Aplication:

Games and Message-Passing Systems

Game Trees.

The goal of game theory is to understand *games* and how they should be played. To a game theorist, a game is an abstraction of a situation where players interact by making "moves". Based on the moves made by the players, there is an outcome, or payoff, to the game.

It should be clear that standard games such as poker, chess, and bridge are games in this sense. For example, the "moves" in bridge consist of bidding and playing the cards.

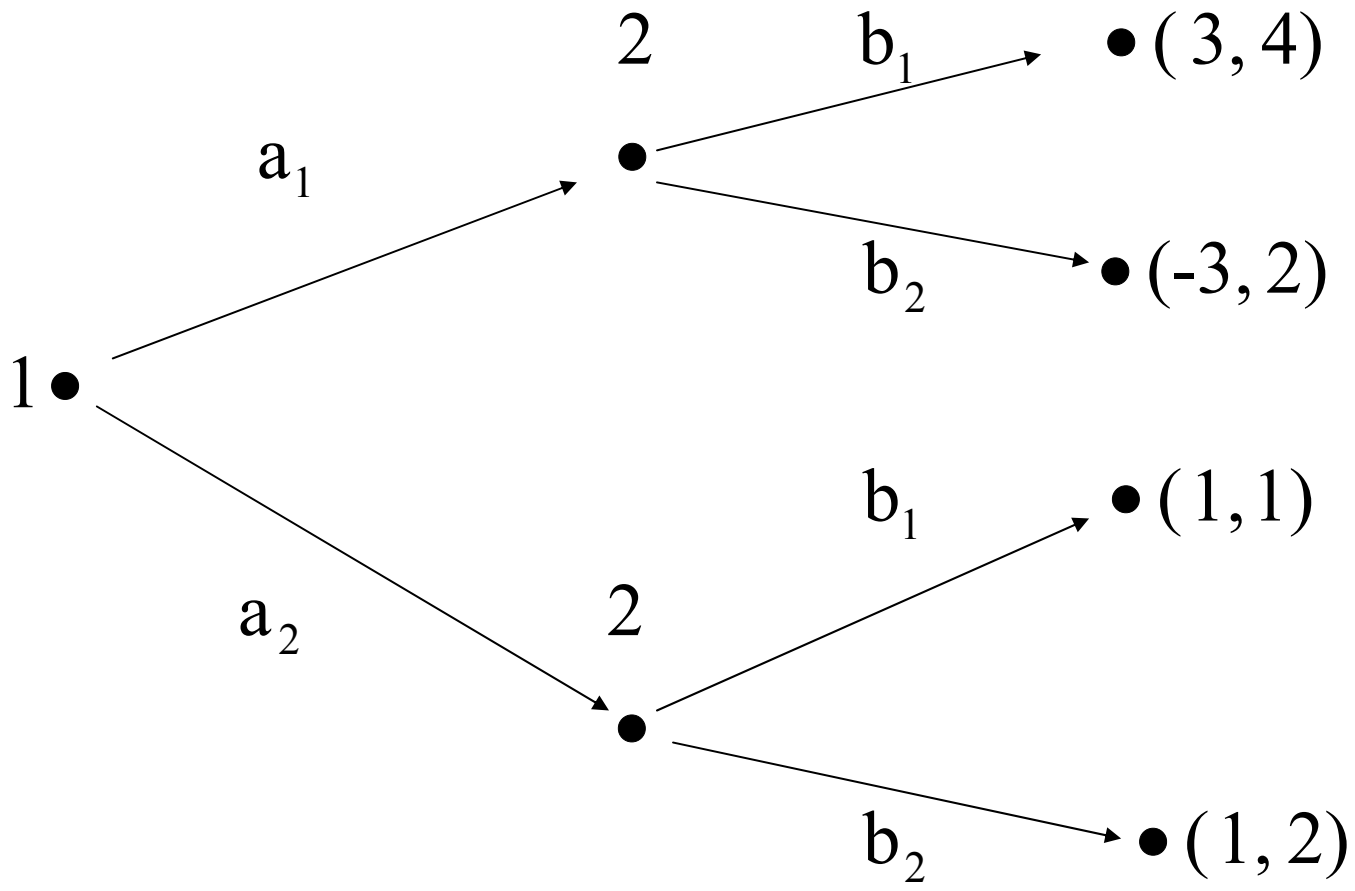
There are rules for computing how many points each side gets at the end of a hand of bridge. This is the payoff.

Standard economic interactions such as trading and bargaining can also be viewed as games, where players make moves a receive payoffs.

Games with several moves in sequence are typically described by means of a *game tree.*

We shall illustrate it by a game tree of Game 1.

Game 1


1
a1
a2
2
2
b1
b2
b1
b2
(3, 4)
(-3, 2)
(1, 1)
(1, 2)

In the Game 1, there are two players, 1 and 2, who move alternately. Player 1 moves first, and has a choice of taking action $\mathbf{a}_1$ or $\mathbf{a}_2$.

This is indicated by labeling the root of the tree with a 1, and labeling the two edges coming out of the root with $\mathbf{a}_1$ and $\mathbf{a}_2$.

After player 1 moves, it is player 2's turn. In Game 1, we assume that player 2 knows the move made by play - er 1 before she moves. At each of the nodes labeled with a 2, player 2 can choose between taking action $\mathbf{b}_1$ or $\mathbf{b}_2$.

In general, player 2's set of possible actions after player 1 takes the action $\mathbf{a}_1$ may be different from player's 2 actions after player 1 takes the action $\mathbf{a}_2$.

After these moves have been made, the players receive a payoff. The leaves of the tree are labeled with the payoffs.

In the Game 1, if player 1 takes action $\mathbf{a}_1$ and player 2 takes action $\mathbf{b}_1$, then player 1 gets a payoff of 3, while player 2 gets a payoff of 4 (denoted by the pair (3, 4)).

A *play* of the game corresponds to a path in the game tree, i.e. it is a complete sequence of moves by the players from start to finish.

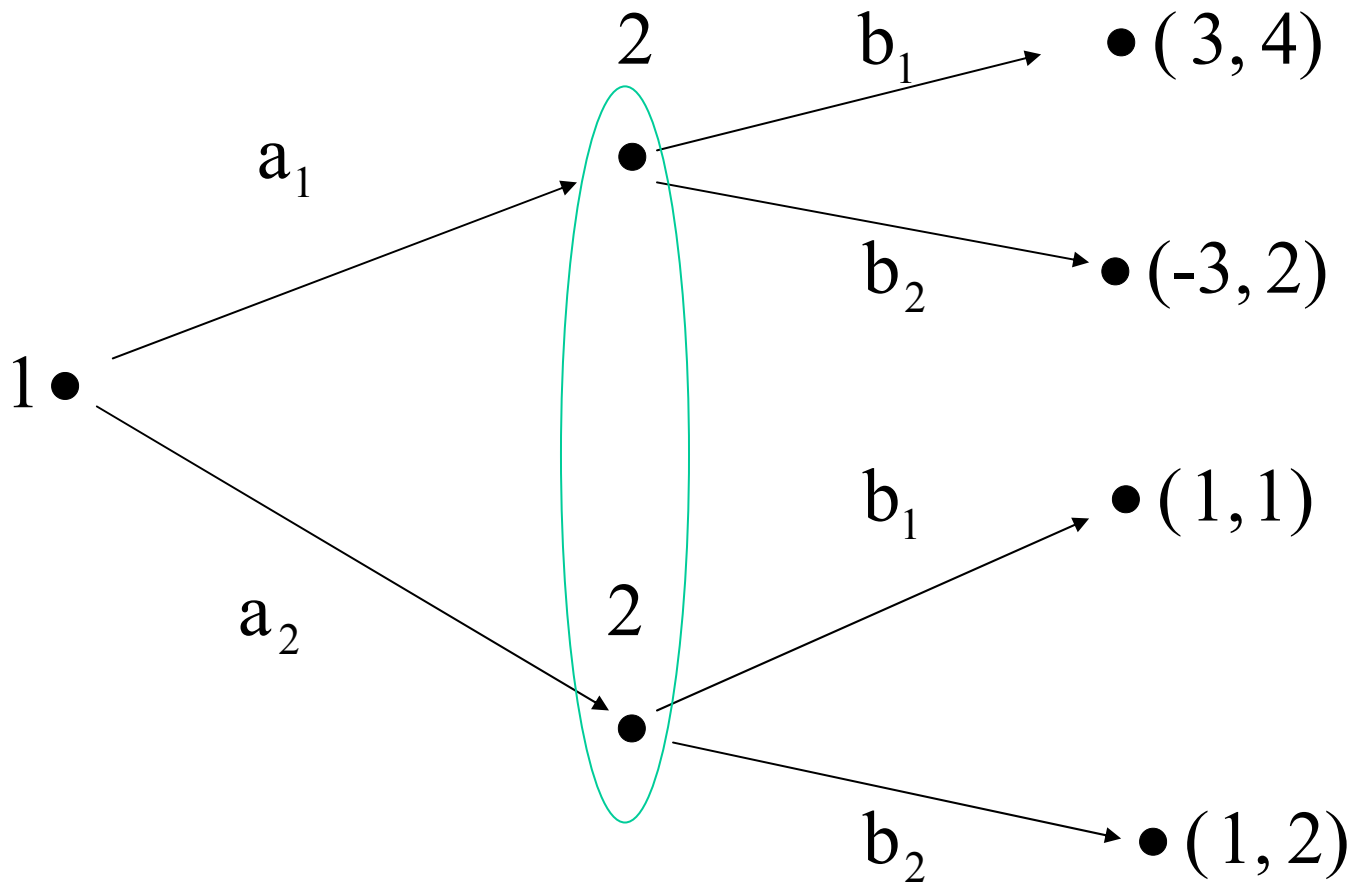
It should be clear, that at least in principle, chess could also be described by a game tree. The nodes represent board positions, and the leaves of the tree represent positions where the game ended. If we suppose that all games are played to the end, then the moves at each node are the legal chess moves in this position. There are only three possible outcomes: a win for White (player 1), a win for Black (player 2, or a draw. The plays in this game tree correspond to the possible (complete) games of chess.

The Game 1 represented by its game tree is an example of a game of *perfect information* . Every event relevant to the game takes place in public.

A player knows all the moves that have been made before she moves. Chess is another example of a game with perfect information. By way of contrast, bridge and poker are not games of perfect information.

One of the key issues studied by game theorists is how the information available to the players when they move affects the outcome of the game. Game theorists are interested mainly in games where agents do not have perfect information.

Game 2


2
b1
(3, 4)
a1
b2
(-3, 2)
1
b1
(1, 1)
a2
2
b2
(1, 2)

The game tree for the Game 2 is identical to the game tree for the Game 1, except that the two nodes where player 2 moves are enclosed by an oval. This indicates that the two nodes are indistinguishable to player 2, or, as the game theorists would say, they are in the same information set.

This means that when player 2 makes her move in this game, she does not know whether player 1 choose action $\mathbf{a}_1$ or $\mathbf{a}_2$.

In general, game theorists use the information set to represent the information that a player has at a given point in the game.

It is assumed that

(a) a player always knows when it is her turn to move,

(b) player *i* has the same choices of actions at all the nodes in her information set.

By (a) there cannot be two nodes in player's *i* information set, such that player *i* is supposed to move at one of the nodes and not at the other.

(b) says that it does not make sense for a player to be able to perform different actions at nodes she cannot distinguish. As we can see from the game tree for the Game 2, the set of actions from which player 2 must choose is identical at the nodes where she moves. In contrast to Game 1, where the set of possible moves did not *have* to be identical, in our example they do.

Perhaps the most obvious way of modeling Game 1 is to take each play as a run. Since we assume that G_1 is a game of perfect information, what happens at each state must be reflected in the player's states.

- Each player's initial state has the form $<>$, representing the fact that nothing has yet happened.
- After player 1's move, the local states of both players encode the move the player 1 has made. Thus the local states of both players have the form $< \mathbf{a}_i >$, for $i = 1, 2$.
- Finally, after player 2's move, we assume that both player's states include player 2's move, and the payoff.

We can ignore the environment's state here. We presume that the player's local states include all the information of interest of us. Thus we take the environment's state to be λ .

We call the resulting system R_1 .

We remark that in a more general settings, game theorists view "nature", or the environment, as another player in the game. In this case it may be appropriate to have a more complicated environment's state.

Note that as we have described R_1 , both players have identical local states at all points. This is a formal counterpart to our assumption that G_1 is a game of perfect information.

It is easy to show that the moves made, as well as the payoffs received by the players, are common knowledge once they take place.

In the games of perfect information there is very little uncertainty, which leads to such a simple model. We remark that even in games of perfect information such as this one, the players usually follow particular strategies which are not necessarily common knowledge. Defining strategies and capturing them in the model will be one of the subjects treated later.

What system does G_2 correspond to ?

- Again, we assume that each play corresponds to a run, and the player's initial states have form <>.
- Just as in R_1 , we can also assume that player 1's local state includes her move after she made it. We do not want

player 2's local state to include this information.

• Nevertheless, player 2's state must encode the fact that player 1 has moved.

• For definiteness, we assume that immediately after player 1's move, player 2's state has the form $\langle move \rangle$, which indicates that it is player 2's turn to move.

• We assume that after player 2's move both player's states include the move and the payoff.

This gives us the system R_2 .

The key difference between G_1 and G_2 is that player 2 does not know what player 1's is after she has made it. Player 2's state must be different before player 1 moves and after player 1 moves, for otherwise she would not know that it is her turn to move. Essentially, the <*move*> is just an indication that it is player 2's move.

In the case that player 2's local state at the end of the game G_2 includes her move and the payoff (for both players), player 2 may discover what player 1's move was. In particular, it is true in our setting, where the payoffs are different for every play.

It is obvious at this point that we can capture a situation in which players are not informed about the payoffs immediately, or perhaps that each player is informed of her or his own payoff and not the other's. All we need is to modify what goes into a players'local state.

The systems R_1 and R_2 correspond to games G_1 and G_2 in that each play of the game is captured by one of the runs and every run captures a possible play of the game.

Of course, these systems are not the only possible representations of these games. For example, we could have used the information sets as local states of the agents. Another possible representation includes a representation of the strategy of the players.

Synchronous Systems

A standard assumption in many systems is that agents have access to a shared clock, or that actions take place in rounds or steps, and agents know what round it is at all times.

Thus, it is implicitely assumed that the time is common knowledge, so that all the agents are running in synchrony.

This assumption has already arisen in some of the systems we have considered. In particular, we implicitely made this assumption in our presentation of the muddy children puzzle and of the two games G_1 and G_2.

In computer science, many protocols are designed so that they proceed in rounds where no agent starts round $(m+1)$ before all agents finish round m.

How can we capture synchrony in our framework? Since an agent's knowledge is determined by his local state, his knowledge of the time must be encoded somehow in the local state. This global clock need not measure "real time".

Definition. (Synchronous Systems)

(i) We say that R is a *synchronous system* if for all agents and points (r,m) and (r',m') in R, if $(r,m) \sim_i (r',m')$, then $m = m'$.

(ii) We say that an interpreted system $I = (R, \pi)$ is synchronous if R is.

(i) expresses our intuition that in a synchronous system, each agent i knows what time it is: at all points that i considers possible at the point (r,m), the time (on the system's shared clock) is m.

The time is encoded in each agent's local state.

In particular, this means that i can distinguish points in the present from points in the future; i has a different local state at every point (r,m) in a run r.

Examples. (a) the systems R_1 and R_2 corresponding to the games G_1 and G_2 and are indeed synchronous .

(b) Intuitively, the muddy children puzzle should be modeled as a synchronous system. (We shall show it later).

(c) On the other hand, the system I^{kb} that we used to model the Knowledge base is not synchronous. Synchrony was not a major issue in that case.

We could make it synchronous, either by adding a clock to the Knowledge base's and Teller's local states, or assumng that the Teller tells the Knowledge base a new formula at every step. In the latter case the number of formulas in the Knowledge base's and Teller's local states encodes the time.

Perfect Recall

According to our definition of knowledge in a system, an agent's knowledge is determined by his local state.

Our definition admits the following two possibilities:

(i) an agent's local state may "grow" to reflect the new knowledge she acquires, while still keeping track of all the old information she had. Our definition does not requires this.

(ii) On the other side are models where it is possible that the agent's i information encoded in her local state $r_i(m)$ at time m in run r no longer appears in $r_i(m+1)$.

There are often scenarios of interest where we want to model the fact that certain information is discarded.

In practice, an agent may simply not have enough memory to remember everything she had learned.

On the other hand, there are many instances where it is natural to model agents as if they do not forget, that is, they have *perfect recall* .

Perfect recall is sufficiently common in applications to warrant a definition and to be studied as a separate property of systems.

Perfect recall means, intuitively , that an agent's local state encodes everything that has happened (from that agent's point of view) thus far in the run.

Among other things, this means that the agent's state at time $(m + 1)$ contains at least as much information as his state at time m . In other words, an agent with perfect recall should, essentially, be able to reconstruct his complete local history.

In the case of synchonous systems, since an agent's local state changes with every tick of external clock, this would imply that the sequence $<r_i(0), \ldots, r_i(m)>$ must be encoded in $r_i(m + 1)$.

In systems that are not synchronous, agents are not necessarily afected by the passage of time on the external clock. Thus, an agent can sense that something happened only when there is a change in her local state. This motivates the following definitions.

Definition. (Perfect recall)

(i) Let *agent i's local state sequence at the point* (r,m) be the sequence of local states she has gone through in run r up to time m, without consecutive repetitions.

(ii) We say that *agent i has perfect recall in system* R if at all points (r,m) and (r',m') in R, if $(r,m) \sim_i (r',m')$, then agent i has the same local state sequence at both (r,m) and (r',m').

(iii) We say that a system R has perfect recall, if for every agent i, $r_i(m)$ encodes i's local state sequence in that, at all points where i's local state is $r_i(m)$, she has the same local state sequence.

Example. Assume that from time 0 through time 4 in run r agent i has gone through the sequence

$$<s_i, s_i, s'_i, s_i, s_i>$$

of local states, where s_i and s'_i are different.

We model this by her local state sequence

$$<s_i, s'_i, s_i>$$

at $(r, 4)$.

Thus, proces i's local state sequence at a point (r,m) describes what has happened in the run up to time m, from i's point of view.

According to the definition, agent i has perfect recall if she "remembers" her local state sequence at all times.

Note that the systems R_1 and R_2 corresponding to the games G_1 and G_2 assume perfect recall, since the players keep track of all the moves that they make.

Our representation I^{kb} of knowledge bases assumes perfect recall as well.

In fact, perfect recall is a standard assumption made by game theorists.

As we shall see, perfect recall is an assumption made, either explicitely or implicitely, in a number of contexts.

Time and ignorance in perfect recall systems

One might expect that in systems where agents have perfect recall, once an agent knows a fact A at a point (r,m) she will know it at all points in the future. That is, we might expect that

$$K_i A \text{ -> } \Box K_i A$$

But this need not be true about some specific statements talking about time.

One problem arises with statement talking about the situation "now", such as the statement A saying "it is currently time 0". At time 0, an agent i may know A (say, if she has access to a clock) but agent i will certainly not always know that it is time 0.

Another problem comes from knowledge about ignorance. Consider the formula $\neg K_i p$ saying "the agent i does not know the fact p". Then it is not hard to construct a system where agents have perfect recall such that agent 1 initially does not know p, but she later learns p.

Thus, we have $\neg K_1 p$, and so $K_1 \neg K_1 p$ at time 0, but by assumption, $K_1 \neg K_1 p$ does not hold at all times in the future.

Hence, certain temporal statements and knowledge about the ignorance does not persist in the presence of perfect recall.

Nevertheless, the intuition that in the presence of perfect recall, once the agent knows A, then she never forgets A is essencially correct. Namely, it is true for all *stable* formulas.

Definition. (Stable formulas)

We say that a formula A is *stable* with respect to the interpreted system I, if once A is true it remains true in future. Hence A is stable if we have $I \models A \rightarrow \Box A$.

We shall remember the following positive and negative facts.

Proposition 1.

Suppose that A and B are stable, then

(i) $(A \,\&\, B)$ and $(A \text{ v } B)$ are stable.

(ii) If I is a system with perfect recall , then $K_i A$ and $C_G A$ are stable. Thus, in a system with perfect recall, if an agent knows a stable formula at some point, then she knows it from then on. And similarly for common knowledge.

(iii) If in addition the system is synchronous, then $D_G A$ is stable, as well.

Proposition 2.

(i) The formula $K_i A$ need not be stable if A is not stable even if assuming perfect recall and synchrony.

(ii) There is an interpreted system where agents have perfect recall and a stable fomula A such that $D_G A$ is not stable. (Thus synchrony is necessary in the statement (iii) of Proposition 1.)

While the proof of Proposition 1 is not complicated, the proof of Proposition 2 is more demanding. We shall not give the proofs here.

How reasonable is the assumption of perfect recall ? This depends on the application and on the model we choose.

It is easy to see that perfect recall requires every agent to have a number of local states at least as large as the number of distinct local state sequences she can have in the system .

• This fact is acceptable in systems where agents change state rather infrequently.

• On the other hand, if we consider systems where there are frequent state changes or look at systems over long intervals of time, then perfect recall is an unreasonable assumption. In such situations, perfect recall may require a rather large (possibly infinite) number of states.

The simple protocol of the bit-transmition problem (Example 1) is one in which the Sender S and the Receiver R undergo very few state changes.

The states of S and R do not reflect every separate sending or receiving a message. The states change only when a message is received *for the first time*.

According to our definitions, both S and R have perfect recall in this case, despite the fact that neither S nor R remember how many times they have received or sent messages.

The point is that S and R recall everything that was ever encoded in their states.

Benefits of the assumption of perfect recall. Frequently, during the design phase of a multi-agent system, we are at a loss what to include in the agent's local state . The problem is aleviated, if we simply include to the state record all events that the agent is involved in and assume that agents have perfect recall.

If we can gain a reasonable understanding of the system under the assumption of perfect recall, we can then consider to what extent forgetting can be allowed without invalidating our analysis.

Message-Passing Systems

In many situations, particularly when analyzing protocols run by processes in a distributed system, we want to focus on the communication aspects of the systems. In what follows, we shall call agents processes.

We introduce the notion of a *message-passing system* , where the most significant actions are *sending* and *receiving* messages and *internal actions* of processes.

The main ingrediences of message-passing systems are

- MSG the set of messages common for all processes,
- Σ_i the set of initial states for each process i ,
- INT_i the set of internal actions for each process i ,

the names of events *send* (μ, j,i), *receive* (μ, j,i) or *int* (**a**,i) , where $\mu \varepsilon MSG$ and **a** ε INT_i . We assume that these names correspond to events as follows:

send (μ, j,i)	"message μ is sent to j by i",
receive (μ, j,i)	"message μ is received from j by i",
int (**a**, i)	"internal action **a** is performed by i".

As we are interested in the communication aspects of the system, the details of the internal actions are not relevant here.

In a run r at a point $r(m)$ a process i may perform several actions.

For example, process i performs the actions of sending the message μ to process j, receives the message η from process k and also performs some internal action **a**.

We shall denote it by the set

$$\{send\ (\mu, j,i),\ receive\ (\eta, k,i),\ int\ (\mathbf{a}, i)\} \qquad (1)$$

Now we can define the concept of history of a process from the start to the point (r,m).

Definition. (History)

History of the process i is a sequence of sets such as (1).

At the point $(r,0)$, process i's history is a sequence consisting from the singleton set containing only i's initial state.

If process i performs some actions at the point (r,m), for example, those in the set (1), then the i's history at the point (r,m) is the result of appending the set (1), hence

$$\{send\ (\mu, j,i),\ receive\ (\eta, k,i),\ int\ (\mathbf{a}, i)\}$$

to the i's history at $(r,m-1)$.

If i performs no action in the round m, then its history at (r,m) is the same as its history at $(r,m-1)$.

Note that we are distinguishing performing no action from performing some kind of null action to indicate that time passed. A null action would be modeled as an internal action.

Definition. (Occurence of an event)

By abuse of terminology, in message-passing systems, we speak of names

$$send\ (\mu, j, i),\ \ receive\ (\eta, k, i)\ \ \text{and}\ \ \ int\ (\mathbf{a}, i)$$

as *events*.

We say that an event *occurs* in round $(m + 1)$ of run r if it appeares in some process's history in $(r, m+1)$, but not in any process's history in (r, m).

In message-passing systems, the process's local states at any point is its history. To define the concept of message passing system, we need to impose some consistency conditions on global states.

Definition. (Message-passing systems (m.p.s))

Given sets Σ_i of initial states and INT_i of internal actions for processes 1, ... , n , and a set MSG of messages common to all processes, we define *a message-passing system* (over Σ_i INT_i and MSG) to be a system such that for each point (r,m) , the following constraints are satisfied.

MP1. $r_i(m)$ is a history over Σ_i , INT_i and MSG,

MP2. for every event $receive\ (\mu, j,i)$ in run r there exist a corresponding event $send\ (\mu, i, j)$ in $r_j(m)$, and

MP3. $r_j(0)$ is a sequence with only one member: singleton set containig the process's i initial state and
$r_j(m+1)$ is either identical to $r_j(m)$ or the result of appending a set of events to $r_j(m)$.

We have ignored the environment's state since it is defined in a different way and its details are not relevant to the constraints MP1 to MP3.
The local state of environment become more important when considering protocols.

Assuming MP1 to MP3, we ensure that processes have perfect recall since local state of each process is its history.

In practice, we may want to add further requirements:

(i) reliability

(ii) keeping the order of messages

(iii) synchrony

(iv) avoiding sleeping processes

(v) limited time for message delivery

• *reliability* makes *communication guaranteed :* every message sent is eventually received. We can express it by the following constraint.
MP4 for all processes i , j , and all points (r,m) , if *send* (μ, j,i) is in $r_i(m)$, then there exists $m' \geq m$ such that *receive* (μ, i, j) is in $r_j(m')$,

• *keeping order* guarantees that messages arrive in the order in which they are sent,

• *synchrony* forces $r_i(m) \neq r_i(m+1)$ for all i , i can then compute the time from its history.

• *avoiding sleeping* forces each process to take some action once every k rounds.

• *limited time delivery* requires that messages arrive in k rounds.

Fortunately, the model can capture a wide range of assumptions quite easily.

Asynchronous message-passing systems (a.m.p.s)

Possible reasons of asynchrony

• little or no information about time (no global clock),

• a process may suddenly slow down relative to other processes,

• no upper bound on message delivery.

We consider asynchronous systems, where we assume that

• processes may work at arbitrary rates relative to each other,

• there is no bound on message delivery times.

We proceed much as in the previous section. As before, we assume that each process's local state consists of its history.

We use to some extent the local states of the environment. As in the case of bit-transmition problem, the environment's local state records the events that have taken place so far and the order in which they occured.

We make the following simplifying assumptions:

(i) in each round, at most one event takes place for each process,
(ii) for each process, all the events in its history are distinct.

It follows from (i) that now a history is a sequence starting with an initial state followed by singleton sets $\{e\}$ consisting of the only event that took place. It is natural to replace the singleton by the event e itself.

The assumption (i) is reasonable if we model time at a sufficiently fine level of granularity.

The assumption (ii) makes the exposition much easier if we do not have to distinguish different occurences of the same event in a given run. It helps if we want to consider occurences of events and their temporal relationships.

In particular, (ii) forces each process never to perform the same action twice in a given run.

(We can dispose with this constraint if we simply change our representation of events in the history replacing the event e by the ordered pair $<k,e>$, where k indicates that it is k^{th} occurence of event e in the given run.)

Preliminary knowledge and its elimination.

In any message passing system, a process knows at least what is in its history. It may well know more, in particular if it has some a priory knowledge. For example, in a system where it is common knowledge that all processes perform an action at every round, a process can certainly deduce information made by other processes from amount of progress it has made itself.

To eliminate all such additional knowledge, we consider possible *all* runs consistent with the assumption MP1 to MP3.

To make it precise, we use the following definition.

Definition. (Prefixes and prefix-closed sets of histories)

Recall that in message-passing systems (synchronous or asynchronous) the local state of a process i in the point (r,m) is its history, hence a sequence of sets consisting of events that took place in the corresponding step.

(i) We call a *prefix* of a history h any non-empty initial sequence of h.

(ii) We say that a set V of histories is *prefix-closed* if whenever h is a history in V, then every prefix of h is also in V.

Definition. (Asynchronous message-passing system)

Let $V_1, \ldots, V_n$ be prefix-closed sets of histories for processes $1, \ldots, n$ respectively.

Let $R(V_1, \ldots, V_n)$ be the set of all runs satisfying MP1, MP2, and MP3 such that all of process i's local states are in V_i.

We define an *asynchronous message-passing system* (a.m.p. system for short) to be one of the form $R(V_1, \ldots, V_n)$ for some choice of $V_1, \ldots, V_n$.

We shall see that this definition requires that with each run r the system contains many other runs that can be constructed from r.

Remark. Why there are no reliable a.m.p. systems.

It seems that we could add MP4 to the definition of asynchronous system in order to guarantee reliability. The following example shows that an a.m.p. system in which communication takes place can never be reliable.

Example 1. Suppose that $R(V_1, \ldots, V_n)$ is an a.m.p. system that includes a run r such that i sends j the message μ in round m of r. It can be shown that then the system must contain another run r' that agrees with r up to the beginning of round m, process i still sends μ to j in round m of r', but j never receives μ.

It folows that if at least one message was sent, the a.m.p. system cannot be reliable.

The following example illustrates the fact that in a.m.p systems every run r induces a number of other runs related to r.

Example 2. Suppose $r \in R$ and r^* is the run in which all events in r are "stretched out by factor of two. Thus, in r^*, all processes start in the same initial state as in r, no events occur in odd rounds of r^*, and, for all m, the same events occur in round $2m$ of run r^* as in round m of run r. Hence for all times m, we have $r^*(2m) = r^*(2m+1) = r(m)$.

It is easy to check that r^* satisfies conditions MP1-3 (since r does) so r^* must also be in R.

Similarly, any run that is like r except that there are arbitra-

ry long “silent intervals” between the events of r is also in R .

This shows that in a precise sense time is meaningless in a.m.p. systems.

To make possible a closer analysis of events, we define a notion of *potential causality* between events. This is intended to capture the intuition that event e might have caused event e’. In particular, we mean by this that e *necessarily* occured no later than e’.

Definition. Potential causality.

(i) For events e and e' in a run r, we write $e \text{-}^{r}\text{->} e'$ if either e' is a *receive* event and e is the corresponding *send* event, or
for some process i, events e, e' are both in i's history at some point (r,m) and either $e = e'$ or e comes earlier in the history.

(ii) we shall use the same symbol for the transitive closure of the above defined relation.

Note that $\text{-}^{r}\text{->}$ is an *anti-symmetric* relation, we cannot have $e \text{-}^{r}\text{->} e'$ and $e' \text{-}^{r}\text{->} e$ unless $e = e'$. However, this would not be the case if we allowed an event occur

more than once in a history. The following result makes precise the degree to which an a.m.p.s. is asynchronous.

It says that the potential causality relation $-^{r}->$ is the closest we can come in a.m.p. system to define a notion of ordering of events. Even if processes combine all their knowledge, they could not deduce any more about the ordering of events in run r than is implied by $-^{r}->$.

Notation. We assume that for each pair of events e , e', $Prec(e,e')$ is a primitive proposition in Φ. We say that the interpretation of these propositions in the interpreted a.m.p. system $I = (R, \pi)$ is standard if

$$\pi(r(m))(\text{Prec}(e,e')) = \textbf{true}$$

exactly if e , e'occur by round m of r , and e occurs no later than e' in r. The definition of π is correct, since we assumed that the environment keeps track of the events that have occured.

Proposition 1. Let G be the group of all processes, R be an a.m.p. system and assume that the interpretation of $Prec(e,e')$ in $I = (R, \pi)$ is standard. Then

$(I, r, m) \models D_G(Prec(e,e'))$ iff e, e'both occurred by round m and $e \text{-}^{r}\text{->} e'$.

Knowledge Gain in A.M.P. Systems

There are even closer connections between the potential causality ordering and knowledge. As we shall see, the relationship between knowledge and communication is mediated by the causality relationship $-^{r}->$.

Roughly speaking, the only way for process i to gain knowledge about process j is to receive a message. Although this message does not have to come directly from process j , it should be the last in a chain of messages, the first of which was sent by j.

Definition. Process chains

Suppose that $i_1, \ldots, i_k$ is a sequence of processes, with repetitions allowed, r is a run and $m < m'$.

(i) We say that $<i_1, \ldots, i_k>$ *is a process chain in* $(r, m..m')$ if there exist events $e_1, \ldots, e_k$ in run r such that event e_1 occurrs at or after the round $m + 1$ in run r, event e_k occurrs at or before round m', event e_j is in process i_j's history for $j = 1, 2, \ldots, k$, and $e_1 \text{-}^r\text{->} e_2 \ldots \text{-}^r\text{->} e_k$.

(ii) We say that $<i_1, \ldots, i_k>$ is a process chain in r if it is a process chain in $(r, m..m')$ for some $m < m'$.

Example 1. Suppose that in run r, process 1 sends the message μ to process 2 in round 1, process 2 receives μ in round 2, process 2 sends the meassage μ' to process 1 in round 3, and μ' is received by process 1 in round 3.

Then <1,2,2,1> is a process chain in $(r,0..3)$ (as is (1,2,1)).

This example suggests that process chains are intimately linked to the sending and receiving of messages. It is easy to see that <1,2,1> is a process chain in run r corresponding to events e_1, e_2, e_3 that occur in rounds m_1, m_2, and m_3, respectively then there must have been a message sent by process 1 between rounds m_1 and m_2 inclusive (i.e. at or after m_1, and at or before round m_2)

to process 2 and a message sent by process 2 between rounds m_2 and m_3 inclusive. More generally, we have the following lemma

Lemma 1. Suppose that $<i_1, \ldots, i_k>$ is a process chain in $(r, m \ldots m')$, with $i_j \neq i_{j+1}$, for $1 \leq j \leq k - 1$. Then there must be a sequence of messages $\mu_1, \ldots, \mu_{k-1}$ sent in r such that μ_1, is sent by i_1, at or after round $m + 1$, and μ_j, is sent by i_j strictly after μ_{j-1} is sent by i_{1-1} for $1 < j \leq k-1$.

In particular, at least k-1 messages must be sent in run r between rounds $m + 1$ and m' inclusive.

Note that it is not necessarily the case that μ_j is sent by i_j, to i_{l+1}, there may be a finite sequence of messages in between. The next definition is the key to relating knowledge and communication.

Definition. If $i_1, \ldots, i_k$ is a sequence of processes, we write $(r,m) \sim_{i1, \ldots, ik} (r',m')$, and say that (r',m') *is* $(i_1, \ldots, i_k)$-*reachable from* (r,m), if there exist points $(r_0,m_0), \ldots, (r_k, m_k)$ such that $(r,m) = (r_0,m_0)$, $(r',m') = (r_k, m_k)$ and $(r_{j-1}, m_{j-1}) \sim_j (r_j, m_j)$ for $i = 1, \ldots, k$.

Thus, (r',m') is $(i_1, \ldots, i_k)$-reachable from (r,m) if at the point (r,m) process i_1 considers it possible that i_2 considers it possible ... that i_k considers it posible that (r', m') is the current point.

Despite the notation, the relation of $(i_1, \ldots, i_k)$-reachability is not in general an equivalence relation if $k > 1$.

Lemma 2. Let R be an a.m.p. system, let r be a run in R, and let $m < m'$. For all $k \geq 1$ and all sequences $i_1, \ldots, i_k$ of processes, either $(r,m) \sim_{i1, \ldots, ik} (r,m')$ or $<i_1, \ldots, i_k>$ is a process chain in $(r, m..m')$.

Proof. We proceed by induction on k. If $k = 1$ and $< i_1 >$ is not a process chain in $(r, m..m')$, then it must be the case that no events occur in process i_1's history in r between rounds $m + 1$ and m' inclusive. It follows $(r,m) \sim_{i1} (r,m')$, as desired.

Suppose $k > 1$ and $< i_1 , \ldots , i_k >$ is not a process chain in $(r, m..m')$. Let e^* be the last event in k's history at the point (r,m'). We now define a new run r'. Intuitively, r' consists of all the events that occurred in r up to and including round m, together with all the events that occurred in r after round m that potentially caused e^*.

The run r' agrees with r up to time m (so that we have

$r'(m'') = r(m'')$ for $0 \leq m'' \leq m$). For $m < m'' \leq m'$ and each process i, we define $r_i'(m'')$ to be the sequence that results from appending to $r_i'(m)$ (in the order) all events e in $r_i(m'')$ that occurred between rounds $m+1$ and m'' inclusive such that $e \text{ -}^r\text{-> } e^*$.

Finally, we take $r'(m'') = r(m')$ for $m'' > m'$, i.e. no event takes place after time m'.

It is easy to check that $r_i'(m'')$ is a prefix (not necessarily strict) of $r_i(m'')$ for all $m'' \geq 0$, because if e' occurs in $r_i(m'')$ before e and $e \text{ -}^r\text{-> } e^*$ then we also have $e' \text{ -}^r\text{-> } e^*$.

It is now not hard to show that

(1) r' ε R i.e. to check that r' satisfies MP1-3,
(2) $(r', m') \sim_{ik} (r, m')$,
(3) $(r, m) \sim_{i1} (r', m)$, and
(4) $\langle i_1 , \ldots , i_{k-1} \rangle$ is not a process chain in $(r', m..m')$.

It follows from (4) and the induction hypothesis that

$$(r', m) \sim_{i1, \ldots, ik-1} (r', m')$$

Applying (2) and (3) we immediately get

$$(r, m) \sim_{i1, \ldots, ik} (r, m')$$

as desired.

The two conditions in Lemma 2 are not mutually exclusive.

It is possible to construct a run r such that, for example, $\langle 1,2 \rangle$ is a process chain in $(r, 0 .. 4)$ and $(r,0) \sim_{1,2} (r,4)$.

Theorem 1.

Let r be a run in an interpreted a.m.p. system I, and assume that $m < m'$.

(a) If $(I, r, m) \models \neg K_{ik} A$ and $(I, r, m') \models K_{i1} \ldots K_{ik} A$, then $\langle i_1, \ldots, i_k \rangle$ is a process chain in $(r, m..m')$.

(b) If $(I, r, m) \models K_{i1} \ldots K_{ik} A$ and $(I, r, m') \models \neg K_{ik} A$, then $\langle i_1, \ldots, i_k \rangle$ is a process chain in $(r, m..m')$.

The theorem essentially says that processes can gain or loose

knowledge only by sending and receiving messages.

Proof. We prove (b) by contradiction, the proof of (a) is similar. Suppose that $<i_1 , \ldots , i_k>$ is not a process chain in $(r, m..m')$. By lemma 2, we have $(r,m) \sim_{i1, \ldots ,ik} (r,m')$.

Thus, by definition, there are points $(r_0,m_0), \ldots , (r_k,m_k)$ such that $(r,m) = (r_0,m_0), (r,m') = (r_k,m_k)$ and for $j = 1, \ldots , k$ we have $(r_{j-1}, m_{j-1}) \sim_j (r_j , m_j)$.

We can now show by induction on j , that for $j = 1,..., k$ $(I, r_j , m_j) \models K_{ij} \ldots K_{ik} A$. In particular, it follows that $(I, r, m') \models K_{ik} A$, a contradiction.

Part (a) of Theorem 1 seems quite intuitive: knowledge gain can occur only as a result of receiving messages. Part (b) may seem somewhat counterintuitive: knowledge loss can occur only as the result of sending messages.

Individual processes typically can lose
(1) "positive" knowledge, and
(2) knowledge of their ignorance.
We give a rough image how it can happen in the following example.

Example 2. Suppose that process 1 sends process 2 a message "Hi", and that this is the first message sent from pro-

cess 1 to process 2. Before process 1 sends the message, 1 knows that 2 has not received any message from it. After it sends the messages, it loses this knowledge.

More deeply nested "positive" knowledge can be lost as well. This needs but a more complicated example. It can be illustrated by a number of locks put by different processes on a database.

Losing knowledge of ignorance can be illustrated by the formula p saying "the value of variable x is 0", where x is a variable local to process 3. Suppose that $(I, r, 0) \models \neg p$. Clearly $\neg K_1 p$ since process 1 cannot know a false fact.

By a lenghty computation using introspection axioms and some assumptions on the run r it is possible to show that at a later point $K_2K_1\,p$ and consequently $K_1\,p$ holds.

On the other hand, the following theorem shows that common knowledge can neither be gained nor lost in a.m.p. systems.

Theorem 2.

Suppose I is an interpreted a.m.p. system, r is a run in I, and G is a group of processes with $|G| \geq 2$. Then for all fromulas A and all times $m \geq 0$, we have

$$(I, r, m) \models C_G\ A \quad \text{iff} \quad (I, r, 0) \models C_G\ A$$

Proof by contradiction. Suppose

$$(I, r, m) \models C_G A \text{ and } (I, r, 0) \models \neg C_G A$$

Suppose that exactly l messages are sent between rounds 1 and m inclusive. Since $(I, r, 0) \models \neg C_G A$ there must be some sequence $i_1, \ldots, i_k$ of pairwise distinct processes in G such that $(I, r, 0) \models \neg K_{ik} \ldots K_{i1} A$. Let i, j be distinct processes in G and j be distinct from i_k.

Since $(I, r, m) \models C_G A$,

it follows that $(I, r, m) \models (K_i K_j)^l K_{ik} \ldots K_{i1} A$

by (a) of theorem 1, where the role of A is played by $K_{ik\text{-}1} \ldots K_{i1} A$, it follows that $\langle i_k, j, i, \ldots, j, i \rangle$ is a process chain in $(r, 0..m)$, where there are l occurrences of j, i in this chain. By Lemma 1, at least $2l$

messages must be sent in round r between rounds 1 and m. But this contradicts our assumption that exactly l messages are sent. Thus common knowledge cannot be gained.

The proof that common knowledge cannot be lost proceeds along identical lines, using part (b) of Theorem 1.

Remark. It can be shown that Theorem 1 and Theorem 2 can be proved with almost no change in the proofs for reliable systems. This shows that reliability does not play an important role in these results. It is the fact that there is no bound on message delivery that is crucial here.

Theorem 1 and lemma 1 can prove a number of lower bounds on number of messages required to solve certain problems.

Example 3. (Mutual exclusion) Intuitively, it means that there is a shared resource but only one process may acces the resource at a time.

We say that a.m.p. system R is a *system with mutual exclusion* if in every run of R, no two processes have simultaneously access to the resource.

It can be shown that for a system R with mutual exclusion and a run r in R in which processes $i_1, \ldots, i_k$ in

sequence have access to the shared resource. If we assume that for every j, $1 \leq j < k$ the proccesses i_j and i_{j+1} are different, then $< i_1 , \ldots , i_k >$ is a process chain in r.

By lemma 1 this implies that at least k-1 messages are sent in r. This gives us a lower bound on the number of messages required for mutual exclusion: for k processes to acquire access to a shared resource, at least k-1 messages should be sent.