Runs and Systems

It is hard enough to reason about the behaviour of one agent. Things get much worse if we have a system of interacting agents.

Consider, for example the muddy children puzzle. If we attempt to model the system in full detail, we would have to include all that happend to each of the children thoughout their lives, a detailed description of their visual and auditory systems and how they operate, details of the whether they can reason logically, etc. etc. The list is potentially endless. All of these factors could, in principle, influence the behaviour of the children.

To cope with the complexity,

first we focus attention on only a few details, and *hope* that these cover everything that is relevant to our analysis.

• ***second*** we try to find good ways to think about a situation in order to minimize the complexity.

We want to show that reasoning about systems in terms of knowledge can be very helpful.

To do that, we need a formal model of multi-agent systems.

One of a key assumptions we make is that if we look at the system at any point in time, each agent is in ***some state.***

We refer to this as the agent‘s ***local*** state.

We assume that an agent‘s local state encapsulates all the information to which the agent has access.

In our abstract framework, we do not make any additional assumptions about the state.

In case of the muddy children, the state of a child might encode what the child has seen and heard i.e. which of the other children have muddy foreheads and which do not, the father‘s initial statement, and the responses of each of the children to the father‘s questions so far.

Modelling a poker game a player's state might consist of the cards he currently holds, the bets made by the other players, any other cards he has seen, and any information he may have about the strategies of the othe player's. [e.g. Bob may know that Alice likes to bluff, while Charlie tends to bet conservatively.]

As the example of a poker game already indicates, representing the states of the agents can be highly nontrivial.

The first problem is deciding what to include in the state.

Certainly if the system is made up of interacting people, then it becomes a rather difficult problem to decide where to draw the line. In the poker example, should we include the fact that Bob had an unhappy childhood as a part of the state? If so, how do we capture this?

Once we have solved the problem of what we include in the state, we then have to decide how to *represent* what we do include.

If we decide that Bob's childhood is relevant, how do we describe the relevant features of his childhood in a reasonable way? In our abstract framework we sidestep these difficulties, and simply assume that at each point in time , each agent in the system is in some unique state. Of course, we are confronted with these difficulties when dealing with concrete examples.

These problems tend to be somewhat easier to solve when dealing with processes in a distributed system rather than with people, but as we shall see later on, even in tnis simpler setting there can be difficult choices to make.

Once we think in terms of each agent having a state, it is but a short step to think of the whole system as being in some state.

The first try might be to make the system's state to be a tuple of the form $(s_1, s_2, \ldots, s_n)$ where s_i, is the agent i' s state. But, in general, more than just the local states of the agents may be relevant when analyzing a system.

If we are analyzing a message-passing system where processes send messages back and forth along communicaton lines, we might want to know about messages that are in transit or about whether a communication line is up or down.

If we are considering a system of sensors observing some terrain, we might need to include features of the terrain in a description of the state of a system.

Motivated by these observations, we conceptually divide a system into two components: the agents and the *environment,* where we view the environment as „everything else to be relevant".

In many ways the environment can be viewed as just another agent, though it typically plays a special role in our analyses.

We define a *global state* of a system with n agents to be an $(n + 1)$-tuple of the form $(s_e, s_2, \ldots, s_n)$, where s_e is the state of the environment and s_i is the local state of agent i.

A given system can be modeled in many ways. How we divide the system into agents and environment depends on the system being analyzed.

In a message-passing system, we can view a message buffer, which stores messages not yet delivered, either as a process (i.e., an agent), and have its state encode which messages have been sent and not yet delivered, or as a part of the environment.

Similarly, we can view a communication line as an agent whose local state might describe (among other things) whether or not it is up, or we can have the status of the communication lines be a part of the environment.

A global state describes the system at a given point of time. But a system is not a static entity; it constantly changes.

• Since we are mainly interested in how systes change over time, we need to build time into our model.

• We define a *run* to be a function from time to global states.

Intuitively, a run is a complete description how the system‘s global state evolves over time.

We take time to range over the natural numbers. Thus $r(0)$ describes the initial global state of the system in a possible execution r, the next global state being $r(1)$, and so on.

Our assumption that time ranges over the natural numbers seems to be quite a strong one. In particular it means that the steps are discrete and that time is infinite. We have made this choice mainly for definitness, but also because it seems appropriate for many applicationas.

Most of our results and comments hold with little or no change if we assume instead that time is continuous (and ranges over, say, the real numbers or the non negative real numbers).

Although we typically think of time as being continuous, assuming that time is discrete is quite natural. Computers proceed in discrete time steps, after all.

Allowing time to be infinite makes it easier to model situations where there is no *a priori* time bound on how long the system will run. An example of this phenomenon is provided by the Muddy children puzzle. It is not clear how many steps it will take the childrens to figure out whether they have mud on their forehead; indeed, in some variants of the puzzle, they never figure it out.

And if we do want to model a system that runs for a bounded number of steps, we can typically capture this by assuming that the system remains in the same global state after it has stopped.

We assume that time is measured on some clock external to the system. We do *not* assume that agents in the system necessarily have access to this clock; at time m measured on the external clock, agent i need not know it is time m. If an agent does know the time, then this information would be encoded in his local state.

This external clock need not measure „real time“.

For example, in the case of Muddy children puzzle, there could be one „tick“of the clock for every round by the puzzle and every round of the answers to the father‘s question.

If we are analyzing a poker game, there could be on tick of the clock in whatever each time someone bets or discards. In general, we model the external clock in whatever way makes it easiest for us to analyze the system.

A system can have many possible runs, since the system‘s global state can evolve in many possible ways: there are a number of possible initial states and many things that could happen from each initial global state.

For example, in poker game, the initial global state could describe the possible deals of the hand, with player i‘s local state s_i describing the cards held initially by player i.

For each fixed deal of the cards, there may still be many possible betting (and discarding) sequences, and thus many many runs.

In a message-passing system, a particular message may or may not be lost, so again, even with fixed initial global state, there are many possible runs.

To capture this, we formally define a *system* to be a *nonempty set of runs.*

Notice how this definition abstracts our intuitive view of a system as a collection of interacting agents.

Instead of trying to model to describe the system directly, our definition models the possible *behaviours* of the system. By the requirement that the set of runs be nonempty we are modelling that the system has *some* behaviours.

Our approach uses the same formal systems of great diversity; a computer system and a poker game are modeled similarly.

We will use the term *system* in two ways: as the „real life" collection of interacting agents or as a set of runs. Our intention should be clear from the context.

We proceed as follows:

let L_e be the set of all possible states for the environment

L_i be the set of all possible states for agent $i, i = 1, 2, \ldots, n$

$\mathcal{G} = L_e \times L_1 \times \ldots \times L_n$ set of all possible global states

A r*un over* $\mathcal{G}$ is a function from the domain of time to $\mathcal{G}$.

Thus a run over $\mathcal{G}$ can be identified with sequence of global states in $\mathcal{G}$.

We refer to a pair (r, m) consisting of a run r and time m as a *point.*

If $r(m) = (s_e, s_1, \ldots, s_n)$ is the global state in the point (r, m), we define

$$r_e(m) = s_e \quad \text{and} \quad r_i(m) = s_i \quad \text{pro } i = 1, \ldots, n$$

Thus, $r_i(m)$ is agent's i local state at the point (r, m).

A *round* takes place between two time points. We define round m in run r to take place between time m - 1 and time m.

It is often convenient to view an agent as performing an *action* during a round.

A *system* R *over* $\mathcal{G}$ is a set of runs over $\mathcal{G}$. We say that (r, m) is *a point in system* R if $r \varepsilon R$.

The following simple example describes a scenario that we call the *bit-transmission problem,* which should give one a better feeling for some of the above definitions.

Example 1. *Bit-transmition problem.*

Imagine, we have two processes, say 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.

Unfortunately the communication line is faulty, and it may lose messages in either direction in any given round; i.e. there is no guarantee that a message sent by either S or R will be received.

For simplicity, we assume that a message is either received or lost in the same round it is sent, or lost altogether.

Since in this example a message may be received in the same round it is sent, we are implicitely assuming that rounds are long enough for a message so be sent and delivered.

We are assuming that this type of message loss is the only possible faulty behaviour in the system.

Because of uncertainty regarding possible message loss, S sends the bit to R in every round, until S receives a message from R acknowledging receipt of the bit. We call this message from R an *ack* message.

R starts sending the *ack* message in the round after it receives the bit.

To allow S to stop sending the bit, R continues to send the *ack* repeatedly from then on.

This informal description gives what we call a *protocol* for S and R : it is a specification what they do at each step.

The protocol dictates that S must continue sending the bit to R until S receives the *ack* message; roughly speaking, this is because before it receives the *ack* message, S does not know whether R received the bit.

On the other hand, R never knows for certain that S actually received its acknowledgment. Note the usage of the word „know" in the two previous sentences. This of course is not an accident. We claim that this type of protocol is thouht in terms of knowledge.

Returning to the protocol, note that R does know perfectly well that S stops sending messages after receiving an *ack* message. But even if R does not receive messages from S for a while, form R's point of view this is not necessarily because S received an *ack* message from R ; this could be because the messages that S has sent were lost in the communication channel.

We could have S send an *ack-ack* message - an acknowledgment to acknowledgment - so that R could stop sending the acknowledgment once it receives an *ack-ack* message from S. But this only pushes the problem up one level: S will not be able to safely stop sending *ack-ack* messages, since S has no way of knowing that R has received an *ack-ack message.* We shall show later on that this type of uncertainty is inherent in systems such as the one we have just described, where communication is not guaranteed.

For now, we focus on the protocol that we described, where R continues to send *ack* messages in every round, and S stops as soon as it receives one of them.

The situation that we have just described in formally can be formalized as a system. To describe the set of runs that make up this system, we must make a number of choices regarding how to model the local states of S , R , and the environment.

It seems reasonable to assume that the value of the bit should be part of S's local state, and it should be part of R's local state as soon as R receives a message from S with the value.

Should we include in S's state the number of times that S has sent the bit or the number of times that S receives an *ack* message from R? Similarly should we include in R's state the number of times R has sent the *ack* message or the number of times R has received the bit from S?

Perhaps we should include in the local state representation of the protocol being used ?

Our choice is to have the local states of S and R include very little information; essentially, just enough to allow us to carry out our analysis.

On the other hand, as we shall see in Example 2 , it is useful to have the environment's state record the events taking place in the system.

Thus we take L_S, the possible local states of S, to be {0, 1, (0, *ack*), (1, *ack*)}, where intuitively, S's local state is k if its initial bit is k and it has not received an *ack* message from R, while S's local state is (k, ack) if its initial bit is k and it has received an *ack* message from R, for $k = 0, 1$.

Similarly, $L_R = \{\lambda, 0, 1\}$, where λ denotes the local state where R has received no meassage from S, and k denotes the local state where R received the message k from S, for $k = 0, 1$.
The environment's local state is used to record the history of events taking place in the system. At each round, either

a) S sends the bit to R and R does nothing,

b) S does nothing and R sends an *ack* to S, or

c) both S and R send messages.

We denote these three possibilities by

(sendbit, Λ) , (Λ, sendack) , (sendbit, sendack)

respectively.

Thus, we let the environment's state be a sequence of elements from the set

{(sendbit, Λ) , (Λ, sendack) , (sendbit, sendack)}.

Here the m-th member of the sequence describes the actions of the sender and receiver in round m.

There are many possible runs in this system , but these runs must satisfy certain constraints. Initially the system must start in a global state where nothing has been recorded in the environment's state, neither S nor R has received any messages, and S has an initial bit of either 0 or 1 .

Thus, the initial global state of every run in the system has the form $(<>, k, \lambda)$, where $<>$ is the empty sequence and k is either 0 or 1.

In addition, consecutive global states $r(m) = (s_e, s_S, s_R)$ and $r(m+1) = (s'_e, s'_S, s'_R)$ in a run r are related by the following conditions:

• If $s_R = \lambda$, then $s'_S = s_S$, $s'_e = s_e \bullet$ (sendbit, Λ) where $\bullet$ is the operation of concatenation, and $s'_R = \lambda$ or $s'_R = s_S$.

Before R receives a message, it sends no messages; as a result S receives no message, so it continues to send the bit and its state does not change. R may or may not receive the message sent by S in round $(m+1)$.

• If $s_S = s_R = k$, then $s'_R = k$, $s'_e = s_e \bullet$ (sendbit, sendack), and either $s'_S = k$ or $s'_S = (k, ack)$.

After R has received S's bit, it starts sending acknowledgments, and its state undergoes no further changes. S continues to send the bit, and it may or may not receive the acknowledgment sent by R in round $m+1$.

• If $s_S = (k, ack)$, then

a) $s'_e = s_e \bullet (\Lambda$, sendack)

b) $s'_S = s_S$

c) $s'_R = s_R$

Once S has received R's acknowledgment, S stops sending the bit and R continues to send acknowledgments. The local states of S and R do not cha nge any more. In b) s'_S contains k as the memory of the initial bit, its stopping to send it is not expressed, since we have not defined the corresponding local state for s.

We take the system R^{bt} describing the bit-trasmission problem to consist of all the runs meeting the constraints just described.

Example 1. shows how many choices have to be made in describing a system, even in simple cases. The example also suggests that the process of describing all the runs in a system of interest can be rather tedious. As we said before, getting a good representation of a system can be difficult. The process is far more of an art than a science. We shall return to this point later on, when we shall extend the framework to deal with protocols and programs. This will give us a relatively straightforward way of describing systems in many applications of interest.

Incorporating Knowledge

We already have seen in our discussion of the bit-transmission problem (Example 1) that we were making statements such as „R does not *know* for certain that S receved its acknowledgment“.

A central thesis of this exposition is that we often want to think of an agent's actions as depending on her knowledge.

Indeed, our framework has been designed so that knowledge can be incorporated in a straightforward way. The basic idea is that a statement such as „*R does not know the statement* A “ means that, as far as R is concerned, the system could be at a point where A does not hold. The way to capture that is closely related to the notion of possible worlds in Kripke structures.

We think of R´s knowledge as being determined by its local state, so that R cannot distiguish two points of the system in which it has the same local state, and it can distinguish points in which its local state differs.

We now formalize these ideas.
As we shall see, a system can be viewed as a Kripke structure except that we have no function π telling us how to assign truth to the primitive propositions from the set Φ of the language of formulas.

In the terminology of general structures, a system can be viewed as a *frame*. To view a system as a Kripke structure, we assume that we have a set Φ of primitive propositions, which we can think of as describing basic facts about the system. In the context of distributed systems, these might be such as „the value of the variable x is 0 “, „ process 1‘s initial input was 17 “, „ process 3 sends the message μ in round 5 of this run “, or „ the system is deadlocked “. For simplicity, we are assuming that the system can be described adequately using propositional logic, however, the extension of the framework to use the first-order logic is not difficult.

An *interpreted system* $\mathbf{I}$ consists of a pair (R, π), where R is a system over a set $\mathcal{G}$ of global states and π is an interpretation for the propositions in Φ over $\mathcal{G}$, which assigns truth

values to the primitive propositions at the global states. Thus, for every p in Φ and state s in $\mathcal{G}$, we have a boolean value $\pi(s)(p)$ in {**true, false**}.

Note that Φ and π are not intrinsic to the system R. They constitute additional structure on top of R that we, as outside observers, add for our convenience, to help us analyze or understand the system better.

We refer to the points and states of the system R as points and states, respectively, of the interpreted system $\mathbf{I}$. That is, we say that the point (r, m) is in the interpreted system $\mathbf{I} = (\mathsf{R}, \pi)$ if $r \varepsilon \mathsf{R}$, and similarly, we say that $\mathbf{I}$ is a system over state space $\mathcal{G}$ if R is.

Of course π induces also an interpretation over the points of R; simply take $\pi(r, m)$ to be $\pi(r(m))$. We refer to the points and states of the system R as points and states, respectively, of the interpreted system I.

To define knowledge in interpreted system, we associate with an interpreted system $\mathbf{I} = (\mathsf{R}, \pi)$ a Kripke structure

$$M_{\mathbf{I}} = (S, \pi, K_1, \ldots, K_n)$$

in a strightforward way: We simply take S to consist of the points in $\mathbf{I}$, and take $K_1, \ldots, K_n$ to be some binary relations on S.

Note that there is no possibility relation for the environment because usually we are not interested what the environment knows

For the possibility relation K_i, we choose a specific relation defined as follows: if $s = (s_1, \ldots, s_n)$ and $s' = (s'_1, \ldots, s'_n)$ are two global states in R, then we say that s and s' are *indistinguishable to agent* i, and write $s \sim_i s'$ if i has the same state in both s and s', *i.e.,* if $s_i = s_i'$.

We can extend the indistinguishability relation $\sim_i$ to points: we say that two points (r, m) and (r', m') are *indistinguishable to* i, and write $(r, m) \sim_i (r', m')$ if $r(m) \sim_i r'(m')$ (or equivalently if $r_i(m) = r'_i(m')$).

Clearly $\sim_i$ is an equivalence relation on points. When we speak of knowledge in interpreted systems, we assume that the K_i relation in M_I is defined by $\sim_i$.

Intuitively, agent i considers a state s' possible in a state s if s and s' are indistinguishable for agent i . Thus, the agents' knowledge is completely determined by their local states.

We shall see many examples where tihis notion of knowledge is useful for analyzing multi-agent systems. However, there are certain applications for which the externaly knowledge is inappropriate.

For example, we shall soon consider an example involving knowledge bases, where it may be more appropriate to consider knowledge base's beliefs , rather than its kowledge. As we shall see, by using a slightly different K_i relation instead of $\sim_i$, we do get a reasonable notion of belief.

By abuse of notation, we shall denote $L_n(\Phi)$ the set of formulas obtained the set of primitive propositions in Φ , by closing it under propositional connectives, and the modal operators $K_1 , \ldots , K_n$. Usually we omit Φ when it is clear from the context, writing just L_n .

Similarly, we denote by L_n^C , L_n^D , and L_n^{CD} the languages that result from L_n by adding the modal operators for common knowledge, distributed knowledge and operators for both common and distributed knowledge.

We can now define what it means for a formula $A \varepsilon L_n^{CD}$ to be true at a point (r, m) in an interpreted system **I** by applying the definitions for the ordinary Kripke structure M to the related Kripke structure M_I .

Thus, we say that $(\mathbf{I}, r, m) \models A$ iff $(M_I, s) \models A$, where $s = (r, m)$.

For example,

$(\mathbf{I}, r, m) \models p$ (for $p \varepsilon \Phi$) iff $\pi(r, m)(p) =$ **true**

$(\mathbf{I}, r, m) \models K_i A$ iff for all $(r' m') \sim_i (r, m)$, we have $(\mathbf{I}, r', m') \models A$

The definitions of truth for formulas involving common and distributed knowledge are left as an excercise.

Since π is a function on global states, the truth of a primitive proposition q at a point (r,m) depends only on the global state $r(m)$.

This seems like a natural assumption. The global state is meant to capture everything relevant about the current situation. Quite often, in fact, the truth of a primitive proposition q of interest depends, not on the whole global state, but only on the component of some particular agent.

For example, the truth of a statement such as "process 2 received process 1's message" might depend only on process 2's state.

In that case, we expect π to respect the *locality* of q, that is, if s and s' are two global states and $s \sim_i s'$, then $\pi(s)(q) = \pi(s')(q)$.

Locality is one natural assumption in many applications. However, there are statements that may depend on more than one global state and the number of states involved cannot be estimated in advance.

We shall concentrate only on such statements that are related to one run in the system in particular.

Statements that depend on more than just one state.

Motivation. Consider, for example, a statement such as

"eventually (at some later point in the run) the variable x is set to 5". (1)

Obviously, there could exists two points (r,m) and (r',m') with the same global state, such that the statement (1) is true at (r,m) and false at (r',m'). Thus, such a (temporal) statement cannot be represented by a primitive proposition from Φ in our framework.

Indeed, although such statements can depend on global states, we cannot express them by any formula in our language.

The following fact is more or less obvious.

Proposition 1.

For every formula $A \in L^{CD}$, if the points (r,m) and (r',m') have the same global state $r(m) = r'(m')$, we have

$$(\mathbf{I},r,m) \models A \quad \text{iff} \quad (\mathbf{I},r',m') \models A$$

While we could deal with this problem by allowing the truth of primitive propositions, i.e. the truth function π to depend on the local point, and not just the the global state, the more acceptable way to express such temporal statements is to adda modal operators for time into the language. We return to this problem in the next paragraph. Now, we shall continue with the semantics of interpreted systems.

Definition. (Formulas valid in interpreted systems)

We say that a formula A is valid in the intepreted system $\mathbf{I}$

Definition. (Formulas valid in interpreted systems)

(i) We say that a formula A is valid in the intepreted system $\mathbf{I}$ and write $\mathbf{I} \models A$, if $(\mathbf{I},r,m) \models A$ holds for all points (r,m) in $\mathbf{I}$.

(ii) For a class C of interpreted systems, we say that a formula A is valid in C, and write $C \models A$, if A is valid in all interpreted systems in the class C.

Incorporating Time

Motivation.

Example 1 has shown that our language is not expressive enough to handle conveniently the full complexity of even simple situations. For example, we might want to make statements like "the receiver eventually knows the sender's initial bit". We have already observed that we cannot express such temporal statements in our language.

To be able to make temporal statements, we extend our language by adding *temporal operators*, which are new modal operators for talking about time. From the variety of such operators, we focus here on four of them.

The operators, we use to extend our language are

- □ ("always"), its dual,
- <> ("eventually"),
- O ("next time") and
- U ("until")

Intuitively, □*A* is true if *A* is true now and at all later points; <>*A* is true if *A* is true at some point in the future; O*A* is true if *A* is true at the next step; and *A* U *B* is true if *A* is true until *B* is true.

More formally, in interpreted systems, we have

$(I, r, m) \models \Box A$ iff $(I, r, m') \models A$ for all $m' \geq m$,

$(I, r, m) \models \Diamond A$ iff $(I, r, m') \models A$ for some $m' \geq m$,

$(I, r, m) \models OA$ iff $(I, r, m+1) \models A$, and

$(I, r, m) \models A \,U\, B$ iff $(I, r, m') \models B$ for some $m' \geq m$ and
$(I, r, m'') \models A$ for all m'', $m \leq m'' < m'$

Note that our interpretation of O*A* as "*A* holds at the next step" makes sense in discrete time (which is our case). All the other temporal operators make perfect sense even for continuous time.

It is easy to show that

$\Box A \leftrightarrow \neg\Diamond\neg A$

$\Diamond A \leftrightarrow \neg\Box\neg A$

$\Diamond A \leftrightarrow true \,U\, A$

Thus, we can take O and U as basic operators, and define □ and <> in terms of U .

As we have proved earlier, if two points (r, m) and (r', m') in an interpreted system **I** have the same global state, then they agree on all formulas in L_n^{CD}.

Once, we add time to the language, it is no more true. For example, it is easy to construct an interpreted system **I** and two points (r, m) and (r', m') in **I** such that $r(m) = r'(m')$, but $(I, r, m) \models \Diamond p$ and $(I, r', m') \models \neg\Diamond p$.

To do that, construct an interpreted system I, with two runs *r* and *r'* , such that $r(0) = r'(0)$ and $r(1)$ is different from $r'(1)$. Define the truth function π in such a way that $\pi(r, 1)(p)$ = true and $\pi(r', m)(p)$ = false for all $m \geq 1$. In fact, both runs can have the same lengt 2 .

In general, temporal operators are used for reasoning about events that happen along a single run.

For example, in the bit-transmission problem, the formula □(*recbit* -> <>*recack*) says that if at some point along a run the receiver receives the bit sent by the sender, then at some point in the future the sender will receive the acknowledment sent by the receiver.

By combining temporal and knowledge operators, we can make assertions about the evolution of knowledge in the system.

For example, in the context of the bit-transmission problem we may want to make statements such as "the receiver eventually knows the sender's initial bit". This statement can now be expressed by the formula

$$\Diamond(K_R(\mathit{bit} = 0) \vee K_R(\mathit{bit} = 1))$$

Once we have temporal operators, there are a number of important notions that we can express. We have already seen the handiness of □ and <>.

We mention two other useful notions obtained by combining □ and <>:

- The formula □<>A is true if A occurs *infinitely often;* i.e. $(I, r, m) \models \Box\Diamond A$ exactly if the set $\{m' \mid (I, r, m') \models A\}$ is infinite.
- The formula <>□A is true if A is true *almost everywhere; i.e.,* $(I, r, m) \models \Diamond\Box A$ if for some m' and all $m'' \geq m'$, we have $(I, r, m'') \models A$.

The temporal operators that we have defined can talk about events that happen only in the present or future, not events that happened in the past. It is possible to inroduce temporal operators for reasoning about the past, for example, an analogue of <> saying "at some time in the past". We have not done so, while the above introduced temporal operators suffice for many applications and necessity to reason about the past occurs very rarely.