CHARACTERIZING DISTRIBUTED SYSTEMS USING KNOWLEDGE-BASED MODELS (Preliminary Version)

Deborah Shands
Dept. of Computer and Information Science
The Ohio State University
2036 Neil Avenue
Columbus, OH 43210
Email: shands@cis.ohio-state.edu

Chung-Kuo Chang
AT&T Bell Laboratories
6200 East Broad Street
Columbus, OH 43213
Email: chang@cblpe.att.com

ABSTRACT

We study a formal method for comparing distributed systems with respect to their abilities to solve various problems. To this end, we introduce a knowledge-based propositional modal language for axiomatically characterizing distributed systems and problems. Given a specification in the language, we show how to build a Kripke model so that a formula is true in the model exactly when it is provable using the axioms which specify the system. The models help us to formalize the description of the global observer's view of a system and show the effects of the global view on our ability to compare systems. An example shows that two distributed systems, running different protocols, are identical when the global view is restricted to a particular set of formulas, extracted from a problem specification. Under the unrestricted view, however, these systems appear quite different. We can generalize our comparisons by using a particular type of graph reduction between models to establish a relationship between seemingly dissimilar systems.

1 INTRODUCTION

Research results in distributed computer systems are based on the multitude of system models chosen by various researchers. A system model is comprised of a number of agents that communicate via some type of communication channels. The agents run a protocol (or program) that determines what they will compute and what messages they will send and receive. Characteristics of the system are usually stated informally. For example, we might assume that a distributed system consists of a set of fault-free processes, connected by unidirectional channels which do not corrupt or lose messages, but may reorder them. We could then provide a protocol to solve a particular distributed problem in our system. By changing one or more of the characteristics of our system (we might, instead, assume that channels could lose or corrupt messages), we get a second system. The behavior of the second system could be very different from that of the original system with respect to our chosen problem. Given a long list of system characteristics, it is difficult to see whether the solution to a problem in our original system has any bearing on the solvability of the same problem in the second system. It is equally difficult to determine the solvability of a problem in one system, given that a similar problem is solvable in the same system.

Several approaches have been taken to the comparison of distributed systems. [DDS87], for example, compares 32 systems characterized by various system parameters with respect to the distributed consensus problem. Possibility or impossibility results for each system are established individually through operational reasoning. [PSL80, FLP85] and [DLS88] take a similar approach to compare other systems with respect to the same problem. In [Wel87], Welch proves that a system with asynchronous processors and unknown message delay time can simulate synchronous processors, using a simple timestamping protocol. [NT87] defines the problems that can be solved using this simulation. [Coa88, NT88] and [ST87] describe compilers to generate functionally equivalent protocols for various systems. These works operationally establish relationships among systems.

In this paper, we propose a non-operational approach to comparing distributed systems by capturing system characteristics in a formal model. The model can then be used to reason about the corresponding system, disregarding our operational intuition. Systems can be compared by relating formal models.

Informally speaking, a *distributed system* is comprised of agents and communication channels. An *agent* is our smallest state-bearing entity—it may be a person, a discrete piece of processing machinery (a processor, node, neuron, etc.) or, perhaps, several persons or pieces of machinery working together to perform a specific function. A set of assumptions and requirements describes a *problem* to be solved in a particular distributed system. Given a problem and a system, a *protocol* prescribes the behavior of agents in the system with respect to the problem.

We provide an axiomatic specification language for describing distributed systems, protocols and problems. Specifications are based on knowledge—if agents have knowledge about the condition of other agents, then a protocol can be written to exploit this knowledge to coordinate the agents toward solving a given problem. Our understanding of knowledge is based on the works of Halpern and Moses who introduced the use of knowledge to analyze distributed systems (see [HM90].) [HM85] explains the Kripke semantics of a propositional language with knowledge operators, [HF85] describes knowledge-based protocols, while [CM86] considers the effects of system operations, such as message passing, on the knowledge of agents involved. These works contribute a general framework for using knowledge in reasoning about distributed systems; our work details a knowledge-based approach to building models of specific distributed systems.

We begin by introducing a propositional modal specification language, containing both temporal and knowledge operators. A particular distributed system, protocol and problem are specified by a set of logical axioms written in our language. The set of axioms could be used to deduce that the problem is solvable in the system, running the given protocol. Instead, we describe a method for using the set of axioms to derive a Kripke model which shows a formula to be true exactly when it is deducible from those axioms. This approach avoids some of the difficulties of deductive proofs and permits the use of a mechanical model checker, as described in [CES86] to determine which formulas are true in a given model.

A model defined by a set of axioms shows the truth value of every formula in the specification language, an infinite amount of information! In order to make sense of a system model, we must restrict our view of it. We define the global observer's view of a model by a set of formulas. Since a problem specification often restricts our view of a system, the formulas defining a view are usually derived from the problem requirement. Given a set of such formulas, we show how to filter a model to remove parts of the model

irrelevant to those formulas. An understanding of our view as global observer is essential to reasoning about differences or similarities we see among distributed systems, just as the local view of an agent is essential to reasoning about an agent's behavior in its distributed system. [Nei87] discusses the importance of an agent's local view, while our work demonstrates the importance of the global observer's view.

Once we have constructed formal models of a few distributed systems, we can compare the systems by showing relationships between the system models. In this paper, we show that two different distributed systems (defined by different axioms), running two different protocols are identical if we restrict our view of the systems to the set of formulas given by a particular problem specification. The unrestricted global view of these systems shows them to be quite different. In general, models of different systems will not appear identical. For that reason, we define a class of reductions from one model to another which allows us to formally relate two different systems.

2 THE SPECIFICATION LANGUAGE

In this section, we describe our specification language and associated logic. Distributed systems, protocols and problems are specified using sets of logical axioms from our language. Protocol axioms describe a solution to a given problem in a particular distributed system by relating the system specification to the problem specification. All system, problem and protocol axioms and the logic are common knowledge among system agents.

2.1 Syntax

$Wffs(\mathcal{B})$ is the set of well-formed formulas A, generated by the set of propositional variables $\mathcal{B}$ such that

$A ::= B_j \mid \neg A \mid A_1 \wedge A_2 \mid \oplus A \mid \ominus A \mid [F]A \mid [P]A \mid < F > A \mid < P > A \mid K_i A \mid CA,$

for $B_j \in \mathcal{B}$ and $1 \leq i \leq n$, where n is the number of agents in the system.

The symbol $\oplus$(respectively,$\ominus$) represents "at the next point (last point)." The formula [F]A ([P]A) means "from the next (last) point onward in the future (past), A holds." The dual formula, $< F > A$, ($< P > A$) represents "sometime in the future (past), A will hold. The formula K_iA means "agent P_i knows fact A " and the formula CA, "it is common knowledge that A." We assume that an agent knows a fact if the agent can use that fact in its reasoning.

We assume the usual abbreviations: $\vee$, $\rightarrow$, and $\leftrightarrow$, as well as those shown in Table 1.

Table 1: Abbreviations

Abbreviation	Formula	Explanation
K_i(x)	$\bigvee_{d \in \mathcal{D}} K_i$(x = d), for finite domain $\mathcal{D}$	P_i knows the value of x
$\oplus^n$A	$\underbrace{\oplus \oplus \cdots \oplus}_{n}$A	n points later A
$\ominus^n$A	$\underbrace{\ominus \ominus \cdots \ominus}_{n}$A	n points earlier A
EA	$\bigwedge_i K_i$A	everyone knows A
SA	$\bigvee_i K_i$A	someone knows A
$\Box$ A	[P]A $\wedge$ A $\wedge$ [F]A	always A
$\Diamond A$	$< P > A \vee$ A $\vee < F > A$	sometime A

2.2 Semantics

The semantics of $Wffs(\mathcal{B})$ is defined using models. A model is a Kripke structure (or graph) where each point (or vertex) is a set of formulas $s \subseteq Wffs(\mathcal{B})$. A valuation function V assigns a truth value to each formula in each point. The binary relations $\mathcal{R}_{\mathcal{F}}$ and $\sim_i$, $1 \leq i \leq n$ on the set $\mathcal{S}$ of points form the edges of the graph. A model provides no operational information, but rather describes relationships among all possible points.

Formally, a *model* is a triple $\mathcal{M} = (\mathcal{S}, \{\mathcal{R}_\mathcal{F}, \sim_i\}, V)$. For propositional variables, V: $\mathcal{B} \rightarrow \mathcal{P}(\mathcal{S})$, i.e. $V(B_j)$ is the set of points which contain B_j. We can now define the truth of a formula A $\in$ *Wffs*($\mathcal{B}$) at point s in model $\mathcal{M}$ by induction on the structure of A. The semantics of the $\neg$, and $\wedge$ operators is standard for propositional logic.

$\mathcal{M} \models_s B_j \ \ iff \ \ s \in V(B_j)$

$\mathcal{M} \models_s \neg A \ \ iff \ \ \mathcal{M} \not\models_s A$

$\mathcal{M} \models_s A_1 \wedge A_2 \ \ iff \ \ \mathcal{M} \models_s A_1 \ \ and \ \ \mathcal{M} \models_s A_2$

The semantics of the temporal and knowledge operators are as follows:

$\mathcal{M} \models_s \oplus A \ \ iff \ \ s\mathcal{R}_\mathcal{F}t \ \ implies \ \ \mathcal{M} \models_t A$

$\mathcal{M} \models_s \ominus A \ \ iff \ \ t\mathcal{R}_\mathcal{F}s \ \ implies \ \ \mathcal{M} \models_t A$

$\mathcal{M} \models_s K_i A \ \ iff \ \ s \sim_i t \ \ implies \ \ \mathcal{M} \models_t A$

Let $\mathcal{R}^*_\mathcal{F}$ be the transitive closure of the $\mathcal{R}_\mathcal{F}$ relation, then

$\mathcal{M} \models_s [F]A \ \ iff \ \ s\mathcal{R}^*_\mathcal{F}t \ \ implies \ \ \mathcal{M} \models_t A$

$\mathcal{M} \models_s [P]A \ \ iff \ \ t\mathcal{R}^*_\mathcal{F}s \ \ implies \ \ \mathcal{M} \models_t A$

$\mathcal{M} \models_s < F > A \ \ iff \ \ \exists t(s\mathcal{R}^*_\mathcal{F}t \ \ implies \ \ \mathcal{M} \models_t A)$

$\mathcal{M} \models_s < P > A \ \ iff \ \ \exists t(t\mathcal{R}^*_\mathcal{F}s \ \ implies \ \ \mathcal{M} \models_t A)$

Let $\sim$ be $\bigcup_i \sim_i$ and $\sim^*$ be the transitive closure of $\sim$, then

$\mathcal{M} \models_s CA \ \ iff \ \ s \sim^* t \ \ implies \ \ \mathcal{M} \models_t A$

$\mathcal{R}_\mathcal{F}$ is the "next time" relation: if $s\mathcal{R}_\mathcal{F}t$, then t is immediately after s and s is just before t. $\mathcal{R}_\mathcal{F}$ need not be transitive. $\sim_i$ is the set of relations defining the view of each agent P_i. $s \sim_i t$ iff P_i cannot distinguish between points s and t; P_i knows a fact A when it can distinguish points at which A is true from points at which A is false. Note that $\mathcal{M} \models_s EA \ \ iff \ \ s \sim t \ \ implies \ \ \mathcal{M} \models_t A$. To determine the truth of formulas K_1A, EA and CA at point s in model $\mathcal{M}$, we examine points t reachable from s. For K_1A, we look at points reachable from s, using the $\sim_1$ relation. For EA, we consider points reachable from s, using the $\sim$ relation. Finally, for CA, we look at points reachable from s, by the $\sim^*$ relation.

2.3 The Logic $\mathcal{L}$ and Specification Language *Spec*

In this section, we give the axioms and rules for our logic $\mathcal{L}$ and describe our language *Spec* for specifying systems, protocols, and problems. $\mathcal{L}$ includes all propositional tautologies, and the inference rules Modus Ponens, Necessitation, and Uniform Substitution. We also provide a rule for inferring the existence of common knowledge. Axiom schemata for $\mathcal{L}$ are shown in Table 2.

MP: $\dfrac{\vdash_\mathcal{L} A_1 \ , \vdash_\mathcal{L} A_1 \rightarrow A_2}{\vdash_\mathcal{L} A_2}$, *N*: $\dfrac{\vdash_\mathcal{L} A}{\vdash_\mathcal{L} CA}$, $\dfrac{\vdash_\mathcal{L} A}{\vdash_\mathcal{L} \Box A}$, *CInf*: $\dfrac{\vdash_\mathcal{L} A_1 \rightarrow E(A_1 \wedge A_2)}{\vdash_\mathcal{L} A_1 \rightarrow CA_2}$,

US: $\dfrac{\vdash_\mathcal{L} A_1}{\vdash_\mathcal{L} A_2}$, where A_2 is the result of uniformly substituting formulas for atomic formulas in A_1.

For technical reasons, we restrict our specification language *Spec* to those formulas of *Wffs*($\mathcal{B}$) in which the operators [F], [P], $< F >$, $< P >$, and C only appear positively (within the scope of an even number of $\neg$ symbols). For example, "$\neg\neg$[F]A" is in *Spec*, but "$[F]A \rightarrow A$" is an abbreviation for the formula "$\neg([F]A \wedge \neg A)$", which is not in *Spec*. Note that those axioms from Table 2 which have [F]A, [P]A, $< F >$A, $< P >$A, or CA on the left side of the $\rightarrow$ symbol do not satisfy the above requirement, so are not elements of *Spec*. In Section 3.1 we show an example of what could happen if our specification language contained all of the formulas of *Wffs*($\mathcal{B}$).

Table 2: Axiom Schemata of $\mathcal{L}$

Axiom 1	$\oplus(A_1 \rightarrow A_2) \rightarrow (\oplus A_1 \rightarrow \oplus A_2)$
	$\ominus(A_1 \rightarrow A_2) \rightarrow (\ominus A_1 \rightarrow \ominus A_2)$
	$K_i(A_1 \rightarrow A_2) \rightarrow (K_i A_1 \rightarrow K_i A_2)$
	$C(A_1 \rightarrow A_2) \rightarrow (CA_1 \rightarrow CA_2)$
	$[F](A_1 \rightarrow A_2) \rightarrow ([F]A_1 \rightarrow [F]A_2)$
	$[P](A_1 \rightarrow A_2) \rightarrow ([P]A_1 \rightarrow [P]A_2)$
Axiom 2	$K_i A \rightarrow A$
	$CA \rightarrow A$
Axiom 3	$K_i A \rightarrow K_i K_i A$
	$CA \rightarrow CCA$
Axiom 4	$\neg K_i A \rightarrow K_i \neg K_i A$
Axiom 5	$CA \rightarrow E(A \wedge CA)$
Axiom 6	$\oplus \neg A \leftrightarrow \neg \oplus A$
	$\ominus \neg A \leftrightarrow \neg \ominus A$
Axiom 7	$[F]A \rightarrow \oplus(A \wedge [F]A)$
	$[P]A \rightarrow \ominus(A \wedge [P]A)$
Axiom 8	$< F > A \rightarrow \oplus(A \vee < F > A)$
	$< P > A \rightarrow \ominus(A \vee < P > A)$
Axiom 9	$[F](A \rightarrow \oplus A) \rightarrow (\oplus A \rightarrow [F]A)$
	$[P](A \rightarrow \ominus A) \rightarrow (\ominus A \rightarrow [P]A)$
Axiom 10	$A \rightarrow [P] < F > A$
	$A \rightarrow [F] < P > A$

2.4 Example: The CSF System

We illustrate the use of our specification language *Spec* by describing the CSF system. In this system, two agents communicate synchronously and may fail by crashing. Agents which have not failed are said to be *working*. Agents compute synchronously, meaning that the local clock values of all working agents are identical. When the knowledge of one agent changes, the local clock value of every working agent is incremented by one. Agents either work or fail by crashing: a message received by an agent must have arrived; an agent sends only information that it knows to be correct. A crash failure permanently stops the local clock of the failed agent. Axioms CSF 1 to CSF 5 of Table 3 describe agent behavior in CSF.

Communication between agents is point-to-point via channels which are non-FIFO, but failure free. A message which is sent eventually arrives, and a message that has arrived was actually sent. We distinguish message arrival (i.e. message becomes available for receipt) from message receipt. Axioms CSF 6 to CSF 7 describe the channels of CSF.

Synchronous communication is defined using a commonly known upper bound on message arrival time, with messages received immediately upon arrival, barring the crash failure of the recipient, as described by CSF 8 and CSF 9.

Our example uses a simple information exchange problem. Assuming that (eventually) agent P_1 will know a fact A, our problem requires that (eventually) P_1 know that P_2 knows A. The formal statement is:

Problem 1: **assum:** $< F > K_1 \mathrm{A}$
req: $< F > K_1 K_2 \mathrm{A} \vee \text{failed}(1) \vee \text{failed}(2)$

Table 4 describes the protocol used to solve problem 1 in CSF. Agent P_1 is required to send to P_2 any formula that it knows. P_2, in turn, must acknowledge a message received from P_1. Once sent, formulas must stabilize, i.e. a formula that is sent, must remain true forever after.

Table 3: System Axioms of CSF

Agents: Synchronous with Crash Failures	
CSF 1	$\neg$failed(i) $\wedge$ $\neg$failed(j) $\rightarrow$ (*Clock*(i) = *Clock*(j)) $\wedge$ [P](*Clock*(i) = *Clock*(j))
CSF 2	$\neg K_i$A $\wedge$ $\oplus K_i$A $\leftrightarrow$ ((*Clock*(i) = a) $\rightarrow$ $\oplus$(*Clock*(i) = a + 1))
CSF 3	received(i, j, A) $\rightarrow$ arrived(i, j, A)
CSF 4	sent(i, j, A) $\rightarrow$ K_iA
CSF 5	failed(i) $\rightarrow$ ((*Clock*(i) = a) $\rightarrow$ [F](*Clock*(i) = a))
Channels: non-FIFO, pt-to-pt, no failures	
CSF 6	sent(i, j, A) $\rightarrow$ $< F >$ arrived(i, j, A)
CSF 7	arrived(i, j, A) $\rightarrow$ sent(i, j, A)
Communication: Synchronous	
CSF 8	C(n) $\wedge$ (sent(i, j, A) $\rightarrow$ $\oplus^n$arrived(i, j, A))
CSF 9	arrived(i, j, A) $\rightarrow$ (received(i, j, A) $\vee$ failed(j))

Table 4: Send-Acknowledge Protocol with Failures

PF 1	$K_1 A \rightarrow$ ($< F >$sent(1, 2, A) $\vee$ $< F >$failed(1))
PF 2	received(1, 2, A) $\rightarrow$ ($< F >$sent(2, 1, K_2A) $\vee$ $< F >$failed(2))
PF 3	received(2, 1, A) $\rightarrow$ K_1A
PF 4	sent(i, j, A) $\rightarrow$ [F]A

We can prove that problem 1 can be solved in CSF, running the protocol of Table 4, by deducing the problem requirement (**req**) from $\mathcal{L}$, the problem assumptions (**assum**), and the system and protocol axioms. □

For convenience, we collect the logic $\mathcal{L}$, a particular set of system and protocol axioms (SA and PA), and problem assumptions (**assum**) under the name Λ.

Definition 1 $\Lambda \equiv \mathcal{L} \cup SA \cup PA \cup$ **assum**

The deductive approach has disadvantages, however. The construction of a deductive proof requires ingenuity on the part of the prover, and we cannot rely on deductions to prove impossibility results: sometimes a problem requirement or its negation can be deduced from a set of axioms, but sometimes neither can be deduced.

3 BUILDING A MODEL OF A LOGIC

In this section, we describe a formal model of a distributed system derived from a set Λ of system, protocol and problem assumption axioms which shows a formula to be true exactly when it is provable from Λ. This allows us to make use of axiomatic specifications, while avoiding some of the difficulties associated with deductions.

The canonical model of a consistent, normal logic Λ is defined by the axioms of Λ so that a formula of *Spec* is true in the canonical model iff it is deducible from Λ. A canonical model, containing information about all the formulas in *Spec*, is far too comprehensive to be usable. Our view of a model must be restricted in order for us to make use of its information. We usually define a view focusing on formulas contained in a given problem requirement. For example, formulas to consider for a system running a protocol to solve problem 1 might include A, K_1A, K_2A, and K_1K_2A. In general, our canonical models should be filtered with respect to a set of formulas describing the global view, removing parts of the model irrelevant to the

set. The definitions and lemmas regarding canonical models and filtrations presented in Sections 3.1 and 3.2 can be found in [Gol87].

Unfortunately, the canonical model of any logic Λ is infinitely large, making it impossible to simply construct Λ's canonical model, then filter to get a small graph. Our approach is to build only those parts of the model necessary to ensure that filtration by some set yields the same result as building (in theory) the entire canonical model, then filtering.

3.1 The Canonical Model Defined by a Logic

Each point in the canonical model $\mathcal{M}^\Lambda$ of Λ is an infinite, Λ-consistent set $s \subseteq Spec$.

Definition 2 *A set* $s \subseteq$ Spec *is* Λ*-consistent if* $s \not\vdash_\Lambda false$.

Definition 3 describes restrictions on formulas in a point of $\mathcal{M}^\Lambda$.

Definition 3 *A set* $\Phi \subseteq$ Spec *is* Λ*-maximal if*

1. Φ *is* Λ*-consistent, and*
2. *if* $A \in$ Spec *and* $\neg A \in$ Spec, *then either* $A \in \Phi$ *or* $\neg A \in \Phi$
   otherwise, if $A \in$ Spec *and* $\Phi \cup \{A\}$ *is* Λ*-consistent, then* $A \in \Phi$.

The canonical model of Λ has a set S^Λ of points; each of those points $s \in S^\Lambda$ is a Λ-maximal set of formulas. Edges are added between points s and t of $\mathcal{M}^\Lambda$ based on the presence of modal operators in the formulas of s and t. For example, an R_F^Λ edge will exist from s to t when for all formulas A, $\oplus A \in s \rightarrow A \in t$. I.e. the R_F^Λ relation is constructed so that the $\oplus$ operator is modeled correctly, as per Section 2.2.

Definition 4 *The canonical model of a consistent, normal logic* Λ *is* $\mathcal{M}^\Lambda = (\mathcal{S}^\Lambda, \{\mathcal{R}_\mathcal{F}{}^\Lambda, \sim_i{}^\Lambda\}, \mathrm{V}^\Lambda)$, *where*

1. $\mathcal{S}^\Lambda = \{s \subseteq \text{Spec} : s \text{ is } \Lambda - maximal\}$,
2. $s\mathcal{R}_\mathcal{F}{}^\Lambda t \text{ iff } \oplus A \in s \Rightarrow A \in t$,
3. $s\sim_i{}^\Lambda t \text{ iff } K_i A \in s \Rightarrow A \in t$,
4. $\mathrm{V}^\Lambda(p) = \{s \in \mathcal{S}^\Lambda : p \in s\}$

Lemma 1 explains how to read the canonical model to determine which formulas are true at a given point. Formula A is true at point s exactly when $A \in s$. A formula is true in model $\mathcal{M}^\Lambda$ if it is true at every point in $\mathcal{M}^\Lambda$.

Lemma 1 *For* $A \in$ Spec *and* $s \in S^\Lambda$,

$$M^\Lambda \models_s A \text{ iff } A \in s;$$

Corollary 1 states that $\mathcal{M}^\Lambda$ is the correct model for system Λ, i.e. a formula is true in $\mathcal{M}^\Lambda$ iff it is deducible from Λ.

Corollary 1 $\mathcal{M}^\Lambda$ *determines* Λ, *i.e. for all* $A \in$ Spec,

$$M^\Lambda \models A \text{ iff } \vdash_\Lambda A;$$

At this point, we can explain the restrictions placed in Section 2.3 on formulas of *Spec*. For the sake of example, let's take a model $\mathcal{M}^\Lambda$, where for $s \in S^\Lambda$, $A \in s \Rightarrow A \in Wffs(\mathcal{B})$. Let $\{\neg CA\} \subseteq s$ and suppose that only for set $u \in S^\Lambda$ are there values of i such that $K_i A \in s \Rightarrow A \in u$ and that there is no $v \in S^\Lambda$ such that for any i, $K_i A \in u \Rightarrow A \in v$. By Definition 4, there must be edges $s\sim_i{}^\Lambda u$ and there are no edges $u\sim_i{}^\Lambda v$. We then have for all $t \in S^\Lambda$, $s \sim^* t \Rightarrow A \in t$. Since this is the semantic definition of common knowledge, $\mathcal{M} \models_s CA$. s is Λ-consistent and we assumed that $\neg CA \in s$, so $CA \notin s$, and Lemma 1 fails. Similar situations can be constructed, using the temporal operators.

3.2 Excluding Information Using Filtrations

In this section, we describe how to restrict our global view of a model $\mathcal{M}$, focusing on truth values of formulas in a set Γ. We begin by grouping points of $\mathcal{M}$ together, using subsets of Γ to define equivalence classes. Each equivalence class will constitute a point in the filtered model.

Definition 5 *Let $\mathcal{M} = (\mathcal{S}, \mathcal{R}, \mathrm{V})$ be a model, and $\Gamma \subseteq$ Spec be closed under subformulae. For each $s, t \in \mathcal{S}$ define the equivalence relation $\sim_\Gamma$ on $\mathcal{S} \times \mathcal{S}$ so that*

$$s \sim_\Gamma t \ \textit{iff for all}\ A \in \Gamma,\ \ M \models_s A \ \textit{iff}\ \ M \models_t A.$$

Definition 6 *Let a Γ-equivalence class $|s| = \{t \in \mathcal{S} : s \sim_\Gamma t\}$ and the set of Γ-equivalence classes $\mathcal{S}_\Gamma = \{|s| : s \in \mathcal{S}\}$*

Once the points of the filtered model have been fixed, the $\sim_i$ and R_F relations between the points are determined, as given by Definition 7. If an edge from point s to point t existed in the original model, then a similar edge will exist between points $|s|$ and $|t|$ of the filtered model. An edge from $|s|$ to $|t|$ can exist in the filtered model only if points s and t of the original model contain the appropriate modal formulas.

Definition 7 *A model $\mathcal{M}_\Gamma = (\mathcal{S}_\Gamma, \{\sim_{i,\Gamma}, R_{F,\Gamma}\}, \mathrm{V}_\Gamma)$ is a Γ-filtration of $\mathcal{M} = (\mathcal{S}, \{\sim_i, \mathcal{R}_\mathcal{F}\}, \mathrm{V})$:*

1. *if $s \mathcal{R}_\mathcal{F} t$, then $|s| \mathcal{R}_{F,\Gamma} |t|$;*
2. *if $s \sim_i t$, then $|s| \sim_{i,\Gamma} |t|$;*
3. *if $|s| \mathcal{R}_{F,\Gamma} |t|$, then for all A, if $\oplus A \in \Gamma$ and $\mathcal{M} \models_s \oplus A$, then $\mathcal{M} \models_t A$.*
4. *if $|s| \sim_{i,\Gamma} |t|$, then for all A, if $K_i A \in \Gamma$ and $\mathcal{M} \models_s K_i A$, then $\mathcal{M} \models_t A$.*

Lemma 2 states that, for the formulas of Γ, the original model $\mathcal{M}$ and the filtered model $\mathcal{M}_\Gamma$ are identical.

Lemma 2 *If $A \in \Gamma$, then for any $s \in \mathcal{S}$,*

$$M \models_s A \ \textit{iff}\ \ M_\Gamma \models_{|s|} A.$$

3.3 Constructing a Γ-Relevant Model

In this section, we show how to construct parts of $\mathcal{M}^\Lambda$ which are relevant to Γ, and then filter to get $\mathcal{M}^\Lambda_\Gamma$. In Definition 4, the set $\mathcal{S}^\Lambda$ is constructed using the formulas from *Spec*. To construct our Γ-relevant part of $\mathcal{M}^\Lambda$, we use only a subset of *Spec*. Note that once some model $\mathcal{M}$ is Γ-filtered, only points and edges determined by Γ remain. In constructing a restricted model of Λ which will later be filtered by Γ, we must add enough defining formulas to guarantee that S^Λ_Γ will be formed by the filtration. We must also include formulas which will force the appropriate edges (R^Λ_Γ and $\sim^\Lambda_\Gamma$) to be added to the filtered model. Our restricted model must, therefore, be built using the relevant modal formulas. Definition 8 defines the set of formulas $Spec_{\Lambda,\Gamma-rel}$ from which our restricted model will be built.

Definition 8 *The set of Λ, Γ-relevant formulas,* Spec $_{\Lambda,\Gamma-rel}$ *is constructed as follows. $A \in$* Spec $_{\Lambda,\Gamma-rel}$ *if:*

1. *$A \in \Gamma$, or*
2. *A is $K_i A_1$, or $K_i \neg A_1$ for $A_1 \in \Gamma$, or*
3. *A is $\oplus A_1$, or $\oplus \neg A_1$ for $A_1 \in \Gamma$, or*
4. *A is $\neg A_1$, for $A_1 \in$* Spec $_{\Lambda,\Gamma-rel}$ *and $\neg A_1 \in$* Spec

To define Λ,Γ-maximality, we modify clause 2 of Definition 3 to

2': if $A \in Spec_{\Lambda,\Gamma-rel}$ and $\neg A \in Spec_{\Lambda,\Gamma-rel}$, then either $A \in \Phi$ or $\neg A \in \Phi$

otherwise, if $A \in Spec_{\Lambda,\Gamma-rel}$ and $\Phi \cup \{A\}$ is Λ-consistent, then $A \in \Phi$.

Using the Λ,Γ-relevant formulas of Definition 8, we can now construct $\mathcal{M}^{\Lambda,\Gamma-rel}$.

Definition 9 $\mathcal{M}^{\Lambda,\Gamma-rel} = (\mathcal{S}^{\Lambda,\Gamma-rel}, \{\mathcal{R}_{\mathcal{F}}{}^{\Lambda,\Gamma-rel}, \sim_i{}^{\Lambda,\Gamma-rel}\}, \mathrm{V}^{\Lambda,\Gamma-\mathrm{rel}})$, *where*
$\mathcal{S}^{\Lambda,\Gamma-rel} = \{s \subseteq \mathrm{Spec}_{\Lambda,\Gamma-rel} : s \text{ is } \Lambda,\Gamma - maximal\}$, *and* $\mathcal{R}_{\mathcal{F}}{}^{\Lambda,\Gamma-rel}$, $\sim_i{}^{\Lambda,\Gamma-rel}$, *and* $\mathrm{V}^{\Lambda,\Gamma-\mathrm{rel}}$ *are as in Definition 4.*

Theorem 1 is fundamental to our approach, stating that the model formed by building $\mathcal{M}^{\Lambda,\Gamma-rel}$ and filtering by Γ is the same as the model defined by the Γ-filtration of $\mathcal{M}^{\Lambda}$.

Theorem 1 *For all* $A \in \Gamma$,

$$\mathcal{M}^{\Lambda,\Gamma-rel}_{\Gamma} \models A \text{ iff } \mathcal{M}^{\Lambda}_{\Gamma} \models A$$

This result shows the construction of $\mathcal{M}^{\Lambda}_{\Gamma}$ to be feasible. Note that the intermediate model $\mathcal{M}^{\Lambda,\Gamma-rel}$ does not have the nice canonical model property of Corollary 1. $\mathcal{M}^{\Lambda,\Gamma-rel}$ can be built and filtered mechanically, then a model checker can be used to determine which formulas are true in $\mathcal{M}^{\Lambda}_{\Gamma}$. The complexity of checking whether a formula is true in a model is very high, when operators like C, [F], and [P] are permitted. For example, to determine whether the formula "CA" is true at point s in some model, we must inspect every point t which is $\sim^*$-reachable from s. Since the $\sim^*$ relation is the transitive closure of the union of the $\sim_i$ relations, this could be a very time consuming process for large models. [HV89] and [HM] cover the complexity issues involved. We expect that filtered models will be small enough that the time needed for model checking will be reasonable.

Example (continued): In Figure 1, we show part of $\mathcal{M}^{\Lambda,\Gamma-rel}$, where Λ contains the axioms of CSF, and Γ is the set $\{K_1K_2A,\ K_2A,\ K_1A,\ A\}$. Note that the set $Spec_{\Lambda,\Gamma-rel}$ is very large, even for such a small Γ. The vertices of the graph in Figure 1 contain only a few of the formulas in points of $\mathcal{M}^{\Lambda,\Gamma-rel}$, and salient predicates have been added to suggest the other Λ-consistent formulas which should complete the points. Some of the edges of the R_F and $\sim_1$ relations are also shown. The dashed boxes show the equivalence classes into which Γ groups the vertices. On filtering this model with respect to Γ, we get the model $\mathcal{M}^{\Lambda,\Gamma-rel}_{\Gamma}$ (equivalently, $\mathcal{M}^{\Lambda}_{\Gamma}$), shown in Figure 2. The formula K_1K_2A is not true in the filtered model (since it is not true at every point in the model) and, therefore, by Corollary 1, Lemma 2 and Theorem 1 is not deducible from Λ. If we enlarged Γ to include the requirement of problem 1, $< F > K_1K_2\mathrm{A} \vee \text{failed}(1) \vee \text{failed}(2)$, we would see that problem 1 is solvable in CSF, running the send-acknowledge protocol PF. □

4 COMPARING SYSTEMS USING MODELS

Section 3 described a method of producing a non-operational model of a distributed system with a specified global view. To compare two such models, we look for a specific type of graph reduction from one model to another. A function f is a *p-morphism* from a model $\mathcal{M}_1$ to a model $\mathcal{M}_2$ when the following three conditions hold. First, if there is an edge between two points, s and t, of $\mathcal{M}_1$, then there must be a similar edge between f(s) and f(t) in $\mathcal{M}_2$. Second, if there is an edge from f(s) to a point u in $\mathcal{M}_2$, then there must be a point t in $\mathcal{M}_1$ such that there is a similar edge from s to t and f(t) = u. Finally, a formula is contained in a point s in $\mathcal{M}_1$ iff it is contained in f(s). If a p-morphism can be found between two models, then the truth value of any formula is the same at corresponding points in the two models. The p-morphism formally shows the relationship between models.

Example (continued): we now compare CSF to a system which communicates asynchronously with no failures (CAN), whose system axioms are shown in Table 5. CAN differs from CSF in the behavior of its agents (no failures are permitted in CAN) and the fact that there is no known bound on message delay. Channel behavior is identical in CSF and CAN. Running the send-acknowledge protocol of Table 6, CAN solves problem 2.

Problem 2: **assum:** $< F > K_1\mathrm{A}$
req: $< F > K_1K_2\mathrm{A}$

The requirement of problem 2 is stronger than that of problem 1. Since CSF cannot solve problem 2, CSF and CAN are not equivalent. Are the two systems similar in any respect? If we again restrict

Table 5: System Axioms of CAN

Agents: Synchronous with No Failures	
CAN 1	$\Box(Clock(\mathrm{i}) = Clock(\mathrm{j}))$
CAN 2	$\neg K_i\mathrm{A} \wedge \oplus K_i\mathrm{A} \leftrightarrow ((Clock(\mathrm{i}) = \mathrm{a}) \rightarrow \oplus(Clock(\mathrm{i}) = \mathrm{a} + 1))$
CAN 3	received(i, j, A) $\rightarrow$ arrived(i, j, A)
CAN 4	sent(i, j, A) $\rightarrow K_i$A
Channels: non-FIFO, pt-to-pt, no failures	
CAN 5	sent(i, j, A) $\rightarrow <F>$ arrived(i, j, A)
CAN 6	arrived(i, j, A) $\rightarrow$ sent(i, j, A)
Asynchronous Communication	
CAN 6	sent(i, j, A) $\rightarrow <F>$arrived(i, j, A)
CAN 7	arrived(i, j, A) $\rightarrow <F>$received(i, j, A)

Table 6: Send-Acknowledge Protocol without Failures

PN 1	$K_1A \rightarrow <F>$sent(1, 2, A)
PN 2	received(1, 2, A) $\rightarrow <F>$sent(2, 1, K_2A)
PN 3	received(2, 1, A) $\rightarrow K_1$A
PN 4	sent(i, j, A) $\rightarrow$ [F]A

our attention to truth values of formulas in the Γ defined in Section 3.3, then CSF and CAN (with their respective protocols) appear to be the same. Some points in the Λ, Γ-relevant model of CAN are shown in Figure 3. In general, different systems (defined by different axioms) will not appear identical. Only by carefully choosing the formulas of Γ can we make two different systems seem the same. The Γ-filtered models of CSF and CAN are identical, as shown in Figure 2, hence an immediate p-morphism is the identity function. □

5 CONCLUSIONS

The axiomatic specifications we introduced can uniformly describe the essential properties of a distributed system, protocol and problem. These specifications can then be used to build a formal model which shows whether the protocol solves the problem in the system, without referring to the operational details of the solution. We found it necessary to limit the size of our models, showing only those parts crucial to the solution of the stated problem. To do this, the global observer's view must be specified in advance, to determine how much and which parts of the model will be built. Using one view to restrict models of two systems, running different protocols, we showed that the systems are equivalent under that view.

Our method permits further comparisons of dissimilar systems and problems, using non-trivial p-morphisms to establish relationships between models. Such relationships can be used to determine whether the solvability of a problem in one system is related to its solvability in the other. A system hierarchy could be constructed, based on the the difficulty of problems each system is able to solve.

Acknowledgments

We would like to thank Sanjay Gadkari and Timothy Carlson for many helpful discussions. We are also grateful to Yoram Moses for his comments on an earlier draft of this paper.

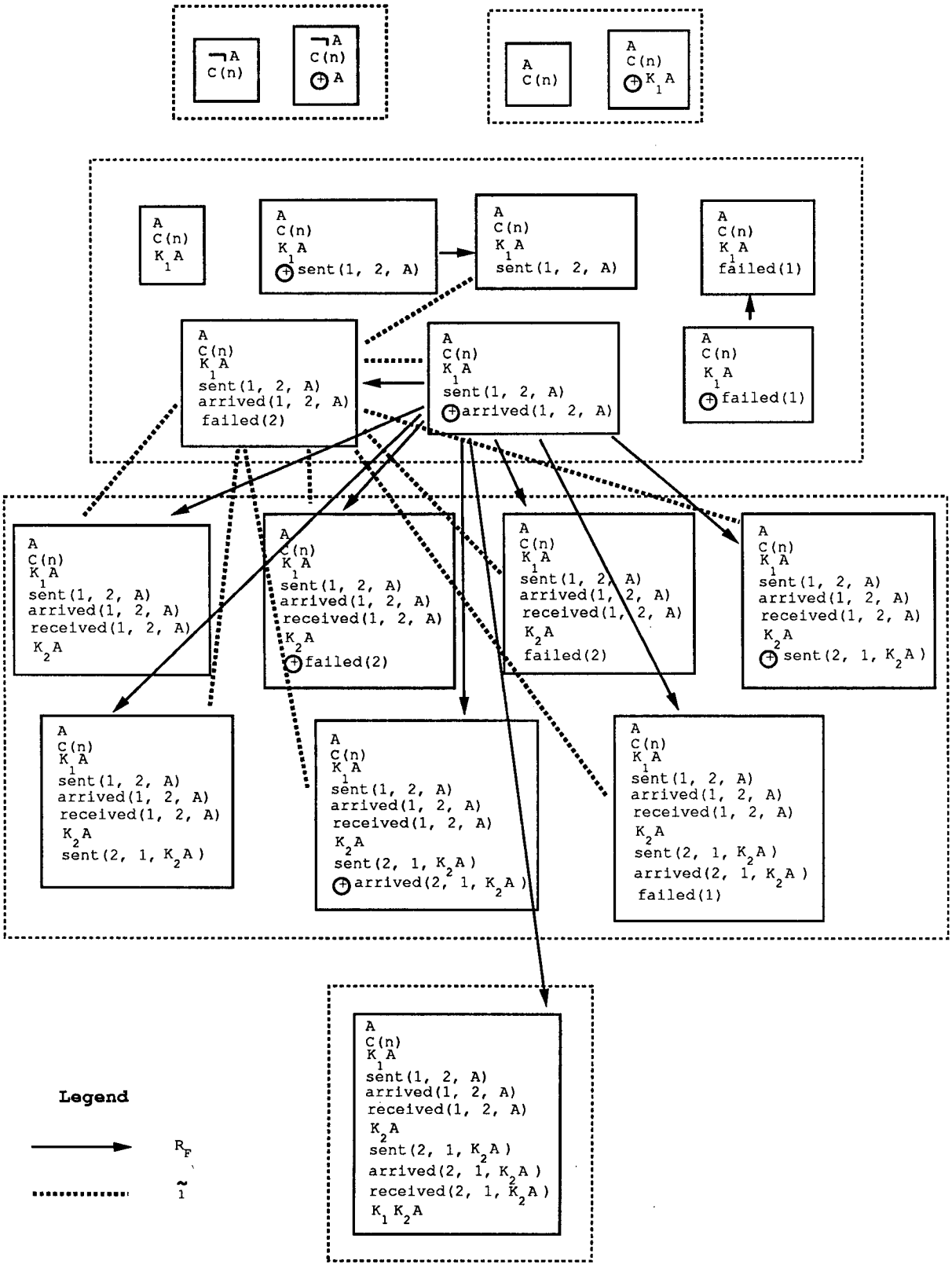

Figure 1: Part of $\mathcal{M}^{\Lambda,\Gamma-rel}$ for CSF

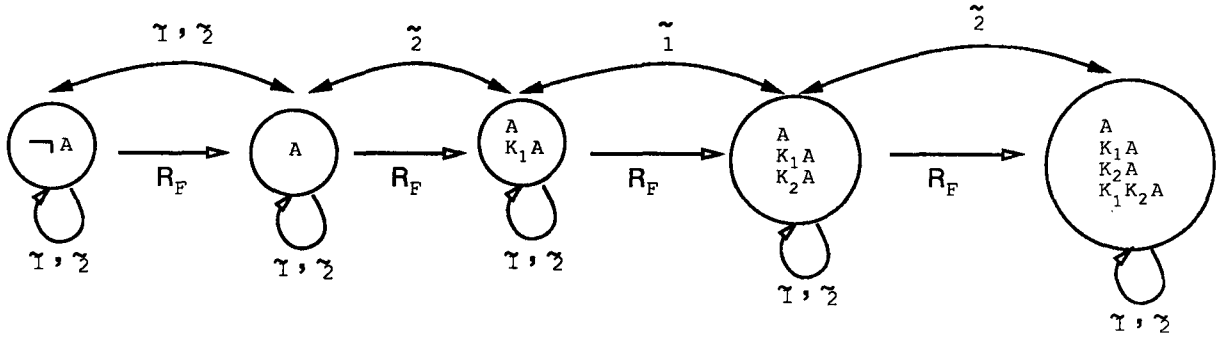

Figure 2: A Model of CSF, Filtered by Γ

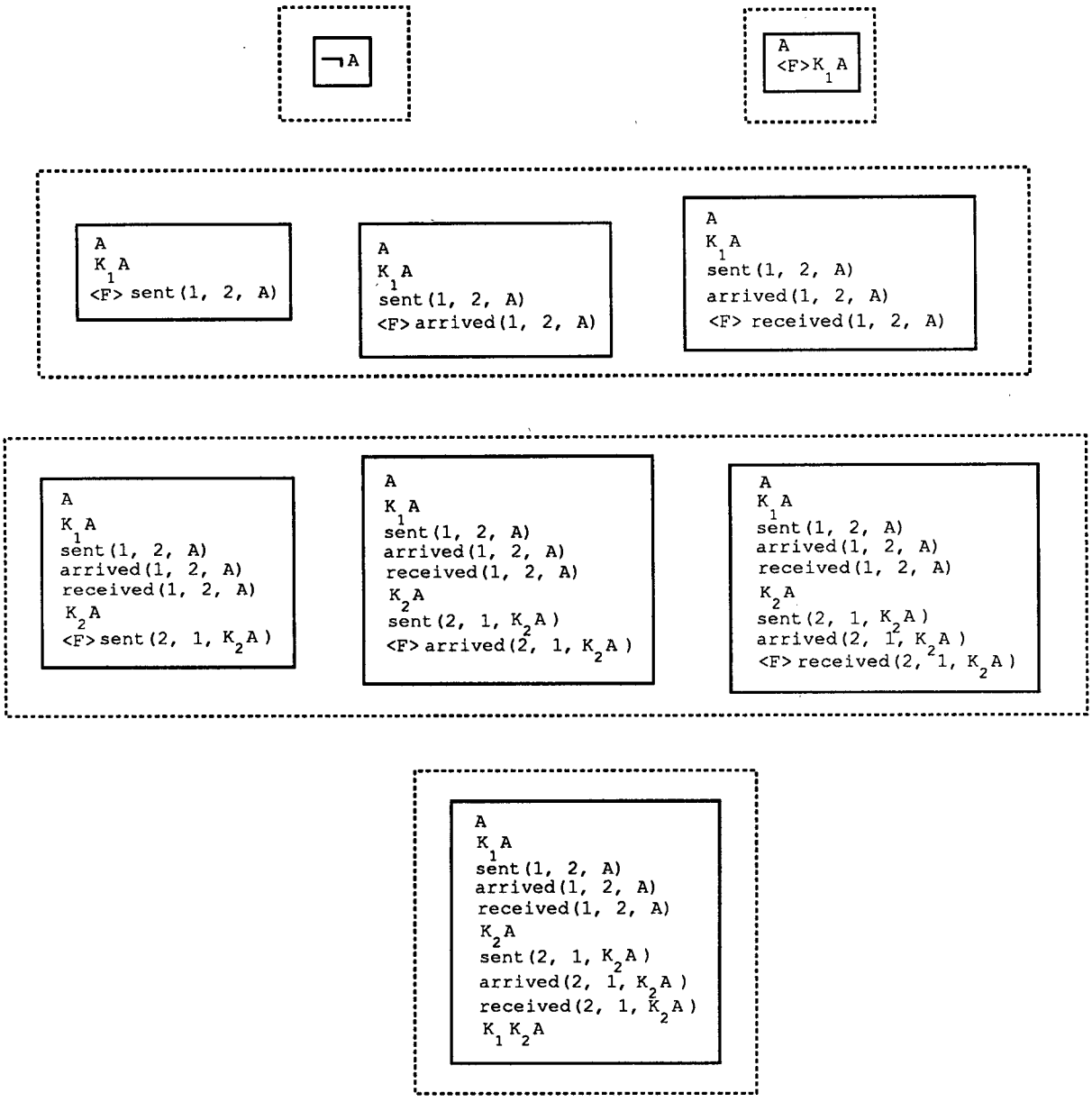

Figure 3: Part of $\mathcal{M}^{\Lambda,\Gamma-rel}$ for CAN

References

[CES86] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[CM86] K. M. Chandy and J. Misra. How processes learn. *Distributed Computing*, 1(1):40–52, 1986.

[Coa88] B. Coan. A compiler that increases the fault tolerance of asynchronous protocols. *ACM Transactions on Computer Systems*, 37(12):1541–1553, 1988.

[DDS87] D. Dolev, C. Dwork, and L. Stockmeyer. On the minimal synchronism needed for distributed consensus. *Journal of the ACM*, 34(1):77–97, 1987.

[DLS88] C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288–323, 1988.

[FLP85] M. Fischer, N. Lynch, and M. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, 1985.

[Gol87] R. Goldblatt. *Logics of Time and Computation.* CSLI/Stanford, 1987.

[GPSS80] D. Gabbay, A. Pnueli, S. Shelah, and J. Stavi. On the temporal analysis of fairness. In *Proc. of the 7th ACM Symp. on Principles of Programming Languages*, pages 163–173, 1980.

[HF85] J. Halpern and R. Fagin. A formal model of knowledge, action, and communication in distributed systems: Preliminary report. In *Proc. of the 4th ACM Symp. on Principles of Distributed Computing*, pages 224–236, 1985.

[HM] J. Halpern and Y. Moses. A guide to completeness and complexity for modal logics of knowledge and belief. (manuscript 1991).

[HM85] J. Halpern and Y. Moses. A guide to the modal logic of knowledge and belief. In *Proc. of the 9th International Joint Conference on Artificial Intelligence (IJCAI-85)*, pages 480–490, 1985.

[HM90] J. Y. Halpern and Y. Moses. Knowledge and common knowledge in a distributed environment. *Journal of the ACM*, 37(3):549–587, July 1990.

[HV89] J. Halpern and M. Vardi. The complexity of reasoning about knowledge and time, I: Lower bounds. *Journal of Computer and System Sciences*, 38(1):195–237, 1989.

[Leh84] D. Lehmann. Knowledge, common knowledge and related puzzles. In *Proc. of the 3rd ACM Symp. on Principles of Distributed Computing*, pages 62–67, 1984.

[Nei87] G. Neiger. Knowledge consistency: A useful suspension of disbelief. In *Proc. of the 2nd Conference on Theoretical Aspects of Reasoning about Knowledge*, pages 295–308, 1987.

[NT87] G. Neiger and S. Toueg. Substituting for real time and common knowledge in asynchronous distributed systems. In *Proc. of the 6th ACM Symp. on Principles of Distributed Computing*, pages 281–293, 1987.

[NT88] G. Neiger and S. Toueg. Automatically increasing the fault-tolerance of distributed systems: Preliminary version. In *Proc. of the 7th ACM Symp. on Principles of Distributed Computing*, pages 248–262, 1988.

[PSL80] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234, 1980.

[ST87] T. Srikanth and S. Toueg. Simulating authenticated broadcasts to derive simple fault-tolerant algorithms. *Distributed Computing*, 2:80–94, 1987.

[Wel87] J. L. Welch. Simulating synchronous processors. *Information and Computation*, 74:159–171, 1987.