust between agents and how that affects interaction are beginning to be used CSCW research—consider how this sort of relationship might be formalized.

References

Duke, D.J. and Harrison, M.D. (1995) Interaction and Task Requirements, in proceedings of the 2nd Eurographics Workshop on Design, Specification and Verification of Interactive Systems, Bastide and Palanque (eds.), Bonas, France.

Goldblatt, R., (1987) *Logics of Time and Computation*, CSLI Lecture Notes, Number 7.

Harel, D. (1979) *First-order dynamic logic.*, Lecture notes in Computer Science 69, Springer-Verlag.

Moore, R. C. (1984) A Formal Theory of Knowledge and Action, in J.R. Hobbs and R.C. Moore (eds.), *Formal Theories of the Commonsense World*, Ablex, pp. 319-358.

Reeves, S. and Clarke. M (1990) *Logic for Computer Science*, Addison-Wesley.

Ryan, M., Fiadeiro, J. and Maibaum, T. (1991) Sharing Actions and Attributes in Modal Action Logic, in Ito and Meyer (eds.), *Theoretical Aspects of Computer Software*, Lecture notes in Computer Science 526, Springer-Verlag.

Turner, R (1990) *Truth and Modality for Knowledge Representation*, Pitman.