Simultaneous Byzantine Agreement

Motivation.

So far, common knowledge has served as a tool for proving impossibility results. Namely, the fact that there are no protocols solving the coordinated attack problem or for agreeing to disagree.

We now present a case in which common knowledge is used in a positive manner, as a tool for the design of efficient protocols.

The Coordinated Attack problem deals with the impact that unreliable communication has on coordination in multi-agent protocols.

Another major source of difficulty in distributed system is the fact that processes may fail during the execution of a protocol.

This can cause particular difficulties when it comes to coordinating actions between different sites in such a system.

We do not want two sites in an airline reservation system to sell the same seat to two different people.

A bank must ensure that every transaction made at one of its automated tellers is appropriately recorded in its central database.

Because the components of such a system do fail *occasionally* for various reasons, it is important to program them in such a way that the overall behaviour of the system will not be jeopardized by the failure of a small number of its components.

The paradigmatic problem concerning reaching agreement at different sites in a system in the presence of failures is the *Byzantine agreement problem.*

This problem can be informally described as follows.

We have n generals, of which at most $t < n$ might be traitors. Each general initially prefers either attack or retreat (although they are willing to do something other than what they initially prefer) .

At the end of the protocol, they must reach agreement, so that the loyal generals either all attack or all retreat.

The traitors can do whatever they like, we have no control over them.

Although the generals can talk to each other (over reliable channels), there is no broadcast facility. It is not possible for a general to take a loudspeaker and announce his vote to all the others.

There is a trivial protocol satisfying these conditions : one where the generals retreat, no matter what their initial preference.

In practice, however, this is not a satisfactory solution.

A natural additional property to demand is that if all generals initially prefer to do the same thing (either attack or retreat), this is what they will agreed to do.

This condition eliminates trivial solutions such as retreating no matter what.

If we are guaranteed that there are no traitors in the system , then reaching agreement is trivial.

The generals all talk to each other , find out what each of them initially prefers to do, and use some uniform rule for reaching the a decision (for example, the rule might be to attack if any general initially prefers to attack).

In the presence of traitors, however, the situation becomes considerably more complicated.

What to do if a traitor tells one loyal general that he wants to attack and tells another that he wants to retreat ?

This problem turns out to be remarkably subtle and has been studied at length in the literature.

We focus here on one particular variant of the problem, where we require that not only do the generals decide, but that they decide simultaneously. We refer to this variant as the *Simultaneous Byzantine agreement* (*SBA*) problem.

Just as in the case of Coordinated Attack, we can show that in a large class of interpreted contexts, the requirement of simultaneity here leads to common knowledge.

We now define this class of interpreted contexts, which we call *ba-compatible.* These ba-compatible contexts share many of the features of ca-compatible contexts.

Again, we want to make minimal assumptions about that processes` (i.e. generals`) local states and have environment‘s state record the actions performed.

The main difference is that we now want the environment to specify which processes are faulty and how they are faulty.

This means that there are more possible actions that the environment performs.

For now, we do not go into details of what these actions are. We will turn to it later.

Definition. (ba-compatible contexts)

Formally, we say that an interpreted context (γ , π) is *ba-compatible* if it satisfies the following assumptions:

(1) For each process $i = 1, \ldots , n$ one of i's actions is denoted $\mathbf{decide}_i(y)$, for $y = 1$ or $y = 0$. We can think that $\mathbf{decide}_i(0)$ as representing a decision to retreat, while $\mathbf{decide}_i(1)$ represents a decision to attack.

(2) The environment´s actions include ones of the form $(\mathbf{a}_{e1} , \ldots , \mathbf{a}_{en})$.

(3) The components $\mathbf{a}_{ei}$ themselves are tuples, which describe which messages sent by the process j are delivered to process i in that round, whether or not i fails in that round, which we capture by a $\mathbf{fail}_i$ action, and its faulty behaviour if it does **fail**.

We discuss later on how this faulty behaviour is described, since this depends on the type of faulty behaviour we allow.

(4) We say that process i *fails in round* k *of run* r if the environment‘s action at this round has a $\mathbf{fail}_i$ component.

(5) We say that process i is *faulty* in round k of run r (or at the point (r , k)) if process i failed in some round $k‘$ up to k of run r. Otherwise, we say that i is *nonfaulty* or *correct.*

(6) γ is a recording context. Note that recording context allows us to tell from the environment‘s state which processes are faulty, by seeing which $\mathbf{fail}_i$ actions have been performed.

(7) Process i‘s initial state is a tuple of the form $(x_i , \ldots .)$ where x_i is either 0 (retreat) or 1 (attack) . Thus, x_i represents i‘s initial preferences.

(8) We also assume that process i‘s local state records whether process i has performed an action $\mathbf{decide}_i(y)$.

(9) The environment‘s state includes the tuple $(x_1, \ldots, x_n)$, where x_i is either 0 or 1 representing the processes initial preferences.

(10) The language includes the propositions $decided_i(y)$, $decided_N(y)$, and Ey, for all i and Boolean y.

We define π so that

$decided_i(y)$ is true if the action $\mathbf{decide}_i(y)$ was performed at any previous round,

$decided_N(y)$ is true if each nonfaulty process i has performed the action $\mathbf{decide}_i(y)$,

Ey is true if $x_i = y$ for some process i.

$deciding_N(y)$ is defined as an abbreviation of

$$\neg\, decided_N(y) \;\&\; \mathrm{O}\, decided_N(y)$$

(11) recall that the nonfaulty processes can be determined from the environment‘s local state.

Comment. All the results of this section hold even without the assumption (8), but it turns out to be convenient for results in later sections. We make no other assumptions about the form of the processes‘local states.

Definition. (Ba-compatible interpreted systems)

A *ba-compatible interpreted system* is one of the form $\mathbf{IS}^{rep}(P, \gamma, \pi)$ where P is a protocol and (γ, π) is a ba-compatible interpreted context.

Comment. In a ba-compatible interpreted system, we can talk about processes‘ initial preferences and their decisions, so it makes sense to talk about SBA.

As we said earlier, we intend to focus here on the *simultaneous* Byzantine agreement (SBA).

Definition. (The specifications σ^{sba})

The specification σ^{sba} is run-based. It is satisfied by all ba-compatible interpreted systems I such that each run satisfies

• *Decision:* every process i that is nonfaulty in r performs exactly one $\mathbf{decide}_i$ action in r,

• *Agreement:* the nonfaulty processes all decide the same value,

• *Validity:* if all processes have the same initial preference x, then all nonfaulty processes decide on the value x,

• *Simultaneity:* the nonfaulty processes all decide simultaneously, i.e. in the same round.

Comment.

•The first clause ensures that nonfaulty processes decide exactly once,

• the second one ensures that their decisions are in agreement,

• the third ensures the decision is related to the initial preferences in a nontrivial way, and

• the fourth guarantees that the decision is simultaneous.

Note that the third clause prevents trivial solutions such as one in which everyone simply always decides on the value 0 in the first round and halts.

Indeed, the third clause ensures that for any given y in $\{0 , 1\}$ the processes may decide on the value y only if at least one process had y as its initial preference.

Definition. (Protocols for *SBA*)

(i) We say that P is a *protocol for SBA* or that P *attains* *SBA*, in a ba-compatible interpreted context (γ, π) if P satisfies σ^{sba} in (γ, π).

(ii) If r is a run in $\mathbf{IS}^{rep}(P, \gamma, \pi)$, we say that P attains SBA *in* k *rounds in* r if $(I, r, k) \models \mathit{deciding}_N$.

(Note that this means that a nonfaulty process i actually performs the action $\mathbf{decide}_i$ in round $k + 1$.)

(iii) We say that P *attains* SBA *in* k *rounds* if P attains SBA in k rounds in all runs of $\mathbf{IS}^{rep}(P, \gamma, \pi)$.

Comment. It should be clear from the specifications and descriptions of the problems that SBA and Coordinated Attack are closely related.

Indeed, we can reformulate Coordinated Attack slightly to resemble SBA even more as follows:

• we assume that each general i has initial preference x_i regarding whether or not he would like to attack,

• we could then restate the coordinated attack problem by requiring that if both generals initially prefer to attack, then they should attack, while if both initially prefer to retreat, they should retreat.

• While this version of the coordinated attack is slightly different from the one, we considered, we can easily prove results analogous to Theorem 1. and Corollary 2. for it.

• In fact, it is easy to show that if P is a deterministic protocol for this modified version of the coordinated attack in an interpreted context (γ, π) allowing all four configuration of initial preferences, then $\mathbf{I}^{rep}(P, \gamma, \pi)$ satisfies σ^{ca}. (Excercise)

Despite these similarities, there are some significant differences between SBA and CA, at least in the contexts of most iterest to us.

• In Coordinated Attack, both generals are assumed to be reliable : the problem is with the communication links.

• In SBA, we are mainly interested in contexts where correct generals have no problem communicating. Thus, we focus on contexts where communication is *reliable* and *immediate*, so that a message is guaranteed to arrive in the same rou nd in which it is sent, provided that neither the sender nor the in tended recipient of the message is faulty.

• The problem in these contexts is not with communication, but with the faulty processes. [end of Comment]

The Byzantine agreement problem is sensitive to the type of faulty behaviour a faulty process can display. The literature has concentrated on three basic failure modes.

(1) ***Crash failures*** : a faulty process behaves according to the protocol, except that it may crash at some point, after which it sends no messages. In the round in which the process fails, the process may perform an arbitrary subset of the actions it is supposed to perform, according to the protocol it is following. In particular, it may send only a subset of messages the it is supposed to send according to its protocol.

(2) ***Omission failures*** : a faulty process behaves according to the protocol, except that it may omit to send or receive an arbitrary set of messages in any given round. We sometimes refer to this case as the *general-omission* failure mode.

(3) ***Byzantine failures*** : faulty processes may deviate from the protocol in an arbitrary fashion : they may „lie,“ send deceiving messages, and collude to fool the nonfaulty processes in the most malicious ways.
Comment. In practice,

• crash failures occur quite regularly, as a result of mechanical and electrical failures. Crash failures can be viewed as a restricted type of omission failures (a process omits to send all messages from a certain point on.

• Omissions failures are often result of communication problems. Omission failures in turn can be viewed as a restricted type of the Byzantine failures to be defined below.

• Byzantine failures represent the worst possible failures, where we can make no assumptions on behaviour of faulty processes.

• We model these failures in terms of the environment‘s actions.

What we may expect ? (A short review of results)

Assume that there are n processes and that at most t of them are in any time faulty.

(i) It is known that there are protocols that attain SBA in $t + 1$ rounds in all these modes, provided that communication is reliable and immediate.

(ii) In the case of Byzantine failures i.e. if failed processes may lie, there is a constraint on the relationship between the total number n of processes in the system and the upper bound t on the number of faulty processes:
There is a protocol for SBA in this case iff $n > 3t$.

(iii) Moreover,
there is no protocol that attains SBA in fewer than $t + 1$ rounds.

In fact, it is known that any protocol for SBA in one of these failure modes requires at least $t + 1$ rounds to attain SBA in runs where are no failures at all.

Comment. It might seem surprising that, even if we consider such relatively benign failures as crash failures, we still have to take $t + 1$ rounds to reach agreement even in runs where there are no faulty processes.
As the following example shows, the problem here, as in the case of Coordinated Attack is not what does happen, but what *might* happen.

Example 1. (Attaining SBA - the case of Crash Failures)

Suppose that $n = 3$ and $t = 1$, and we restrict to crash failures. Consider a run where all the processes have initial preference 0 and there are no failures.

By the Validity requirement, this means, that all the processes must decide on the value 0 .

Suppose that in round 1 every process sends its initial preference to every other process . Thus, at time 1 , all the processes know that every process initially preferred 0 .

We can represent the situation at time 1 in a run using a 3 x 3 table, where each column represents information that one process has at time 1 about the initial preferences x_1, x_2, and x_3 (see figure SBA 1).

A failed process is assumed to have no information, and we mark this by the letter X.

Otherwise an entry in the table can be either 0, 1, or #, where we use a # in row i of column j to represent the fact that j did not receive information about i's initial preference x_i (because process i crashed before sending j a message).

Since we are considering only crash failures, all the processes are telling truth here. This means that all the processes know at time 1 that they must all ultimately decide on the value 0.

Why are they not able to decide on the value right away ?

In the run, we just described, the situation is represented by a table whose all entries are 0 ; this is table T_1 in the figure SBA 1.

	1	2	3
x_1	0	0	0
x_2	0	0	0
x_3	0	0	0

T_1

	1	2	3
x_1	0	X	0
x_2	0	X	*
x_3	0	X	0

T_2

	1	2	3
x_1	0	X	0
x_2	1	X	*
x_3	0	X	0

T_3

x_1	0	0	0
x_2	1	1	1
x_3	0	0	0

T_4

…

x_1	1	1	1
x_2	1	1	1
x_3	1	1	1

T_{10}

Figure SBA 1.

Recall that the nonfaulty processes must all decide simultaneously.

In particular, although in the situation depicted by T_1 process 1 received a 0 from all processes, it is possible, as far as process 1 is concerned, that process 3 did not receive a message at all from process 2 ; this could happen if 2 crashed after sending a message to 1 , but before sending a message to 3 ; this situation is depicted by table T_2 .

Note that in T_1 process 1 does not consider it possible that process 2 told 3 that its initial preference is 1 . We are not allowing lying here ; this would require Byzantine failures .

Clearly, process 1 cannot distinguish T_1 from T_2 ; it has the same local state in both situations which is described by the first column of the table.

Now in the situation described by T_2 , process 3 , which did not get a message from process 2 , considers it possible that 2‘s initial preference x_2 is 1 , and that 2 passed x_2 on to process 1 .

Thus, in the situation described by T_2 , process 3 considers the situation described by T_3 possible.

Finally, in T_3 process 1 does not know of any failure , and considers T_4 a possibility.

Notice that our tour from T_1 to T_4 involved „silencing“ process 2 , changing its initial preference x_2 , and „reviving“ the process. By applying this type of reasoning it is possible to do the same to process 3 and then to process 1 .

As a result, we can construct a sequence of tables T_1 , … , T_{10} such that T_1 tis the table all of whose entries are 0 and T_{10} is a table whose all entries are 1 .

For each consecutive pair of tables T_i , T_{i+1} , with $i < 10$, there is some process that cannot distinguish the situation described by T_i and T_{i+1} and is correct in both these situations .

Now suppose that some correct process decides at time 1 on the value 0 in the situation described by table T_1 . By the agreement and simultaneity requirements of SBA , all the processes must decide at time 1 on the value 0 in this situation.

Since process 1 cannot distinguish the situation described by T_1 from T_2 , process 1 must decide at time 1 on the value 0 in the situation described by T_2 . Again, by the agreement and simultaneity requirements, the two processes that are correct in this case (1 and 3) must decide at time 1 on the value 0 in the situation described by T_3 .

Continuing this argument, we get that all processes decide at time 1 on the value 0 in the situation described by T_{10} .

But in the situation described by T_{10} , all the processes are correct and have 1 as their initial preference. The fact that they decide on the value 0 in T_{10} contradicts the validity requirement ! (End of Example 1.)

Comment. As should be clear from our discussion, simultaneity plays a crucial role here. Our claim would not hold had we not require simultaneity.

To see why, suppose again that process 1 decides on value 0 at time 1 in the situation described by table T_1 .

Since process 1 cannot distinguish the situation described by T_1 from that described by T_2 , we know that process 1 also decides on the value 0 in the latter situation.

Without the requirement of simultaneity, we cannot, however, conclude that process 3 decides on the value 0 in the situation described by T_2 although we can conclude that process 3 will eventually decide on the value 0 in this situation. (This may, however, require further messages from process 1 .)

How many rounds are needed in the general case ?

We have stated (but not proved) that for any given number n of processes and an upper bound t, for the number of faulty processes that $t + 1$ rounds are required to attain SBA even if there are no failures.

It might seem that if we need $t + 1$ rounds, if there are no failures, things could be worse if there are failures.

We shall show that this is not the case.

In fact, by doing a knowledge-based analysis of SBA , we can completely characterize the number of rounds that are required to reach agreement.

Motivation. The validity requirement of SBA implies, that if all initial preferences are 1, then the nonfaulty processes should decide on the value 1.
In particular, for a process to decide on the value 0, the process must know that not all initial preferences are 1. Since the only possible initial preferences are 0 and 1, this says that for a process to decide on the value 0, the process must know that some initial preference is 0.

Of course, knowledge that some initial preference is v is not sufficient for a process to decide on the value v. Otherwise, a process could simply always decide on its initial preference, creating a violation of the agreement property in cases where two processes had different initial preferences.

In fact, Example 1. shows that even if a process knows that all initial preferences are 0, this is not sufficient to decide on the value 0.

What other knowledge do the processes need ?

A digression (coordinated Attack) Just as SBA requires simultaneity (namely simultaneous attacking), so the coordinated attack problem requires simultaneity, as well. (Namely simultaneous attacking.)

CA Corollary 1. of coordinated attack problem tells us that if the generals attack simultaneously, then the fact that they are both attacking must be common knowledge.

It is natural to expect that SBA to require attaining common knowledge as well. The question is, which group of processes actually attains common knowledge ?

It is not the set of *all* processes, since we do not place any require-ments on the actions of the faulty processes.

The SBA problem specification σ^{sba} requires only that the *nonfaulty processes* decide on the appropriate value. SBA therefore involves coordinating the actions of nonfaulty processes.

Sets of Nonfaulty Processes

Motivation. The set of nonfaulty proceses may change during each run.

Definition. (Rigid and varying sets)

Let R be a system over a set of global states of n processes. Let S be a function that gives to every point (r , m) a subset $S(r , m)$ of the set $\{1, 2, \ldots , n\}$ of processes. We assume that $S(r , m)$ *is a set of nonfaulty processes at point* (r , m) . By a misuse of terminology, *we shall think that the function* S *is a set varying in time.*

We say that S *is a rigid set* if it is constant, i.e. if there is a subset G of the set of processes such that $S(r , m) = G$ for every point (r , m) .

Otherwise, we say that S is a varying set.

Thus, we expect that the nonfaulty processes will need to attain common knowledge.

However, the set of nonfaulty processes is not fixed, but varies from one point to another. Hence the set of nonfaulty processes is a set described by a function $S(r , m)$.

We use the formula $i \varepsilon S$ to denote that i is in the varying set S .

We take $i \varepsilon S$ to be true at a point (r , m) if $i \varepsilon S (r , m)$.

Comment. Varying sets arise naturally in the analysis of various problems.

For example, when we consider a system in which processes can join and leave the system dynamically , the set of processes in the system is varying.

Similarly, the set of processes that have direct communication links to a given process in such a system is varying.

The varying set of most interest for us here is the set of nonfaulty processes, which we denote by N.

Thus, the set $N(r, m)$ *consists of all processes that are not faulty at the point* (r, m).

Motivation. Before we can relate SBA to common knowledge, we need to extend the definition of common knowledge to varying sets.

Given a varying set S, a natural candidate would be

$$E_S\varphi \text{ defined as } \bigcap_{i \varepsilon S} \varphi$$

According to this definition, $E_S\varphi$ would hold at a point (r, m) if all the processes in $S(r, m)$ know φ at (r, m).

C_S would then be defined in terms of E_S as usual.

But note that in a formula $E_S E_S \varphi$, or even $K_i E_S \varphi$ the value that the varying set S takes on may change.

For example, in evaluating the truth of $K_i E_s \varphi$, the value of S may be different at different points that i considers possible.

How can we judge whether the proposed definitions of E_s and C_S are appropriate ?

One criterion that is important for our application is whether we can prove that if the members of S have coordinated their actions, then this fact is guaranteed to be common knowledge among the members of S .

Does the definition of C_S we have just given have this property ?

For example, in SBA , it is necessarily the case that when a nonfaulty process decides, then it is common knowledge among the nonfaulty processes are deciding ? We would expect this to be the case, by analogy with the situation for coordinated attack (CA Proposition 1).

Remark. It can be shown, if a nonfaulty process is guaranteed to know that it is nonfaulty, then there is indeed such a common knowledge among the nonfaulty processes.

But in general, a process does not know whether it is faulty (at least, not in the round that it fails).

The consequences of this lack of knowledge can be easily seen in the case of general-omission failures.

In this case, it is not difficult to construct a run in which a nonfaulty process decides on the value 0 and yet considers it possible that the nonfaulty processes are deciding on the value 1. This is a consequence of the fact, that the process does not know whether it or the other processes are faulty. (Excersise)

We now define a notion of common knowledge that is appropriate even when the nonfaulty processes do not necessarily know that they are nonfaulty.

Let us go back to the Example 1. Note that while the nonfaulty process in the example does not know that the nonfaulty processes are all deciding on the value 0 , it might know that *if it is nonfaulty* , then they are all deciding on the value 0 .

This motivates the following :

Definition. (Processes‘ believes)

Given a varying set S and a process i, define $B_i^S \varphi$ to be an abbreviation for K_i(i ε S -> φ) . Thus,

$(I, r, m) \models B_i^S \varphi$ iff $(I, r', m') \models \varphi$ for all points (r', m') such that $r_i(m) = r_i'(m')$ and i ε S(r', m')

Comment. $B_i^S \varphi$ holds iff if i knows that *if it is in* S , then φ holds .

It is easy to check that B_i^S satisfies the axioms of S5, except for the Knowledge Axiom ($B_i^S \varphi$ -> φ).

Nevertheless, the Knowledge Axiom is satisfied at points where i is in the varying set S . That is,

$$\text{if } i \in S(r', m') \text{ then } (I, r, m) \models B_i^S \varphi \rightarrow \varphi$$

In general, it may be better to view B_i^S as a notion of *belief* rather than knowledge.

Corresponding to the varying set S , we add new modal operators E_S and C_S .

Definition. (E_S and C_S)

(i) We define $E_S \varphi$ as $\bigcap_{i \varepsilon S} B_i^S \varphi$. Thus,

$$(I , r , m) \models E_s \varphi \text{ iff } (I , r , m) \models B_i^S \varphi \text{ for all } i \varepsilon S (r , m)$$

Comment. In other words *everyone in* S *knows* φ at the point (r , m) exactly if every process in $S (r , m)$ knows that *if it is in* S, then φ holds.

Note that if $S (r , m)$ is empty, then by definition $E_S \varphi$ holds.

The notion $C_S \varphi$ is now defined as an infinite conjunction in terms of $E_S \varphi$.

(ii) Defining $E_S{}^{k+1} \varphi$ inductively as $E_S E_S{}^k \varphi$ we have

$$(I , r , m) \models C_s \varphi \text{ iff } (I , r , m) \models E_s^k \varphi \text{ for } k = 1, 2, ...$$

Comment. It is easy to see that if S is fixed, so that $S(r, m) = G$ for some set of processes and for all points (r, m), then $C_S\varphi$ <-> $C_G\varphi$.

Thus this definition extends our original definition of $C_G\varphi$ to the case of varying sets.

Let us now reconsider the case where a nonfaulty process is guaranteed to know that it is nonfaulty. In this case, when S is the varying set of nonfaulty processes, it is clear that if process i is nonfaulty, then $B_i^S\varphi$ is equivalent to $K_i^S\varphi$, for every formula φ.

Consequently, results we obtain later (Theorem 1. and Corollary 1.) would hold also in the case had we used the first definition of common knowledge for non-rigid sets.

As in the case of C_G we can relate $C_S\varphi$ to a notion of reachability.

Definition. (*S*-reachability)

(i) We say that a point (r', m') is *S-reachable from a point* (r, m) *in* k *steps* $(k > 0)$ if there exists a sequence of points

$$(r, m) = (r_0, m_0), (r_1, m_1), \ldots, (r_k, m_k) = (r', m')$$

such that for all l, $l < k$, there exists $i \in (S(r_l, m_l) \cap S(r_{l+1}, m_{l+1}))$ with $(r_l, m_l) \sim_i (r_{l+1}, m_{l+1})$.

(ii) We say that (r', m') is *S-reachable from* (r, m) if (r', m') is *S*-reachable from (r, m) in k steps for some k.

Now we get the following analogue of lemma on *G*-reachability:

Lemma 1. $(I, r, m) \models C_S\varphi$ iff $(I, r', m') \models \varphi$ for all points (r', m') that are *S*-reachable from (r, m).

Using lemma 1. we can also show that C_S satisfies many properties of the common knowledge operator C_G.

In particular, it satisfies all the axioms of S5 except possibly the Knowledge Axiom. It also satisfies the Fixed-Point Axiom and Induction Rule.

Moreover, if $S(r, m)$ is non-empty for all points (r, m) in an interpreted system I, then C_S satisfies the Knowledge Axiom in I as well.

Using C_N, we can get an analogue of CA Proposition 1. for SBA.

Theorem 1. *Let* (γ, π) *be a ba-compatible interpreted context and let* P *be a deterministic protocol. If* $I = \mathbf{I}^{rep}(P, \gamma, \pi)$ *satisfies* σ^{sba}, *then*

$$\textit{if } I \models \textit{deciding}_N(y) \textit{ then } C_N(\textit{deciding}_N(y))$$

Comment. As with the coordinated attack, Theorem 1. does not hold for nondeterministic protocols.

• But knowledge-based analysis shows that if we put some further restrictions on ψ in appropriate contexts, then we can extend the Theorem to nondeteministic protocols.

• Theorem 1. states that it is valid in a system I satifying σ^{sba} that whenever the nonfaulty processes decide on the value y, then it is a common knowledge among the nonfaulty processes, i.e. $C_N(deciding_N(y))$ holds.

• Knowing this, we are interested *how long* it will take to attain common knowledge in the contexts that are of interest to us for SBA.

• In the case of SBA, the processes are not allowed to decide on the value 1 when all initial preferences are 0. This implies that when they decide on the value 1 it must be the case that some process‘ initial preference was 1, as stated in the following Corollary 2.

Corollary 2. *Let* (γ, π) *be a ba-compatible interpreted context and let* P *be a deterministic protocol . If* $I = \mathbf{I}^{rep}(P, \gamma, \pi)$ *satisfies* σ^{sba} *, then*

$$\textit{if} \quad I \models \textit{deciding}_N(y) \quad \textit{then} \quad I \models C_N(\mathrm{E}y)$$

Comment. Neither Theorem 1. nor Corollary 2. would have held in the case of general-omission failures if we had used the first attempted definition of common knowledge for non-rigid sets.

But still, there are variants of SBA for which the first definition is appropriate.

Theorem 2. There are deterministic protocols that attain SBA in t +1 rounds in each of the contexts in Γ^{sba}.

Theorem 3. *If P is a deterministic protocol that satisfies σ^{sba}* in a context $\gamma \varepsilon \Gamma^{sba}$, *$r$ is a failure-free run in $\mathbf{R}^{rep}(P, \gamma, \pi)$, and P attains* SBA *in t‘ rounds in r, then t‘ > t.*

Attaining SBA

Corollary 2. shows that attaining common knowledge that some process had initial preference y is necessary in order to decide on the value y .

One of our goals here is to show that it is a sufficient condition as well , by describing a program that attains SBA by deciding which of $C_N(\text{E0})$ or $C_N(\text{E1})$ holds.

Notation. Let $decided_i$ be an abbreviation for

$$decided_i(0) \text{ v } decided_i(1)$$

so $decided_i$ is true if process i has made a decision.

Process' i's program would have the following form:

case of

if $\neg\, decided_i$ & $B_i^N C_N(\text{E0})$
. **do** $\text{decide}_i(0)$

if $\neg\, decided_i$ & $\neg\, B_i^N C_N(\text{E0})$ & $B_i^N C_N(\text{E1})$
. **do** $\text{decide}_i(1)$

if $\neg\, decided_i$ & $\neg\, B_i^N C_N(\text{E0})$ & $\neg\, B_i^N C_N(\text{E1})$
. **do** $\text{sendall}_i(\textit{local state})$

end case

Comment. Recall that the action sendall_i(*local state*) has the effect of sending each process other than i the message ℓ if process i's local state is ℓ .

Thus, messages in SBA are sent according to ***FIP*** (Full Information Protocol).

Note that since $B_i^N \varphi$ is an abbreviation for K_i($i \in N$ -> φ) , test such as $B_i^N C_N$(E0) and $B_i^N C_N$(E1) are indeed knowledge tests. Thus the above program is not a run-based program like standard programs are , but it is a knowledge-based program.

We argued informally that , provided that all nonfaulty processes do eventually decide , then a program of this form satisfies the specification σ^{sba} . Now, we can state it as a Theorem.

Theorem 4. *If* (γ, π) *is a ba-compatible interpreted context ,* I *is consistent with the program* SBA *in* (γ, π), *and* $C_N(\text{E0}) \vee C_N(\text{E1})$ *is attained in every run* r *of* I *, then* I *satisfies* σ^{sba} .

Moreover, the processes decide in a run r *of* I *at the round following the first time that* $C_N(\text{E0}) \vee C_N(\text{E1})$ *is attained.*