Appendix F: Examples and Applications

F.1 Compatibility of Properties

If we represent the MAC and DAC policies as formal policies, as given below, we can examine the compatiblity of these properties.

Predicates:

own(s,o)	Subject s owns object o
grant_access(s,s',o)	Subject s grants access to object o to subject s'
read_access(s,o)	Subject s has read access to object o
write_access(s,o)	Subject s has write access to object o
dominates(s,o)	Security level of subject s strictly dominates security level of object o
equal(s,o)	Security level of subject s is equal to security level of object o

Properties:

$$\text{DAC} = \forall s : \text{Subjects} \cdot \forall o : \text{Objects} \cdot$$
$$\text{own}(s,o) \wedge \text{grant_access}(s,s',o) \Rightarrow \text{read_access}(s',o) \wedge \text{write_access}(s',o)$$

$$\text{MAC} = \forall s : \text{Subjects} \cdot \forall o : \text{Objects} \cdot$$
$$(\text{dominates}(s,o) \Rightarrow \text{read_access}(s,o) \wedge \neg \text{write_access}(s'o))$$
$$\wedge (\neg \text{dominates}(s,o) \Rightarrow \neg \text{read_access}(s,o) \wedge \text{write_access}(s,o))$$
$$\wedge (\text{equal}(s,o) \Rightarrow \text{read_access}(s,o) \wedge \text{write_access}(s,o))$$

$$\text{MAC} \wedge \text{DAC}$$
$$= \forall s : \text{Subjects} \cdot \forall o : \text{Objects} \cdot$$
$$(\text{own}(s,o) \wedge \text{grant_access}(s,s',o) \Rightarrow \text{read_access}(s',o) \wedge \text{write_access}(s',o))$$
$$\wedge (\text{dominates}(s,o) \Rightarrow \text{read_access}(s',o) \wedge \neg \text{write_access}(s',o))$$
$$\wedge (\neg \text{dominates}(s,o) \Rightarrow \neg \text{read_access}(s',o) \wedge \text{write_access}(s',o))$$
$$\wedge (\text{equal}(s,o) \Rightarrow \text{read_access}(s',o) \wedge \text{write_access}(s',o))$$
$$= \forall s : \text{Subjects} \cdot \forall o : \text{Objects} \cdot$$
$$((\text{own}(s,o) \wedge \text{grant_access}(s,s',o) \Rightarrow \text{read_access}(s',o) \wedge \text{write_access}(s',o))$$
$$\vee ((\text{dominates}(s,o) \Rightarrow \text{read_access}(s',o) \wedge \text{write_access}(s',o))$$
$$\wedge (\neg \text{dominates}(s,o) \Rightarrow \text{read_access}(s',o) \wedge \text{write_access}(s',o))$$
$$\wedge (\text{equal}(s,o)) \Rightarrow \text{read_access}(s',o) \wedge \text{write_access}(s',o))))$$
$$\wedge (\text{dominates}(s,o) \Rightarrow \text{read_access}(s',o) \wedge \neg \text{write_access}(s',o))$$
$$\wedge (\neg \text{dominates}(s,o) \Rightarrow \neg \text{read_access}(s',o) \wedge \text{write_access}(s',o))$$
$$\wedge (\text{equal}(s,o) \Rightarrow \text{read_access}(s',o) \wedge \text{write_access}(s',o))$$

$= \forall s : Subjects \cdot \forall o : Objects \cdot$

$((own(s,o) \wedge grant_access(s,s',o) \Rightarrow read_access(s',o) \wedge write_access(s',o))$

$\wedge (dominates(s,o) \Rightarrow read_access(s',o) \wedge \neg write_access(s',o))$

$\wedge (\neg dominates(s,o) \Rightarrow \neg read_access(s',o) \wedge write_access(s',o))$

$\wedge (equal(s,o) \Rightarrow read_access(s',o) \wedge write_access(s',o)))$

$\vee ((dominates(s,o) \Rightarrow read_access(s',o) \wedge write_access(s',o))$

$\wedge (\neg dominates(s,o) \Rightarrow read_access(s',o) \wedge write_access(s',o))$

$\wedge (equal(s,o)) \Rightarrow read_access(s',o) \wedge write_access(s',o))$

$\wedge (dominates(s,o) \Rightarrow read_access(s',o) \wedge \neg write_access(s',o))$

$\wedge (\neg dominates(s,o) \Rightarrow \neg read_access(s',o) \wedge write_access(s',o))$

$\wedge (equal(s,o) \Rightarrow read_access(s',o) \wedge write_access(s',o)))$

$= \bot$

Assume that a given component C_i satisfies DAC

sat(C_i, DAC)

$= own(s,o) \wedge grant_access(s,s',o) \Rightarrow read_access(s',o) \wedge write_access(s',o)$

$= (own(s,o) \wedge grant_access(s,s',o) \Rightarrow read_access(s',o) \wedge write_access(s',o))$

$\vee ((dominates(s,o) \Rightarrow read_access(s',o) \wedge write_access(s',o))$

$\wedge (\neg dominates(s,o) \Rightarrow read_access(s',o) \wedge write_access(s',o))$

$\wedge (equal(s,o)) \Rightarrow read_access(s',o) \wedge write_access(s',o)))$

$\not\Rightarrow$ MAC

Assume that a given component C_i satisfies MAC

sat(C_i, MAC)

$\Rightarrow (dominates(s,o) \wedge own(s,o) \wedge grant_access(s,s',o)$

$\Rightarrow read_access(s',o) \wedge \neg write_access(s',o)))$

$\wedge (\neg dominates(s,o) \wedge own(s,o) \wedge grant_access(s,s',o)$

$\Rightarrow \neg read_access(s',o) \wedge write_access(s',o)))$

$\wedge (equal(s,o) \wedge own(s,o) \wedge grant_access(s,s',o)$

$\Rightarrow read_access(s',o) \wedge write_access(s',o)))$

$\not\Rightarrow$ DAC

F.2. Emergent Properties

This system is made up of two components, a multi-level system, B, (with *hi* and *lo*) and a single-level system, A, (*hi* only). System A is used to form a feedback loop over system B. The complete system is illustrated in Figure 11.1. This system is described in detail in [TLBS88].

F.2.1 GNI as an Emergent Property

The structure of system B is given in Figure F.1. The composite system has been shown to be GNI-secure [TLBS88]. B3 is defined as a multi-level security component and therefore system B is also multi-level. Even though components B2 and B3 are functionally identical, the output of B2 is rated hi, while the output of B3 is rated lo. Either the output of B2 must be *hi*, implying that the output of B3 must also be *hi*, or that the output of B2 must be *lo*, implying that the

connection of B2 and B3 is inadmissible. Either of these scenaria should be enough to disallow the *lo* output of B3.

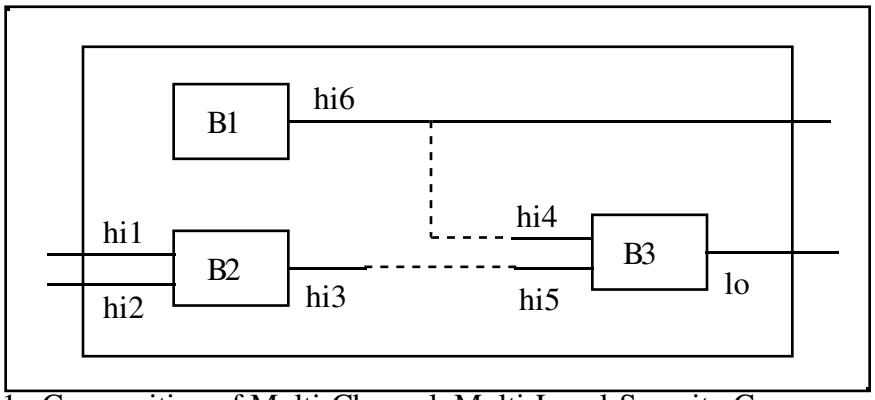

Figure F.1: Composition of Multi-Channel, Multi-Level-Security Components

The desired and actual behaviour of the individual components is defined as follows:

B1: P_1: $\forall x : \mathbb{N} \cdot \exists \chi : \Phi(hi,out) \cdot \sigma\{B1\}(out)[hi6] \models_x \chi$

B2: P_2: $\forall x : \mathbb{N} \cdot \forall \alpha,\beta : \Phi(hi,in) \cdot \exists \chi : \Phi(hi,out) \cdot$

$(\sigma\{B2\}(in)[hi1] \models_x \alpha \wedge \sigma\{B2\}(in)[hi2] \models_x \beta) \wedge \sigma\{B2\}(out)[hi3] \models_x \phi$

P_3: $\forall x : \mathbb{N} \cdot \forall \alpha,\beta : \Phi(hi,in) \cdot \exists \phi : \Phi(hi,out) \cdot \phi = \alpha \dot{\vee} \beta$

B3: P_4: $\forall x : \mathbb{N} \cdot \forall \alpha,\beta : \Phi(hi,in) \cdot \exists \delta : \Phi(lo,out) \cdot$

$(\sigma\{B3\}(in)[hi4] \models_x \alpha \wedge \sigma\{B3\}(in)[hi5] \models_x \beta) \wedge (\sigma\{B3\}(out)[lo] \models_x \delta)$

P_5: $\forall x : \mathbb{N} \cdot \forall \alpha,\beta : \Phi(hi,in) \cdot \exists \delta : \Phi(lo,out) \cdot \delta = \alpha \dot{\vee} \beta$

B3 as described is NOT GNI-secure: any change to the *lo*-level output can be attributed to a change in one of the two *hi*-level inputs.

Given the interconnections of B1, B2, and B3 defined in Figure F.1, the behaviour of the composite system $\mathbf{\&}^P(\mathbf{\&}^C(B_1,B_3),\mathbf{\&}^C(B_2,B_3)) = B123$ is given as

B(B123): B_6: $\forall x : \mathbb{N} \cdot \forall \alpha,\beta : \Phi(hi,in) \cdot \exists \delta : \Phi(lo,out) \cdot$

$(\sigma\{B123\}(in)[hi1] \models_x \alpha \wedge \sigma\{B123\}(in)[hi2] \models_x \beta \wedge \sigma\{B123\}(out)[hi6] \models_x \chi)$

$\Rightarrow (\sigma\{B123\}(out)[lo] \models_x \delta)$

B_7: $\forall x : \mathbb{N} \cdot \forall \alpha,\beta,\chi : \Phi(hi) \cdot \exists \delta : \Phi(lo,out) \cdot \delta = (\alpha \dot{\vee} \beta \dot{\vee} \chi)$

The goal of this system is for two *hi*-level inputs to be the exclusive-or of the *hi*-inputs and the *hi*-output, producing a single *lo*-level output. Represented formally, this gives the required properties of the composite system behaviour.

P(B123): P_6: $\forall x : \mathbb{N} \cdot \forall \alpha,\beta : \Phi(hi,in) \cdot \forall \chi : \Phi(hi,out) \cdot \exists \delta : \Phi(lo,out) \cdot$

$(\sigma\{B123\}(in)[hi1] \models_x \alpha \wedge \sigma\{B123\}(in)[hi2] \models_x \beta \wedge \sigma\{B123\}(out)[hi6] \models_x \chi)$

$\Rightarrow (\sigma\{B123\}(out)[lo] \models_x \delta)$

P_7: $\forall x : \mathbb{N} \cdot \forall \alpha,\beta,\chi : \Phi(hi) \cdot \exists \delta : \Phi(lo,out) \cdot \delta = (\alpha \dot{\vee} \beta \dot{\vee} \chi)$

The output δ does not give any information about the values of the high-level inputs. We may formalise this goal for system B123 in terms of GNI as follows:

GNI = P_8: $\forall x : \mathbb{N} \cdot \forall \alpha,\beta,\lambda : \Phi(hi,in) \cdot \forall \chi,\kappa : \Phi(hi,out) \cdot \exists \delta : \Phi(lo,out) \cdot$

$(\sigma\{B123\}(in)[hi1] \models_x (\alpha \vee \lambda) \wedge \sigma\{B123\}(in)[hi2] \models_x (\beta \vee \lambda)$

$\wedge \sigma\{B123\}(out)[hi6] \models_x (\chi \vee \kappa)) \Rightarrow (\sigma\{B123\}(out)[lo] \models_x \delta)$

The *hi*-level output χ can account for any changes caused by replacing a *hi*-level input with any other high-level input, including null. A *lo*-level user cannot determine if a change in δ is due to a change in the *hi*-level input or the *hi*-level output χ.

For the composite system, following the algorithms of chapter 10, we prove that GNI acts as an emergent property this (GNI-secure) composite system; GNI is gained on the composition of the individual components of this system.

$\bigcup_{i=1}^{\mathbb{C}} \mathbf{P}(C_i) = \mathbf{P}(B_1) \cup \mathbf{P}(B_2) \cup \mathbf{P}(B_3) = \{ P_1, P_2, P_3, P_4, P_5 \}$

$\bigcap_{i=1}^{\mathbb{C}} \mathbf{P}(C_i) = \varnothing$

PART I:

$\forall P_k : \{ P_1, P_2, P_3, P_4, P_5 \}$

$\mathbf{P}(\mathbb{C}) \cup= P_1 \wedge$ *satisfies*$(P_1,\mathbb{C})$

$\neg$ *relevant*$(P_2,\mathbb{C})$

$\neg$ *rrelevant*$(P_3,\mathbb{C})$

$\mathbf{P}(\mathbb{C}) \cup= P_4 \wedge$ *satisfies*$(P_4,\mathbb{C})$

$\mathbf{P}(\mathbb{C}) \cup= P_5 \wedge$ *satisfies*$(P_5,\mathbb{C})$

RESULT: $\mathbf{P}(\mathbb{C}) = \mathbf{P}(B123) = \{P_1, P_4, P_5\}$

PART II:

$\forall P_j : \mathbf{B}(B123) = \{P_6, P_7, P_8\}$

$P_6 \notin \mathbf{P}(\mathbb{C}) \vee \mathbf{P}$(similar systems)

$\mathbf{P}(\mathbb{C}) \cup= P_6 \wedge$ *satisfies*$(P_6,\mathbb{C})$

$P_7 \notin \mathbf{P}(\mathbb{C}) \vee \mathbf{P}$(similar systems)

$\mathbf{P}(\mathbb{C}) \cup= P_7 \wedge$ *satisfies*$(P_7,\mathbb{C})$

$P_8 \notin \mathbf{P}(\mathbb{C}) \vee \mathbf{P}$(similar systems)

$\mathbf{P}(\mathbb{C}) \cup= P_8 \wedge$ *satisfies*$(P_8,\mathbb{C})$

RESULT: $\mathbf{P}(\mathbb{C}) = \mathbf{P}(B123) = \{P_1, P_4, P_5, P_6, P_7, P_8\}$

satisfies$(P_8,\mathbb{C})$:

$\mathbf{B}(B123) = P_6 \wedge P_7$

$= \forall x : \mathbb{N} \cdot \forall \alpha,\beta : \Phi(hi,in) \cdot \forall \chi : \Phi(hi,out) \cdot \exists \delta : \Phi(lo,out) \cdot$

$[(\sigma\{B123\}(in)[hi1] \models_x \alpha \wedge \sigma\{B123\}(in)[hi2] \models_x \beta$

$\wedge \sigma\{B123\}(out)[hi6] \models_x \chi) \Rightarrow (\sigma\{B123\}(out)[lo] \models_x \delta)] \wedge \delta = (\alpha \dot{\vee} \beta \dot{\vee} \chi)$

Replace α by $(\alpha \vee \lambda)$, β by $(\beta \vee \lambda)$ and χ by $(\chi \vee \kappa)$

$= \forall x : \mathbb{N} \cdot \forall \alpha,\beta : \Phi(hi,in) \cdot \forall \chi : \Phi(hi,out) \cdot \exists \delta : \Phi(lo,out) \cdot$

$[(\sigma\{B123\}(in)[hi1] \models_x (\alpha \vee \lambda) \wedge \sigma\{B123\}(in)[hi2] \models_x (\beta \vee \lambda)$

$\wedge \sigma\{B123\}(out)[hi6] \models_x (\chi \vee \kappa)) \Rightarrow (\sigma\{B123\}(out)[lo] \models_x \delta)]$

$\wedge \delta = ((\alpha \vee \lambda) \dot{\vee} (\beta \vee \lambda) \dot{\vee} (\chi \vee \kappa))$

$= \forall x : \mathbb{N} \cdot \ \forall \alpha,\beta : \Phi(hi,in) \cdot \ \forall \chi : \Phi(hi,out) \cdot \ \exists \delta : \Phi(lo,out) \cdot$

$[(\sigma\{B123\}(in)[hi1] \models_x (\alpha \vee \lambda) \wedge \sigma\{B123\}(in)[hi2] \models_x (\beta \vee \lambda)$

$\wedge \sigma\{B123\}(out)[hi6] \models_x (\chi \vee \kappa)) \Rightarrow (\sigma\{B123\}(out)[lo] \models_x \delta)]$

$\wedge \delta = ((\alpha \dot{\vee} \beta \dot{\vee} \chi) \vee (\alpha \dot{\vee} \lambda \dot{\vee} \chi) \vee (\alpha \dot{\vee} \beta \dot{\vee} \kappa \ \vee (\alpha \dot{\vee} \lambda \dot{\vee} \kappa)$

$\vee (\lambda \dot{\vee} \beta \dot{\vee} \chi) \vee (\chi) \vee (\lambda \dot{\vee} \beta \dot{\vee} \kappa) \vee (\kappa))$

$\Rightarrow \forall x : \mathbb{N} \cdot \ \forall \alpha,\beta : \Phi(hi,in) \cdot \ \forall \chi : \Phi(hi,out) \cdot \ \exists \delta : \Phi(lo,out) \cdot$

$[(\sigma\{B123\}(in)[hi1] \models_x (\alpha \vee \lambda) \wedge \sigma\{B123\}(in)[hi2] \models_x (\beta \vee \lambda)$

$\wedge \sigma\{B123\}(out)[hi6] \models_x (\chi \vee \kappa)) \Rightarrow (\sigma\{B123\}(out)[lo] \models_x \delta)]$

$\wedge \delta = (\alpha \dot{\vee} \beta \dot{\vee} \chi)$

$\Rightarrow$ GNI

F.2.2 GNI as a De-Emergent Property

In this section we examine the composition of components A and B, both individually generalised non-interference secure (GNI-secure). The behaviour and properties of system B is described in the previous section (**B**(B123) = **P**(B123)). The behaviour and properties of component A are described by:

$\mathbf{B}(A) = \mathbf{P}(A) = \forall x : \mathbb{N} \cdot \ \forall \alpha : \Phi(hi) \cdot \ in(x) = \alpha \Leftrightarrow out(x) = \alpha$

If we connect the output of system B, out(hi6) with the input of system A, in(hiA), and the output of system A, out(hiA), with one of the high inputs of system B, in(hi1). The behaviour of the composite system is given as:

$\mathbf{B}(\&(B123,A)) = \mathbf{B}(B123) \wedge \mathbf{B}(A)$:

$\forall x : \mathbb{N} \cdot \ \forall \alpha,\beta : \Phi(hi,in) \cdot \ \exists \delta : \Phi(lo,out) \cdot$

$[(\sigma\{B123\}(in)[hi1] \models_x \alpha \wedge \sigma\{B123\}(in)[hi2] \models_x \beta$

$\wedge \ \sigma\{B123\}(out)[hi6] \models_x \chi) \Rightarrow (\sigma\{B123\}(out)[lo] \models_x \delta)]$

$\wedge \ \delta = (\alpha \dot{\vee} \beta \dot{\vee} \chi) \wedge \ [(\sigma\{A\}(in)[hiA] \models_x \chi \wedge \sigma\{A\}(out)[hiA] \models_x \alpha$

But, $\alpha = \chi$ by **P**(A)

$\forall x : \mathbb{N} \cdot \ \forall \alpha,\beta : \Phi(hi,in) \cdot \ \exists \delta : \Phi(lo,out) \cdot$

$[(\sigma\{B123\}(in)[hi1] \models_x \alpha \wedge \sigma\{B123\}(in)[hi2] \models_x \beta$

$\wedge \ \sigma\{B123\}(out)[hi6] \models_x \alpha) \Rightarrow (\sigma\{B123\}(out)[lo] \models_x \delta)] \wedge \ \delta = \beta$

$\wedge \ [(\sigma\{A\}(in)[hiA] \models_x \alpha \wedge \sigma\{A\}(out)[hiA] \models_x \alpha$

Writing this as two individual, independent behaviours,

$B_9 = \forall x : \mathbb{N} \cdot \ \forall \alpha,\beta : \Phi(hi,in) \cdot \ \exists \delta : \Phi(lo,out) \cdot$

$\sigma\{B123A\}(in)[hi2] \models_x \beta \wedge \sigma\{B123A\}(out)[lo] \models_x \delta$

$B_{10} = \forall x : \mathbb{N} \cdot \ \forall \alpha,\beta : \Phi(hi,in) \cdot \ \exists \delta : \Phi(lo,out) \cdot \delta = \beta$

The desired behaviour for this component is given by the following properties **P**(&(B123,A)):

$P_9 = \forall x : \mathbb{N} \cdot \ \forall \alpha,\beta : \Phi(hi,in) \cdot \ \exists \delta : \Phi(lo,out) \cdot$

$\sigma\{B123A\}(in)[hi2] \models_x \beta \wedge \sigma\{B123A\}(out)[lo] \models_x \delta$

$P_{10} = \forall x : \mathbb{N} \cdot \ \forall \alpha,\beta : \Phi(hi,in) \cdot \ \exists \delta : \Phi(lo,out) \cdot \delta \neq \beta$

The actual behaviour, **B**(&(B123,A)) does *not* entail the desired behaviour, **P**(&(B123,A)). This is easily determined by comparing the behaviours:

$B_9 \Rightarrow P_9 = \top$

$B_{10} \Rightarrow P_{10} = \bot$

Therefore, the goal of the behaviour of this system (treated as a pedantic property) is not satisfied by the actual behaviour of the system.

The informally described GNI-property of system B is applicable (relevant) to the composite system and will ideally act as a type 1 emergent property for the composite system. Assessing GNI for the system given the behaviour **B**(&(B123,A)), we see that the composite system does *not* satisfy this property.

$\mathbf{B}(\&(B123,A))$

$= \forall x : \mathbb{N} \cdot \ \forall \alpha,\beta : \Phi(hi,in) \cdot \ \exists \delta : \Phi(lo,out) \cdot$

$\quad \sigma\{B123A\}(in)[hi2] \models_x (\beta \vee \lambda) \wedge \sigma\{B123A\}(out)[lo] \models_x \delta \wedge \ \delta = (\beta \vee \lambda)$

$\not\Rightarrow$ GNI

There are no possible changes to the high-level output (there are none) that may account for any changes to the high-level inputs: all such changes are reflected in the low-level outputs. Thus GNI acts as a de-emergent property.

F.3 Progress

F.3.1 Specification of Components

Each event is defined as a predicate (for example, $gotF_i$) that may be true or false at any time during execution. If the predicate $gotF_i$ is true for component P_i, for example, this indicates that the philosopher P_i has possession of the component F_i. A P-component, P_i, must gain possession of the F-components F_i and $F_{i\oplus 1}$, where $i\oplus 1$ represents modular addition. We also use $i\ominus 1$ to represent modular subtraction.

Allowable Events:

Philosophers: $input P_i$ { eat, $gotF_i$, $gotF_{i\oplus 1}$ }

$output P_i$ { $getF_i$, $getF_{i\oplus 1}$, $releaseF_i$, $releaseF_{i\oplus 1}$, think }

Forks: $inputF_i$ { $getF_i$, $releaseF_i$ }

$outputF_i$ { $gotF_i$, null }

Input-output behaviour rules:

Philosophers: P_i(eat ; $getF_i$)

P_i($gotF_i$; $getF_{i\oplus 1}$)

P_i($gotF_{i\oplus 1}$; eating)

P_i(eating ; $releaseF_i \wedge releaseF_{i\oplus 1} \wedge$ think)

Forks: F_i($getF_i$; $gotF_i$)

F_i($releaseF_i$; null)

Component Behaviour Axioms:

Philosophers:

- A philosopher not possessing any forks and told to try to eat will attempt to obtain the fork to his left. He is still in the state of having no forks.

CB1 $\forall P_i : \{Philosophers\} \cdot \forall x : \mathbb{N} \cdot \ \sigma(\mathfrak{E};P_i) \models_x eat \Rightarrow \sigma(P_i;\mathfrak{E}) \models_x \Diamond getF_i$

- Once the philospher has received his left fork, he is in the state of having that fork and will attempt to get his right-hand fork.

CB2 $\forall P_i : \{Philosophers\} \cdot \forall x : \mathbb{N} \cdot$

$\quad \sigma\{P_i\} \models_x (\neg gotF_i \wedge \neg gotF_{i\oplus 1}) \wedge \sigma(\mathfrak{E};P_i) \models_x gotF_i \ \Rightarrow$

$\quad \sigma\{P_i\} \models_x O(gotF_i \wedge \neg gotF_{i\oplus 1}) \wedge \sigma(P_i;\mathfrak{E}) \models_x \Diamond getF_{i\oplus 1}$

- Once the philospher has his right fork, he is in the state of having both forks. He will begin eating.

CB3 $\forall P_i : \{\text{Philosophers}\} \cdot \forall x : \mathbb{N} \cdot$

$\sigma\{P_i\} \models_x (\text{gotF}_i \wedge \neg \text{gotF}_{i\oplus 1}) \wedge \sigma(\mathfrak{E};P_i) \models_x \text{gotF}_{i\oplus 1}$

$\Rightarrow \sigma\{P_i\} \models_x \bigcirc(\text{gotF}_i \wedge \text{gotF}_{i\oplus 1}) \wedge \sigma(P_i;P_i) \models_x \Diamond\text{eating}$

- A philosopher that is eating will eventually release both forks and return to thinking.

CB4 $\forall P_i : \{\text{Philosophers}\} \cdot \forall x : \mathbb{N} \cdot \sigma\{P_i\} \models_x (\text{gotF}_i \wedge \text{gotF}_{i\oplus 1}) \wedge \sigma(P_i;P_i) \models_x \text{eating}$

$\Rightarrow \sigma\{P_i\} \models_x \bigcirc(\text{gotF}_i \wedge \text{gotF}_{i\oplus 1})$

$\wedge \sigma(P_i;\mathfrak{E}) \models_x \Diamond(\text{releaseF}_i \wedge \text{releaseF}_{i\oplus 1} \wedge \text{think})$

Forks:

- A fork that is not granted, and receives a request for ownership will be granted and the fork will eventually acknowledge this granting to the philosopher.

CB5 $\forall F_j : \{\text{Forks}\} \cdot \forall x : \mathbb{N} \cdot (j=i \vee j=i\oplus 1) \wedge \sigma\{F_j\} \models_x \neg\text{granted} \wedge \sigma(\mathfrak{E};F_j) \models_x \text{getF}_j$

$\Rightarrow \sigma\{F_j\} \models_x \bigcirc\text{granted} \wedge \sigma(F_j;\mathfrak{E}) \models_x \Diamond\text{gotF}_j$

- A fork that is granted and possessed by a component will not be granted and will not be possessed by that component after that component sends a release fork request.

CB6 $\forall P_i : \{\text{Philosophers}\} \cdot \forall x : \mathbb{N} \cdot$

$(j=i \vee j=i\oplus 1) \wedge \sigma\{F_j\} \models_x \text{granted} \wedge \sigma(\mathfrak{E};F_j) \models_x \text{releaseF}_j$

$\Rightarrow \sigma\{F_j\} \models_x \bigcirc\neg\text{granted} \wedge \sigma(F_j;\mathfrak{E}) \models_x \Diamond\text{null}$

F.3.2 Proofs of Basic Progress Properties of Components

Classification of Behaviours:
Input- & Output-State Independent: CB1
Input-State Independent: None
Output-State Independent: CB4
State-Dependent: CB2, CB3, CB5, CB6

We will examine the satisfiabilty of faithfulness for the component behaviours. If and when we find behaviours that do not satisfy faithfulness, we will examine the behaviours in detail with respect to the first four basic progress properties.

Input- & Output-State Independent Behaviours:
sat(CB1, FT)

FT $\forall x : \mathbb{N} \cdot \forall \alpha : \Phi\{P_i\} \cdot \exists \beta : \Phi\{P_i\} \cdot$

$((\alpha;\beta) \wedge \sigma\{P_i\}(\text{in}) \models_x \alpha) \Rightarrow \sigma\{P_i\}(\text{out}) \models_x \Diamond\beta$

Let $(\alpha;\beta)$ be the input-output pair (eat ; getF_i)

$= \forall x : \mathbb{N} \cdot ((\text{eat} ; \text{getF}_i) \wedge \sigma\{P_i\}(\text{in}) \models_x \text{eat}) \Rightarrow \sigma\{P_i\}(\text{out}) \models_x \Diamond\text{getF}_i$

$= \forall x : \mathbb{N} \cdot ((\mathsf{T}) \wedge \sigma\{P_i\}(\text{in}) \models_x \text{eat}) \Rightarrow \sigma\{P_i\}(\text{out}) \models_x \Diamond\text{getF}_i$

$=$ **CB1**

Output-State Independent Behaviours:
sat(CB4, FT)

FT $= \forall x : \mathbb{N} \cdot \forall \alpha : \Phi\{P_i\} \cdot \exists \beta : \Phi\{P_i\} \cdot$

$((\alpha;\beta) \wedge \sigma\{P_i\}(\text{in}) \models_x \alpha) \Rightarrow \sigma\{P_i\}(\text{out}) \models_x \Diamond\beta$

Let $(\alpha;\beta)$ be the input-output pair $(\text{eating} ; \text{releaseF}_i \wedge \text{releaseF}_{i\oplus 1} \wedge \text{think})$

$= \forall x : \mathbb{N} \cdot \ ((\text{eating} ; \text{releaseF}_i \wedge \text{releaseF}_{i\oplus 1} \wedge \text{think}) \wedge \sigma\{P_i\}(\text{in}) \models_x \text{eating})$

$\Rightarrow \sigma\{P_i\}(\text{out}) \models_x \Diamond(\text{releaseF}_i \wedge \text{releaseF}_{i\oplus 1} \wedge \text{think})$

$= \forall x : \mathbb{N} \cdot$

$((\top) \wedge \sigma\{P_i\}(\text{in}) \models_x \text{eating} \wedge \sigma\{P_i\} \models_x (\text{gotF}_i \wedge \text{gotF}_{i\oplus 1}) \vee \neg(\text{gotF}_i \wedge \text{gotF}_{i\oplus 1}))$

$\Rightarrow \sigma\{P_i\}(\text{out}) \models_x \Diamond(\text{releaseF}_i \wedge \text{releaseF}_{i\oplus 1} \wedge \text{think})$

$= \forall x : \mathbb{N} \cdot (\sigma\{P_i\}(\text{in}) \models_x \text{eating} \wedge \sigma\{P_i\} \models_x (\text{gotF}_i \wedge \text{gotF}_{i\oplus 1}))$

$\Rightarrow [\sigma\{P_i\}(\text{out}) \models_x \Diamond(\text{releaseF}_i \wedge \text{releaseF}_{i\oplus 1} \wedge \text{think})$

$\wedge \sigma\{P_i\} \models_x \bigcirc((\text{gotF}_i \wedge \text{gotF}_{i\oplus 1}) \vee \neg(\text{gotF}_i \wedge \text{gotF}_{i\oplus 1}))]$

$\wedge (\sigma\{P_i\}(\text{in}) \models_x \text{eating} \wedge \sigma\{P_i\} \models_x \neg(\text{gotF}_i \wedge \text{gotF}_{i\oplus 1}))$

$\Rightarrow [\sigma\{P_i\}(\text{out}) \models_x \Diamond(\text{releaseF}_i \wedge \text{releaseF}_{i\oplus 1} \wedge \text{think})$

$\wedge \sigma\{P_i\} \models_x \bigcirc((\text{gotF}_i \wedge \text{gotF}_{i\oplus 1}) \vee \neg(\text{gotF}_i \wedge \text{gotF}_{i\oplus 1}))]$

$= \forall x : \mathbb{N} \cdot \{ [(\sigma\{P_i\}(\text{in}) \models_x \text{eating} \wedge \sigma\{P_i\} \models_x (\text{gotF}_i \wedge \text{gotF}_{i\oplus 1}))$

$\Rightarrow [\sigma\{P_i\}(\text{out}) \models_x \Diamond(\text{releaseF}_i \wedge \text{releaseF}_{i\oplus 1} \wedge \text{think})$

$\wedge \sigma\{P_i\} \models_x \bigcirc((\text{gotF}_i \wedge \text{gotF}_{i\oplus 1})]]$

$\vee [\sigma\{P_i\}(\text{in}) \models_x \text{eating} \wedge \sigma\{P_i\} \models_x (\text{gotF}_i \wedge \text{gotF}_{i\oplus 1}))$

$\Rightarrow [\sigma\{P_i\}(\text{out}) \models_x \Diamond(\text{releaseF}_i \wedge \text{releaseF}_{i\oplus 1} \wedge \text{think})$

$\wedge \sigma\{P_i\} \models_x \bigcirc\neg(\text{gotF}_i \wedge \text{gotF}_{i\oplus 1}))]] \}$

$\wedge \{ [(\sigma\{P_i\}(\text{in}) \models_x \text{eating} \wedge \sigma\{P_i\} \models_x \neg(\text{gotF}_i \wedge \text{gotF}_{i\oplus 1}))$

$\Rightarrow [\sigma\{P_i\}(\text{out}) \models_x \Diamond(\text{releaseF}_i \wedge \text{releaseF}_{i\oplus 1} \wedge \text{think})$

$\wedge \sigma\{P_i\} \models_x \bigcirc((\text{gotF}_i \wedge \text{gotF}_{i\oplus 1})]]$

$\vee [(\sigma\{P_i\}(\text{in}) \models_x \text{eating} \wedge \sigma\{P_i\} \models_x \neg(\text{gotF}_i \wedge \text{gotF}_{i\oplus 1}))$

$\Rightarrow [\sigma\{P_i\}(\text{out}) \models_x \Diamond(\text{releaseF}_i \wedge \text{releaseF}_{i\oplus 1} \wedge \text{think})$

$\wedge \sigma\{P_i\} \models_x \bigcirc(\neg(\text{gotF}_i \wedge \text{gotF}_{i\oplus 1}))]] \}$

$\Leftarrow \forall x : \mathbb{N} \cdot \sigma\{P_i\} \models_x (\text{gotF}_i \wedge \text{gotF}_{i\oplus 1}) \wedge \sigma(P_i;P_i) \models_x \text{eating}$

$\Rightarrow \sigma\{P_i\} \models_x \bigcirc(\text{gotF}_i \wedge \ \text{gotF}_{i\oplus 1})$

$\wedge \sigma(P_i;\text{E}) \models_x \Diamond(\text{releaseF}_i \wedge \text{releaseF}_{i\oplus 1} \wedge \text{think})$

$=$ **CB4**

sat(CB2,FT)

Let $(\alpha;\beta)$ be the input-output pair $(\text{gotF}_i ; \text{getF}_{i\oplus 1})$

$= \forall x : \mathbb{N} \cdot ((\text{gotF}_i ; \text{getF}_{i\oplus 1}) \wedge \sigma\{P_i\}(\text{in}) \models_x \text{gotF}_i) \Rightarrow \sigma\{P_i\}(\text{out}) \models_x \Diamond\text{getF}_{i\oplus 1}$

$= \forall x : \mathbb{N} \cdot \{ [(\sigma\{P_i\}(\text{in}) \models_x \text{gotF}_i \wedge \sigma\{P_i\} \models_x (\neg\text{gotF}_i \wedge \neg\text{gotF}_{i\oplus 1}))$

$\Rightarrow (\sigma\{P_i\}(\text{out}) \models_x \Diamond\text{getF}_{i\oplus 1} \wedge \sigma\{P_i\} \models_x \bigcirc(\text{gotF}_i \wedge \neg\text{gotF}_{i\oplus 1}))]$

$\vee [(\sigma\{P_i\}(\text{in}) \models_x \text{gotF}_i \wedge \sigma\{P_i\} \models_x (\neg\text{gotF}_i \wedge \neg\text{gotF}_{i\oplus 1}))$

$\Rightarrow (\sigma\{P_i\}(\text{out}) \models_x \Diamond\text{getF}_{i\oplus 1} \wedge \sigma\{P_i\} \models_x \bigcirc\neg(\neg\text{gotF}_i \wedge \neg\text{gotF}_{i\oplus 1}))]$

$\wedge \{ [(\sigma\{P_i\}(\text{in}) \models_x \text{gotF}_i \wedge \sigma\{P_i\} \models_x \neg(\neg\text{gotF}_i \wedge \neg\text{gotF}_{i\oplus 1}))$

$$\Rightarrow (\sigma\{P_i\}(\text{out}) \models_x \Diamond \text{getF}_{i\oplus 1} \wedge \sigma\{P_i\} \models_x \bigcirc(\neg \text{gotF}_i \wedge \neg \text{gotF}_{i\oplus 1}))]$$
$$\vee [(\sigma\{P_i\}(\text{in}) \models_x \text{gotF}_i \wedge \sigma\{P_i\} \models_x \neg(\neg \text{gotF}_i \wedge \neg \text{gotF}_{i\oplus 1}))$$
$$\Rightarrow (\sigma\{P_i\}(\text{out}) \models_x \Diamond \text{getF}_{i\oplus 1} \wedge \sigma\{P_i\} \models_x \bigcirc\neg(\neg \text{gotF}_i \wedge \neg \text{gotF}_{i\oplus 1}))] \}$$
$$\Leftarrow \textbf{CB2}$$

We can likewise prove that **CB3** $\Rightarrow$ **FT**, **CB5** $\Rightarrow$ **FT**, and **CB6** $\Rightarrow$ **FT**. Thus if the component behaviours are accurately described by the component behaviour axioms, then the component's behaviours will satify progress. Because these components are not intended to act as stand-alone components, we will not assess their progress for all possible inputs. Instead, we note those behaviours that are input-state dependent, and the inputs that will not be acted upon in each input-state, for consideration within the context of the composite system.

Behaviour Axiom	Input State	Affected Inputs
CB4	$\sigma\{P_i\} \models_x (\text{gotF}_i \wedge \text{gotF}_{i\oplus 1})$	gotF_i $\text{gotF}_{i\oplus 1}$
CB2	$\sigma\{P_i\} \models_x (\neg \text{gotF}_i \wedge \neg \text{gotF}_{i\oplus 1})$	$\text{gotF}_{i\oplus 1}$ eating
CB3	$\sigma\{P_i\} \models_x (\text{gotF}_i \wedge \neg \text{gotF}_{i\oplus 1})$	gotF_i
CB5	$\sigma\{F_i\} \models_x \neg$granted	releaseF_i
CB6	$\sigma\{F_i\} \models_x$ granted	getF_i

Note that the input *eat* is not included as an affected input. This is because *eat* is defined by **CB1**, an input-state independent behaviour axiom. Thus the component will always be available to an *eat* input.

F.3.3 Composite System Specification

The composite system is pictured in Figure F.2.

Interconnection rules:

($\text{outputP}_i(0)$; $\text{inputF}_i(0)$)
($\text{outputP}_i(1)$; $\text{inputF}_{i\oplus 1}(1)$)
($\text{outputP}_i(2)$; environs)
($\text{outputF}_i(0)$; $\text{inputP}_i(0)$)
($\text{outputF}_{i\oplus 1}(1)$; $\text{inputP}_i(1)$)
(environs ; $\text{inputP}_i(2)$)

Composite System Properties:

The following property is requirement on the behaviour of the composite system:

$$\forall P_i : \{\text{Philosophers}\} \cdot \circ (\sigma(\mathbb{E};P_i) \models_x \text{eat} \Rightarrow \sigma\{P_i;\mathbb{E}\} \models_x \Diamond \text{think})$$

These axioms define the state of the components in the system.

- A fork component that is not granted is not held by either possible philosopher component.

$$\forall P_i : \{\text{Philosophers}\} \cdot$$
$$\circ ((\sigma\{P_i\} \models_x \neg \text{gotF}_i \wedge \sigma\{P_{i\oplus 1}) \models_x \neg \text{gotF}_i) \Leftarrow \sigma\{F_i\} \models_x \neg \text{granted})$$

- If either philosopher component has possession of a fork component then that fork component is granted.

$$\forall P_i : \{\text{Philosophers}\} \cdot \circ ((\sigma\{P_i\} \models_x \text{gotF}_j \vee \sigma\{P_{i\ominus 1}) \models_x \text{gotF}_j) \Rightarrow \sigma\{F_j\} \models_x \text{granted})$$

- If a philosopher component has possession of its right-hand fork then it will also have possession of its left-hand fork.

$$\forall P_i : \{\text{Philosophers}\} \cdot \circ (\sigma\{P_i\} \models_x \text{got}F_{i\oplus 1} \Rightarrow (\sigma\{P_i\} \models_x \text{got}F_i \wedge \text{got}F_{i\oplus 1}))$$

System Behaviour Axioms:

- A philosopher not possessing any forks and told (by an outside source) to try to eat will attempt to obtain the fork to his left. He is still in the state of having no forks.

SB1 $\forall P_i : \{\text{Philosophers}\} \cdot \forall F_i : \{\text{Forks}\} \cdot \forall x : \mathbb{N} \cdot$

$$\sigma\{P_i\} \models_x (\neg \text{got}F_i \wedge \neg \text{got}F_{i\oplus 1}) \wedge \sigma(\mathbb{E};P_i) \models_x \text{eat} \Rightarrow$$

$$\sigma\{P_i\} \models_x \bigcirc(\neg \text{got}F_i \wedge \neg \text{got}F_{i\oplus 1}) \wedge \sigma(P_i;F_i) \models_x \Diamond \text{get}F_i$$

- Once the philospher has received his left fork, he is in the state of having that fork and will attempt to get his right-hand fork.

SB2 $\forall P_i : \{\text{Philosophers}\} \cdot \forall F_i : \{\text{Forks}\} \cdot \forall x : \mathbb{N} \cdot$

$$\sigma\{P_i\} \models_x (\neg \text{got}F_i \wedge \neg \text{got}F_{i\oplus 1}) \wedge \sigma(F_i;P_i) \models_x \text{got}F_i \Rightarrow$$

$$\sigma\{P_i\} \models_x \bigcirc(\text{got}F_i \wedge \neg \text{got}F_{i\oplus 1}) \wedge \sigma(P_i;F_{i\oplus 1}) \models_x \Diamond \text{get}F_{i\oplus 1}$$

- Once the philospher has his right fork, he is in the state of having both forks. He will begin eating.

SB3 $\forall P_i : \{\text{Philosophers}\} \cdot \forall F_i : \{\text{Forks}\} \cdot \forall x : \mathbb{N} \cdot$

$$\sigma\{P_i\} \models_x (\text{got}F_i \wedge \neg \text{got}F_{i\oplus 1}) \wedge \sigma(F_{i\oplus 1};P_i) \models_x \text{got}F_{i\oplus 1}$$

$$\Rightarrow \sigma\{P_i\} \models_x \bigcirc(\text{got}F_i \wedge \text{got}F_{i\oplus 1}) \wedge \sigma(P_i;P_i) \models_x \Diamond \text{eating}$$

- A philosopher that is eating will eventually release both forks and return to thinking.

SB4 $\forall P_i : \{\text{Philosophers}\} \cdot \forall F_i : \{\text{Forks}\} \cdot \forall x : \mathbb{N} \cdot$

$$\sigma\{P_i\} \models_x (\text{got}F_i \wedge \text{got}F_{i\oplus 1}) \wedge \sigma(P_i;P_i) \models_x \text{eating}$$

$$\Rightarrow \sigma\{P_i\} \models_x \bigcirc(\text{got}F_i \wedge \text{got}F_{i\oplus 1})$$

$$\wedge \sigma(P_i;F_i{}^{+}F_{i\oplus 1}{}^{+}\mathbb{E}) \models_x \Diamond(\text{release}F_i \wedge \text{release}F_{i\oplus 1} \wedge \text{think})$$

- A fork that is not granted, and receives a request from a philosopher, will be granted and the fork will eventually acknowledge this granting to the philosopher.

SB5 $\forall P_i : \{\text{Philosophers}\} \cdot \forall F_j : \{\text{Forks}\} \cdot \forall x : \mathbb{N} \cdot$

$$(j=i \vee j=i\oplus 1) \wedge \sigma\{F_j\} \models_x \neg \text{granted} \wedge \sigma(P_i;F_j) \models_x \text{get}F_j$$

$$\Rightarrow \sigma\{F_j\} \models_x \bigcirc\text{granted} \wedge \sigma(F_j;P_i) \models_x \Diamond \text{got}F_j$$

- A fork that is granted and possessed by a component will not be granted and will not be possessed by that component after that component sends a release fork request.

SB6 $\forall P_i : \{\text{Philosophers}\} \cdot \forall F_j : \{\text{Forks}\} \cdot \forall x : \mathbb{N} \cdot$

$$(j=i \vee j=i\oplus 1) \wedge \sigma\{F_j\} \models_x \text{granted} \wedge \sigma\{P_i\} \models_x \text{got}F_j \wedge \sigma(P_i;F_j) \models_x \text{release}F_j$$

$$\Rightarrow \sigma\{F_j\} \models_x \bigcirc\neg\text{granted} \wedge \sigma\{P_i\} \models_x \bigcirc\neg \text{got}F_j \wedge \sigma(F_j;\mathbb{E}) \models_x \Diamond \text{null}$$

Repeating the table given above, we add the column "Required Input", where the required input is that will return the component to the input-state required by the affected input.

Behaviour Axiom	Input State	Affected Inputs	Required Input
CB4	$\sigma\{P_i\} \models_x (\text{got}F_i \wedge \text{got}F_{i\oplus 1})$	$\text{got}F_i$	eating
	$\sigma\{P_i\} \models_x (\text{got}F_i \wedge \text{got}F_{i\oplus 1})$	$\text{got}F_{i\oplus 1}$	eating
CB2	$\sigma\{P_i\} \models_x (\neg\text{got}F_i \wedge \neg\text{got}F_{i\oplus 1})$	$\text{got}F_{i\oplus 1}$	Not Poss
	$\sigma\{P_i\} \models_x (\neg\text{got}F_i \wedge \neg\text{got}F_{i\oplus 1})$	eating	Not Poss
CB3	$\sigma\{P_i\} \models_x (\text{got}F_i \wedge \neg\text{got}F_{i\oplus 1})$	eating	Not Poss
CB3	$\sigma\{P_i\} \models_x (\text{got}F_i \wedge \neg\text{got}F_{i\oplus 1})$	$\text{got}F_i$	Not Poss
CB5	$\sigma\{F_i\} \models_x \neg\text{granted}$	$\text{release}F_i$	Not Poss
CB6	$\sigma\{F_i\} \models_x \text{granted}$	$\text{get}F_i$	$\text{release}F_i$

CB4 requires the input *eating*, which follows from the definition of CB4. The affected inputs of axioms CB2, CB3, and CB5 are not possible for the given input-state within the context of the composite system. The affected input of CB6, $getF_i$ received by a fork component while the fork is *granted*, however, does require that the input $releaseF_i$ be received by the fork component. If this input is not received, then the fork will not be able to process the input $getF_i$. Thus the fork component may not satisfy progress within the composite system if it does not receive the required input $releaseF_i$.

The next question to be addressed, therefore, is whether it is possible that this required input will never be received. If the component F_i is contained within a circular path or behaviour, then it is possible that the lack of the required input will cause the path and system to exhibit deadlock. There are two distinct circular paths within the composite system: P_i, F_i, P_i and $P_i, F_{i\oplus 1}, P_i$. Each must be examined in detail to determine if it will deadlock.

Any of the standard deadlock-detection algorithms, applied to those paths that include the components F_i, can be used to assess the possibility of deadlock. We did not start with these algorithms because it is instructive to examine these behaviours and pin-point potential problems before examining the system for deadlock.

To begin with, we list the source and destination of events and the associated state-change between input and output for each behaviour axiom. We use the Chandy-Misra-Haas algorithm [CMH83], where the path of a probe through the system is the same as the path of a given event through a system.

(SB1) $\mathbf{E}\ ; P_i \Rightarrow P_i\ ; F_i$

No state change

(SB5) $(P_i\ ; F_i \Rightarrow F_i\ ; P_i) \vee (P_{i\ominus 1}; F_i \Rightarrow F_i\ ; P_{i\ominus 1})$

$F_i \models_x \neg\text{granted} \Rightarrow F_i \models_x \bigcirc\text{granted}$

(SB2) $F_i\ ; P_i \Rightarrow P_i\ ; F_{i\oplus 1}$

$P_i \models_x \neg\text{got}F_i \Rightarrow P_i \models_x \bigcirc\text{got}F_i$

(SB5) $(P_i\ ; F_{i\oplus 1} \Rightarrow F_{i\oplus 1}\ ; P_i) \wedge (P_{i\oplus 1}\ ; F_{i\oplus 1} \Rightarrow F_{i\oplus 1}\ ; P_{i\oplus 1})$

$F_{i\oplus 1} \models_x \neg\text{granted} \Rightarrow F_{i\oplus 1} \models_x \bigcirc\text{granted}$

(SB3) $F_{i\oplus 1}\ ; P_i \Rightarrow P_i\ ; P_i$

$P_i \models_x \neg\text{got}F_{i\oplus 1} \Rightarrow P_i \models_x \bigcirc\text{got}F_{i\oplus 1}$

(SB4) $P_i ; P_i \Rightarrow P_i ; F_{i\oplus1}{}^{+} F_i {}^{+}$

$P_i \models_x (\text{got}F_i \wedge \text{got}F_{i\oplus1}) \Rightarrow P_i \models_x \Diamond(\neg\text{got}F_i \wedge \neg\text{got}F_{i\oplus1})$

As described by the system behaviour axioms, the output sent from P_i is received as an input at F_i which may also receive inputs from $P_{i\ominus1}$. Because both inputs will have the same effect on the state of F_i, and because P_i cannot continue to execute until it receives a response from F_i, we must consider the effects of these input events at F_i. We will prove that if P_i sends an event to component F_i, which in turn has previously received a similar request from $P_{i\ominus1}$, that P_i will eventually gain possession of component F_i. Therefore we must prove the following statement:

$\exists i . \exists u . \exists z>u . \sigma(P_i;F_i)\models_u \text{get}F_i \wedge \sigma\{F_i\} \models_u \text{granted} \wedge \sigma(F_i;P_{i\ominus1})\models_u \text{got}F_i$

$\Rightarrow \sigma\{P_i\}\models_z (\text{got}F_i \wedge \neg\text{got}F_{i\oplus1})$

Proof $\exists i . \exists u . \sigma(P_i;F_i)\models_u \text{get}F_i \wedge \sigma\{F_i\}\models_u \text{granted} \wedge \sigma(F_i;P_{i\ominus1})\models_u \text{got}F_i$

State $\Rightarrow \exists i . \exists u . \sigma(P_i;F_i)\models_u \text{get}F_i \wedge \sigma\{F_i\}\models_u \text{granted}$

$\wedge \sigma\{P_i\}\models_z (\neg\text{got}F_i \wedge \neg\text{got}F_{i\oplus1}) \wedge \sigma(F_i;P_{i\ominus1})\models_u \text{got}F_i$

SB3 $\Rightarrow \exists i . \exists v>u . \sigma(P_i;F_i)\models_v \text{get}F_i \wedge \sigma\{F_i\}\models_v \text{granted}$

$\wedge \sigma\{P_i\}\models_v (\neg\text{got}F_i \wedge \neg\text{got}F_{i\oplus1}) \wedge \sigma\{P_{i\ominus1}\}\models_v O(\text{got}F_{i\ominus1} \wedge \text{got}F_i)$

$\wedge \sigma(P_{i\ominus1};P_{i\ominus1}) \models_v \Diamond\text{eating}$

SB4 $\Rightarrow \exists i . \exists w>v . \sigma(P_i;F_i)\models_w \text{get}F_i \wedge \sigma\{F_i\}\models_w \text{granted}$

$\wedge \sigma\{P_{i\ominus1}\}\models_w O(\text{got}F_{i\ominus1} \wedge \text{got}F_i)$

$\wedge \sigma(P_{i\ominus1};F_i{}^{+}F_{i\ominus1}{}^{+}) \models_w \Diamond(\text{release}F_{i\ominus1} \wedge \text{release}F_i \wedge \text{think})$

SB6 $\Rightarrow \exists i . \exists x>w . \sigma(P_i;F_i)\models_x \text{get}F_i \wedge \sigma\{F_i\}\models_x O\neg\text{granted}$

$\wedge \sigma\{P_{i\ominus1}\}\models_x O(\neg\text{got}F_{i\ominus1} \wedge \neg\text{got}F_i) \wedge \sigma(F_i;) \models_x \Diamond\text{null}$

SB5 $\Rightarrow \exists i . \exists y>x . \sigma(F_i;P_i)\models_y \Diamond\text{got}F_i \wedge \sigma\{F_i\}\models_y O\text{granted}$

SB2 $\Rightarrow \exists i . \exists z>y . \sigma\{P_i\}\models_z (\text{got}F_i \wedge \neg\text{got}F_{i\oplus1})$ ❑

The philosopher will get the fork, and will be able to progress. If the event received from $P_{i\ominus1}$ had been a release, the event from P_i would have been acted upon and there would not have been a possible impediment to progress.

Given the second instantiation of SB5, what will happen if, according to SB2, P_i sends an event to component $F_{i\oplus1}$, which in turn has previously received a similar request from $P_{i\oplus1}$? In this case, we are required to prove the following:

$\exists i . \exists w . \exists z>w . \sigma(P_i;F_{i\oplus1})\models_w \text{get}F_{i\oplus1} \wedge \sigma\{F_{i\oplus1}\}\models_w \text{granted}$

$\wedge \sigma(F_{i\oplus1};P_{i\oplus1})\models_w \text{got}F_{i\oplus1} \wedge \sigma\{P_i\}\models_z (\text{got}F_i \wedge \neg\text{got}F_{i\oplus1})$

$\Rightarrow \sigma\{P_i\}\models_z (\text{got}F_i \wedge \text{got}F_{i\oplus1})$

Proof $\exists i . \exists w . \sigma(P_i;F_{i\oplus1})\models_w \text{get}F_{i\oplus1} \wedge \sigma\{F_{i\oplus1}\}\models_w \text{granted}$

$\wedge \sigma(F_{i\oplus1};P_{i\oplus1})\models_w \text{got}F_{i\oplus1}$

State $\Rightarrow \exists i . \exists w . \sigma(P_i;F_{i\oplus1}) \models_w \text{get}F_{i\oplus1} \wedge \sigma\{F_{i\oplus1}\}\models_w \text{granted}$

$\wedge \sigma\{P_i\}\models_w (\text{got}F_i \wedge \neg\text{got}F_{i\oplus1}) \wedge \sigma^{F_{i\oplus1};P_{i\oplus1}}\models_w \text{got}F_{i\oplus1}$

$$\begin{aligned}
\text{SB2} \quad & \Rightarrow \exists i\,.\,\exists x{>}w\,.\,\sigma(P_i;F_{i\oplus 1})\models_x \text{getF}_{i\oplus 1} \wedge \sigma\{F_{i\oplus 1}\}\models_x \text{granted} \\
& \wedge \sigma\{P_i\}\models_z (\text{gotF}_i \wedge \neg\text{gotF}_{i\oplus 1}) \\
& \wedge \sigma\{P_{i\oplus 1}\}\models_x (\text{gotF}_{i\oplus 1} \wedge \neg\text{gotF}_{i\oplus 2}) \wedge \sigma(P_{i\oplus 1};F_{i\oplus 2})\models_x \Diamond\text{getF}_{i\oplus 2} \\
& \Rightarrow \exists i\,.\,\exists y{>}x\,.\,\sigma(P_i;F_{i\oplus 1})\models_x \text{getF}_{i\oplus 1} \wedge \sigma\{F_{i\oplus 1}\}\models_x \text{granted} \\
& \wedge \sigma\{P_i\}\models_z (\text{gotF}_i \wedge \neg\text{gotF}_{i\oplus 1}) \\
& \wedge \sigma\{P_{i\oplus 1}\}\models_x (\text{gotF}_{i\oplus 1} \wedge \neg\text{gotF}_{i\oplus 2}) \wedge \sigma(P_{i\oplus 1};F_{i\oplus 2})\models_x \Diamond\text{getF}_{i\oplus 2} \\
& \wedge \sigma\{F_{i\oplus 2}\}\models_y (\text{granted} \vee \neg\text{granted})
\end{aligned}$$

But we do not know the state of component $F_{i\oplus 2}$. If the component is not granted, then we can simply follow through with the proof that the component will eventually be able to progress. If the component is granted, we must continue with the investigation.

$$\begin{aligned}
\text{SB2} \quad & \Rightarrow \exists i\,.\,\exists y\,.\,\sigma(P_i;F_{i\oplus 1})\models_y \text{getF}_{i\oplus 1} \wedge \sigma\{F_i\}\models_y \text{granted} \\
& \wedge \sigma\{P_i\}\models_y (\text{gotF}_i \wedge \neg\text{gotF}_{i\oplus 1}) \wedge \sigma(P_{i\oplus 1};F_{i\oplus 2})\models_y \text{getF}_{i\oplus 2} \wedge \sigma\{F_{i\oplus 1}\}\models_y \text{granted} \\
& \wedge \sigma\{P_{i\oplus 1}\}\models_y (\text{gotF}_{i\oplus 1} \wedge \neg\text{gotF}_{i\oplus 2}) \wedge \sigma(P_{i\oplus 2};F_{i\oplus 3})\models_y \text{getF}_{i\oplus 3} \\
& \wedge \sigma\{F_{i\oplus 2}\}\models_y \text{granted} \wedge \sigma\{P_{i\oplus 2}\}\models_y (\text{gotF}_{i\oplus 2} \wedge \neg\text{gotF}_{i\oplus 3})
\end{aligned}$$

We can see (and easily prove) that if we continue to experience the worst case scenario of the required component always being granted, we will end up with the following specification:

$$\begin{aligned}
\text{SB2} \quad & \Rightarrow \exists i\,.\,\exists y\,.\,\sigma(P_i;F_{i\oplus 1})\models_y \text{getF}_{i\oplus 1} \wedge \sigma\{F_i\}\models_y \text{granted} \\
& \wedge \sigma\{P_i\}\models_y (\text{gotF}_i \wedge \neg\text{gotF}_{i\oplus 1}) \wedge \sigma(P_{i\oplus 1};F_{i\oplus 2})\models_y \text{getF}_{i\oplus 2} \wedge \sigma\{F_{i\oplus 1}\}\models_y \text{granted} \\
& \wedge \sigma\{P_{i\oplus 1}\}\models_y (\text{gotF}_{i\oplus 1} \wedge \neg\text{gotF}_{i\oplus 2}) \wedge \sigma(P_{i\oplus 2};F_{i\oplus 3})\models_y \text{getF}_{i\oplus 3} \\
& \wedge \sigma\{F_{i\oplus 2}\}\models_y \text{granted} \wedge \sigma\{P_{i\oplus 2}\}\models_y (\text{gotF}_{i\oplus 2} \wedge \neg\text{gotF}_{i\oplus 3}) \\
& \wedge \ldots \\
& \qquad \wedge \sigma(P_{i\ominus 1};F_i)\models_z \text{getF}_i \wedge \sigma\{F_{i\ominus 1}\}\models_z \text{granted} \wedge \sigma\{P_{i\ominus 1}\}\models_z (\text{gotF}_{i\ominus 1} \wedge \neg\text{gotF}_i) \qquad \square
\end{aligned}$$

In this case, every P-component has sent a request to an F-component, which is not able to be granted because the F-component is in turn granted to a different P-component.

This system cannot progress in this situation; each P-component is waiting for a specific input which it will not receive and therefore the P-component cannot progress. Because the P-components cannot progress, they will not be able to produce as an output the input required by the F-components that will allow the F-components to progress, which would in turn allow the P-components to progress.

This situation can be described as follows:

$$\forall i.\exists x.\ \bigwedge_1 \{\sigma\{F_i\}\models_x \text{granted} \wedge \sigma(P_i;F_{i\oplus 1})\models_x \text{getF}_{i\oplus 1} \wedge \sigma\{P_i\}\models_x (\text{gotF}_i \wedge \neg\text{gotF}_{i\oplus 1})\}$$

$$= \forall i.\exists x.\ \bigwedge_1 \{\sigma\{P_{i\oplus 1}\}\models_x \circ\, \text{gotF}_{i\oplus 1} \wedge \sigma(P_i;F_{i\oplus 1})\models_x \text{getF}_{i\oplus 1}\}$$

Therefore, there are behaviours such that the system property

$$\forall i\,.\,\sigma^{\mathbf{E};P_i}\models_x \text{eat} \Rightarrow \sigma^{P_i;\mathbf{E}}\models_x \Diamond\text{think}$$

will never be true. Using the notation given above, livelock and deadlock occur when:

Livelock: $\exists i.\exists x.\ \bigwedge_1 \{\sigma\{P_{i\oplus 1}\}\models_x \circ\, \text{gotF}_{i\oplus 1} \wedge \sigma(P_i;F_{i\oplus 1})\models_x \text{getF}_{i\oplus 1}\}$

Deadlock: $\forall i.\exists x.\ \bigwedge_1 \{\sigma\{P_{i\oplus 1}\}\models_x \circ\, \text{gotF}_{i\oplus 1} \wedge \sigma(P_i;F_{i\oplus 1})\models_x \text{getF}_{i\oplus 1}\}$

There have been many solutions proposed for this particular problem, including round-robin priorities on access to the forks, requiring a philosopher to put down their left fork if they cannot pick up their right fork, and so on. We will chose to avoid the problem altogether by modifying the behaviour of the system. This is in keeping with the design of a system: if it doesn't work, try a minor adjustment to see if that helps.

F.3.4 Solution to Dining Philosophers Livelock

The system as defined requires that each philosopher begin by attempting to pick up their left fork. The solution that we propose is to require all philosophers sitting in odd-numbered chairs to attempt to pick up their right-hand fork first, while the remaining even-numbered philosophers attempt to pick up their left-hand fork first. We will prove this solution for an odd number of philosophers in the system. It can be similarly proven to be true for an even number of philosophers.

Input-Output Events

Each philosopher is labelled as even or odd, depending on whether they are sitting in an even-numbered or odd-numbered chair at the table.

Philosophers:

i odd $P_i(\ eat\ ;\ getF_{i\oplus1}\ )$
$P_i(\ gotF_{i\oplus1}\ ;\ getF_i\ )$
$P_i(\ gotF_i\ ;\ eating\ )$
$P_i(\ eating\ ;\ (releaseF_i \wedge releaseF_{i\oplus1} \wedge think)\ )$

i even $P_i(\ eat\ ;\ getF_i\ )$
$P_i(\ gotF_i\ ;\ getF_{i\oplus1}\ )$
$P_i(\ gotF_{i\oplus1}\ ;\ eating\ )$
$P_i(\ eating\ ;\ (releaseF_i \wedge releaseF_{i\oplus1} \wedge think)\ )$

Forks: $F_i(\ getF_i\ ;\ gotF_i\ )$
$F_i(\ releaseF_i\ ;\ null\ )$

Environment: $\mathfrak{E}(\ think\ ;\ eat\ )$

State Axioms

- If an even-numbered philosopher component has possession of its right-hand fork then it will also have possession of its left-hand fork.

$$\forall i\ .\ i\ even\ .\ \forall x\ .\ \sigma\{P_i\} \models_x gotF_{i\oplus1} \Rightarrow (\ \sigma\{P_i\} \models_x gotF_i \wedge gotF_{i\oplus1}\ )$$

- If an odd-numbered philosopher component has possession of its left-hand fork then it will also have possession of its right-hand fork.

$$\forall i\ .\ i\ odd\ .\ \forall x\ .\ \sigma\{P_i\} \models_x gotF_i \Rightarrow (\ \sigma\{P_i\} \models_x gotF_i \wedge gotF_{i\oplus1}\ )$$

System Behaviour Axioms

- An even-numbered philosopher who is thinking will be told (by an outside source) to try to eat and will eventually try to get their left-hand fork.

SB1 $\forall i\ .\ i\ even\ .\ \exists x\ .\ \sigma\{P_i\} \models_x (\ \neg gotF_i \wedge \neg gotF_{i\oplus1}\ ) \wedge \sigma(\mathfrak{E};P_i) \models_x eat\ \Rightarrow$

$$\sigma\{P_i\} \models_x \bigcirc(\ \neg gotF_i\ \wedge \neg gotF_{i\oplus1}\ ) \wedge \sigma(P_i;F_i) \models_x \Diamond getF_i$$

- An even-numbered philosopher who has got possession of their left-hand fork will eventually try to get their right-hand fork.

SB2 $\forall i\ .\ i\ even\ .\ \exists x\ .\ \sigma\{P_i\} \models_x (\ \neg gotF_i \wedge \neg gotF_{i\oplus1}\ ) \wedge \sigma(F_i;P_i) \models_x gotF_i\ \Rightarrow$

$$\sigma\{P_i\} \models_x \bigcirc(\ gotF_i \wedge \neg gotF_{i\oplus1}\ ) \wedge \sigma(P_i;F_{i\oplus1}) \models_x \Diamond getF_{i\oplus1}$$

- An even-numbered philosopher who has got possession of their right-hand fork will have possession of both forks and will eventually start eating.

SB3 $\forall i\ .\ i\ even\ .\ \exists x\ .\ \sigma\{P_i\} \models_x (\ gotF_i \wedge \neg gotF_{i\oplus1}\ ) \wedge \sigma(F_{i\oplus1};P_i) \models_x gotF_{i\oplus1}$

$$\Rightarrow\ \sigma\{P_i\} \models_x \bigcirc(\ gotF_i \wedge gotF_{i\oplus1}\ ) \wedge \sigma(P_i;P_i) \models_x \Diamond eating$$

- An odd-numbered philosopher who is thinking will be told (by an outside source) to try to eat and will eventually try to get their right-hand fork.

SB4 $\forall i\ .\ i\ odd\ .\ \exists x\ .\ \sigma\{P_i\} \models_x (\ \neg gotF_i \wedge \neg gotF_{i\oplus1}\ ) \wedge \sigma^{\mathfrak{E};P_i} \models_x eat\ \Rightarrow$

$$\sigma\{P_i\} \models_x \bigcirc(\ \neg gotF_i\ \wedge \neg gotF_{i\oplus1}\ ) \wedge \sigma(P_i;F_{i\oplus1}) \models_x \Diamond getF_{i\oplus1}$$

- An odd-numbered philosopher who has got possession of their right-hand fork will eventually try to get their left-hand fork.

SB5 $\forall i \,.\, i \text{ odd} \,.\, \exists x \,.\, \sigma\{P_i\} \models_x (\neg gotF_i \wedge \neg gotF_{i\oplus 1}) \wedge \sigma(F_{i\oplus 1};P_i) \models_x gotF_{i\oplus 1} \Rightarrow$

$\sigma\{P_i\} \models_x O(\neg gotF_i \wedge gotF_{i\oplus 1}) \wedge \sigma(P_i;F_{i\oplus 1}) \models_x \Diamond getF_i$

- An odd-numbered philosopher who has got possession of their left-hand fork will have possession of both forks and will eventually start eating.

SB6 $\forall i \,.\, i \text{ odd} \,.\, \exists x \,.\, \sigma\{P_i\} \models_x (\neg gotF_i \wedge gotF_{i\oplus 1}) \wedge \sigma(F_i;P_i) \models_x gotF_i$

$\Rightarrow \sigma\{P_i\} \models_x O(gotF_i \wedge gotF_{i\oplus 1}) \wedge \sigma(P_i;P_i) \models_x \Diamond eating$

- A philosopher who has got possession of both forks and is eating will eventually release both forks .

SB7 $\forall i \,.\, \exists x \,.\, \sigma\{P_i\} \models_x (gotF_i \wedge gotF_{i\oplus 1}) \wedge \sigma(P_i;P_i) \models_x eating$

$\Rightarrow \sigma\{P_i\} \models_x O(gotF_i \wedge gotF_{i\oplus 1})$

$\wedge \sigma(P_i;F_i{}^{+}F_{i\oplus 1}{}^{+}\text{Æ}) \models_x \Diamond(releaseF_i \wedge releaseF_{i\oplus 1} \wedge think)$

- A fork that is not granted and is requested by a philosopher will be granted and will eventually be acknowledge as granted to the philosopher by the fork

SB8 $\forall i,j \,.\, j=i \vee j=i\oplus 1 \,.\, \exists x \,.\, \sigma\{F_j\} \models_x \neg granted \wedge \sigma(P_i;F_j) \models_x getF_j$

$\Rightarrow \sigma\{F_j\} \models_x O granted \wedge \sigma(F_j;P_i) \models_x \Diamond gotF_j$

- A fork that is granted and is possessed by a philosopher will no longer be granted and will not be possessed by that philosopher after that philosopher has produced a release fork event.

SB9 $\forall i,j \,.\, j=i \vee j=i\oplus 1 \,.\, \exists x \,.\, \sigma\{F_j\} \models_x granted \wedge \sigma\{P_i\} \models_x gotF_j \wedge \sigma(P_i;F_j) \models_x releaseF_j$

$\Rightarrow \sigma\{F_j\} \models_x O\neg granted \wedge \sigma\{P_i\} \models_x O\neg gotF_j \wedge \sigma(F_j;\text{Æ}) \models_x \Diamond null$

Interconnection Rules
These are unaffected by this change in the definition of the system.

From the guidelines for assessing progress applied to the original problem, we see that there are two possible instances in which each philosopher may succumb to livelock. An even-numbered philosopher may first experience livelock while attempting to obtain their left-hand fork, and then while attempting to obtain their right-hand fork. An odd-numbered philosopher may first experience livelock while attempting to obtain their right-hand fork and then while attempting to obtain their left-hand fork. Thus there are four cases which which we must examine to ensure that this modified system is free from livelock.

F.3.4.1 Proof of Freedom from Livelock

Case 1: Even-numbered philosopher attempting to access their left-hand fork, which is already granted. We prove that regardless of whether the left-handed fork is granted to another philosopher or not, the even-numbered philosopher will eventually have possession of the fork.

$\exists i \,.\, i \text{ even} \,.\, \exists x \,.\, \exists z{>}x \,.\, \sigma\{P_i\} \models_x (\neg gotF_i \wedge \neg gotF_{i\oplus 1})$

$\wedge \sigma(P_i;F_i) \models_x \Diamond getF_i \wedge \sigma\{F_i\} \models_x granted \wedge \sigma(P_{i\ominus 1}) \models_x gotF_i$

$\Rightarrow \sigma\{P_i\} \models_z (gotF_i \wedge \neg gotF_{i\oplus 1})$

Case 2: Even-numbered philosopher attempting to get possession of their right-handed fork, which is granted to another philosopher.

$\exists i \,.\, i \text{ even} \,.\, \exists w \,.\, \exists z{>}w \,.\, \sigma\{P_i\} \models_w (gotF_i \wedge \neg gotF_{i\oplus 1})$

$\wedge \sigma\{P_{i\oplus 1}\} \models_w gotF_{i\oplus 1} \wedge \sigma(P_i;F_{i\oplus 1}) \models_w \Diamond getF_{i\oplus 1}$

$\Rightarrow \sigma\{P_i\} \models_z (gotF_i \wedge \neg gotF_{i\oplus 1})$

Proof $\exists i \,.\, i \text{ even} \,.\, \exists w \,.\, \sigma\{P_i\} \models_w (\text{gotF}_i \wedge \neg \text{gotF}_{i\oplus 1}) \wedge \sigma\{P_{i\oplus 1}\} \models_w \text{gotF}_{i\oplus 1}$

$\wedge\, \sigma(P_i;F_{i\oplus 1}) \models_w \Diamond \text{getF}_{i\oplus 1}$

State, $\Rightarrow \exists i \,.\, i \text{ even} \,.\, \exists w \,.\, \sigma\{P_i\} \models_w (\text{gotF}_i \wedge \neg \text{gotF}_{i\oplus 1})$

SB6 $\wedge\, \sigma\{P_{i\oplus 1}\} \models_w (\text{gotF}_{i\oplus 1} \wedge \text{gotF}_{i\oplus 2}) \wedge \sigma^{P_{i\oplus 1};P_{i\oplus 1}} \models_w \Diamond \text{eating}$

$\wedge\, \sigma(P_i;F_{i\oplus 1}) \models_w \Diamond \text{getF}_{i\oplus 1} \wedge \sigma^{F_i} \models_w \text{granted} \wedge \sigma^{F_{i\oplus 1}} \models_w \text{granted}$

SB7, SB9, SB8, SB8

$\Rightarrow \exists i \,.\, i \text{ even} \,.\, \exists z{>}y \,.\, \sigma\{P_i\} \models_z (\text{gotF}_i \wedge \text{gotF}_{i\oplus 1})$ ❑

We can similarily prove that the system will not exhibit livelock for an odd-numbered philosopher attempting to access either his left or right-hand fork. Therefore, all components the system as modified are free from livelock and the system is free from livelock and deadlock.

We can therefore easily prove the system-specific progress property of

$$\forall i \,.\, \sigma^{\mathbb{E};P_i} \models_x \text{eat} \Rightarrow \sigma^{P_i;\mathbb{E}} \models_x \Diamond \text{think}$$

This redefined system will not exhibit livelock providing the individual components (the forks and the philosophers) obey the basic progress properties.

F.4 Network Composition

F.4.1 Composition of Network System

In this section we examine the assessment of the basic safety properties of the network system given in section 11.5. We begin by listing the properties of the components.

$\mathbf{P}(A) = \mathbf{P}(B) = \{$ COC, COI, AU-E, AU $\}$

$\mathbf{P}(S) = \{$ PC, PI, AU-E, AU $\}$

Using the Composition Principle, we begin with the hypothesis of rule 3 of the principle, that is, that the components satisfy their relevant environment properties (assumptions).

3. $\mathbf{sat}(A, \text{COC}) \wedge \mathbf{sat}(A, \text{COI}) \wedge \mathbf{sat}(A, \text{AU-E}) \Rightarrow \mathbf{sat}(A, \text{AU})$

$\mathbf{sat}(B, \text{COC}) \wedge \mathbf{sat}(B, \text{COI}) \wedge \mathbf{sat}(B, \text{AU-E}) \Rightarrow \mathbf{sat}(B, \text{AU})$

$\mathbf{sat}(S, \text{PC}) \wedge \mathbf{sat}(S, \text{PI}) \wedge \mathbf{sat}(S, \text{AU-E}) \Rightarrow \mathbf{sat}(S, \text{AU})$

This follows trivially given that we know that the components have already been evaluated and shown to satisfy all relevant properties. We now use rule 3 to prove rule 1 of the Composition Principle.

1. $\mathbf{P}(A) \cup \mathbf{P}(B) \cup \mathbf{P}(S) = \{$ PC, PI, COC, COI, AU-E, AU $\}$

$\mathbf{P}(A) \cap \mathbf{P}(B) \cap \mathbf{P}(S) = \{$ AU-E, AU $\}$

$\text{AU} \in \mathbf{P}(A) \cap \mathbf{P}(B) \cap \mathbf{P}(S) \cdot \neg \mathit{composable}(\text{AU}) \wedge \mathit{relevant}(\text{AU}) \wedge \text{AU} \in M_{reqd}$

$\text{AU-E} \in \mathbf{P}(A) \cap \mathbf{P}(B) \cap \mathbf{P}(S) \cdot \neg \mathit{composable}(\text{AU-E}) \wedge \mathit{relevant}(\text{AU-E}) \wedge \text{AU-E} \in M_{reqd}$

$\text{COC} \in \mathbf{P}(A) \setminus \mathbf{P}(A) \cap \mathbf{P}(B) \cap \mathbf{P}(S) \cdot \mathit{relevant}(\text{COC}) \wedge \text{COC} \in M_{reqd}$

$\text{COI} \in \mathbf{P}(A) \setminus \mathbf{P}(A) \cap \mathbf{P}(B) \cap \mathbf{P}(S) \cdot \mathit{relevant}(\text{COI}) \wedge \text{COI} \in M_{reqd}$

$\text{PC} \in \mathbf{P}(S) \setminus \mathbf{P}(A) \cap \mathbf{P}(B) \cap \mathbf{P}(S) \cdot \neg \mathit{relevant}(\text{PC})$

$\text{PI} \in \mathbf{P}(S) \setminus \mathbf{P}(A) \cap \mathbf{P}(B) \cap \mathbf{P}(S) \cdot \neg \mathit{relevant}(\text{PI})$

Evaluating the properties of M_{reqd} for the composite system, we find that

$\mathbf{sat}(\mathbb{C}, \text{AU}) \wedge \mathbf{sat}(\mathbb{C}, \text{AU-E}) \wedge \mathbf{sat}(\mathbb{C}, \text{COC}) \wedge \mathbf{sat}(\mathbb{C}, \text{COI})$

The relevant properties of COC and COI are not composable and therefore are not satisfied by the composite system. Examining the system using the algorithm of Figure 10.2, we see that the only way we will have a usable system that is able to handle all three security levels is to trust the switch component not to send Top Secret events to component A and Confidential events to component B. Assessing the domain of the system using Figure 10.1 and assuming the trustworthiness of the switch, we see that component optimistic confidentiality and integrity are satisfied.

Authenticity is not composable because of the effects of changing the domain of the individual systems and the possible effects that would have in rendering previously acceptable events unacceptable. Because the component optimistic properties are satisfied, however, we find that the domains do not change and that in this situation authenticity is in fact composable.

Even though the switch satisfies pessimistic confidentiality and integrity, we can show that it also satisfies component optimistic confidentiality and integrity simply because all events that may be sent to it by the two components are acceptable to the switch. This follows from the authenticity of components A and B, given that they may produce events that are {Confidential, Secret, Top Secret}, all of which are acceptable to the switch.

2. $\mathbf{P}(\mathbb{C}) = \{$ COC, COI, AU-E, AU $\}$

$$\mathbf{sat}(\mathbb{C}, \text{AU-E}) \wedge \mathbf{sat}(\mathbb{C}, \text{COC}) \wedge \mathbf{sat}(\mathbb{C}, \text{COI}) \wedge \mathbf{sat}(\text{B}, \text{AU}) \wedge \mathbf{sat}(\text{S}, \text{AU})$$
$$\Rightarrow \mathbf{sat}(\text{A}, \text{AU-E}) \wedge \mathbf{sat}(\text{A}, \text{COC}) \wedge \mathbf{sat}(\text{A}, \text{COI})$$
$$\mathbf{sat}(\mathbb{C}, \text{AU-E}) \wedge \mathbf{sat}(\mathbb{C}, \text{COC}) \wedge \mathbf{sat}(\mathbb{C}, \text{COI}) \wedge \mathbf{sat}(\text{A}, \text{AU}) \wedge \mathbf{sat}(\text{S}, \text{AU})$$
$$\Rightarrow \mathbf{sat}(\text{B}, \text{AU-E}) \wedge \mathbf{sat}(\text{B}, \text{COC}) \wedge \mathbf{sat}(\text{B}, \text{COI})$$
$$\mathbf{sat}(\mathbb{C}, \text{AU-E}) \wedge \mathbf{sat}(\mathbb{C}, \text{COC}) \wedge \mathbf{sat}(\mathbb{C}, \text{COI}) \wedge \mathbf{sat}(\text{A}, \text{AU}) \wedge \mathbf{sat}(\text{B}, \text{AU})$$
$$\Rightarrow \mathbf{sat}(\text{S}, \text{AU-E}) \wedge \mathbf{sat}(\text{S}, \text{PC}) \wedge \mathbf{sat}(\text{S}, \text{PI})$$

Thus the Composition Principle simplified the proof of the properties of the composite system given the knowledge of the properties of the components of the system. The composition principle did not allow us to make any assumptions about properties that were not composable. Instead, it highlighted where we had to go outside the principle to prove the satisfiability of certain properties.