Hoare Logic for Disjunctive Information Flow

Hanne Riis Nielson, Flemming Nielson(✉), and Ximeng Li

DTU Compute, Technical University of Denmark, Kongens Lyngby, Denmark
{hrni,fnie,ximl}@dtu.dk

Abstract. Information flow control extends access control by not only regulating who is allowed to access what data but also the subsequent use of the data accessed. Applications within communication networks require such information flow control to depend on the actual data. For a concurrent language with synchronous communication and separate data domains we develop a Hoare logic for enforcing disjunctive information flow policies. We establish the soundness of the Hoare logic with respect to an operational semantics and illustrate the development on a running example.

1 Introduction

Access control is a standard technique for guarding the confidentiality and integrity of data. It may take the flavour of a *discretionary* policy where for each file and user it is determined whether or not there is read-access (regarding confidentiality) or write-access (regarding integrity). Alternatively, it may take the flavour of a *mandatory* policy where files and users are characterised according to some security lattice and where flows are only permitted as allowed by the partial order. Examples include Bell and LaPadula [5] (for confidentiality) and Biba [6] (for integrity). Typically, access control is implemented dynamically by a reference monitor that halts execution when a policy is violated.

Information flow control goes one step further in attempting to ensure that subsequent use of the data adheres to the intended policy. It may take the flavour of a *mandatory* policy expressed using security lattices embodying confidentiality considerations (motivated by Bell and LaPadula) or integrity considerations (motivated by Biba). Typically, information flow control is implemented statically by a type system that ensures that policies cannot be violated, and the semantic guarantees are expressed using non-interference results [12,24]. Alternatively, it may take the flavour of a *discretionary* policy where data variables are marked with security labels indicating which users may read (for confidentiality) or write (for integrity) the data variables. A prominent example is the Decentralized Label Model (DLM) [17,18], which is also implemented statically by a type system enforcing the policies. Since the security labels can be seen as elements of a lattice one might employ non-interference ideas for the semantic characterisation.

Information Flow Control in Avionics: The increased use of wireless communication within avionics gives rise to new security challenges that cannot be

C. Bodei et al. (Eds.): Degano Festschrift, LNCS 9465, pp. 47–65, 2015.
DOI: 10.1007/978-3-319-25527-9_6

fully mitigated by current practices. In particular, there is a growing demand for techniques for controlling the information flow between different security and safety domains on-board and off-board the aircraft. Such techniques have to be integrated with the existing software architectures, in particular the MILS (Multiple Independent Levels of Security) architecture [21]. The MILS architecture is based on a strict separation of processes into partitions with isolated resources and a (certifiable) separation kernel controlling (and limiting) the interprocess communication (IPC) between the partitions [16]. This architecture provides a compositional approach for validating the security of the system – however, the constrained flow of information between the partitions being enforced by the separation kernel is now being challenged.

The ARINC-811 report [10] explicitly addresses the security issues in avionics and calls for separating the software in a number of security domains. This is illustrated by a "closed domain" for highly critical applications controlling the aircraft, a "private domain" for the less critical application operating the airline and for informing and entertaining the passengers and a "public domain" for the passenger owned devices. These domains will exchange information with one another and with external domains as for example ground control. The Bell and LaPadula approach and the Biba approach go some way towards controlling the information flow but a more fine-grained control is needed to handle the flexibility required in future avionics architectures.

An emerging challenge [16] is to let policies depend on the actual data. This is illustrated for an avionics gateway where the possible interactions between security domains depend not only on the security domains themselves but also on the content of messages exchanged between them. The essence of the scenario can be illustrated by a simple example: *A multiplexer that merges data from several sources, transport them over a joint channel, and then split them to reach different targets.* The different sources and targets belong to different security domains and hence they are likely to have different security policies; the merged data will include information about the intended source and destination. It then becomes challenging to express the policy for the merged data, as it is dependent on the data values specifying the intended source and destination.

Our Contribution: We extend discretionary information flow policies to deal with content-dependent security policies in a setting inspired by the MILS architecture and adhering to the separation of the software into security domains as advocated by the ARINC-811 report. We illustrate our approach on the multiplexer example mentioned above (and further elaborated in Sect. 2) and we prove the correctness of our approach with respect to a co-inductive correctness predicate defined by means of a formal semantics.

A language of *concurrent processes* each with their own memory and with synchronous communication as the only means for exchange of data is introduced in Sect. 3. It is equipped with a Structural Operational Semantics [20] that is instrumented to record the use of data in the form of a *flow relation*; for dealing with the implicit uses of data [12,24] we use the technique of "local environments" of [20]. The flow relation captures the duality between readers

and influencers that also will be present in the policies in the sense that *forward* flows are appropriate for the constraints on influencers, whereas *backward* flows are appropriate for the constraints on readers.

Our basic and disjunctive policies are introduced in Sect. 4. *Basic policies* are concerned with channels as well as variables and are based on policies for readers (confidentiality) and influencers (integrity) defined using ideas from DLM [17,18]. We extend on DLM by including also content-based policies characterising the permissible value ranges of data. The policies are equipped with a partial ordering capturing the duality between confidentiality and integrity – as known from other studies of access control and information flow control [14,15,17,18]. We define what it means for a flow relation (from the semantics) to satisfy a set of basic policies. *Disjunctive policies* are sets of basic policies and allow to shift between policies as required by the value-range information; they are essential for dealing with the motivating multiplexer example. We conclude by providing a *co-inductively defined* notion of *self-similarity* for expressing what it means for a system to satisfy a disjunctive security policy.

A *combined Hoare logic and type system* for verifying whether a system adheres to the specified disjunctive policies is developed in Sect. 5. While type systems have been used extensively for formulating information flow policies, the need to consider the actual data values leads us to combining it with a Hoare logic in order to determine the appropriateness of the basic security policies contained in the overall disjunctive policy. The preconditions of the Hoare formulae allow us to select the relevant basic security policies and to perform the relevant check on the readers and influencers on just these policies; analogously, the postconditions may restrict which security policies that are enforced for the continuation of the process. Another advantage of using a Hoare logic is that this allows to cleanly incorporate also the results of prior static analyses into the information flow type system; this is needed in order to interact with the approach of industrial users and is a need also discussed in [1]. Although we are studying a concurrent language, the underlying Hoare logic is fairly standard because we are modelling a MILS architecture and therefore the individual processes have no shared variables. The semantic correctness takes the form of proving that *typability is a self-simulation*.

Related Work: The approach of [7] shares some of our aims of discretionary information flow, but we deal with both confidentiality and integrity as well as value ranges, we provide a clear explanation of the opposite directions of flow that are appropriate for their formalisation, we deal with a concurrent language rather than a purely sequential (functional) language, and we admit disjunctive policies; although we do not consider the relationship to non-interference our self-simulation based approach points in that direction. Our use of "local environments" may be compared with the use of stacks in the monitoring rules of [22] for achieving mandatory information flow for confidentiality.

The use of locks [8] would appear to have some relationship to value ranges (whether or not a lock is taken) but the main purpose is that of modelling stateful policies. An interesting Hoare logic for dealing with mandatory information

flow for confidentiality is considered for a rich concurrent language with procedures in [2]; the Hoare logic permits dealing with data in the spirit of our value ranges, but there is no consideration of integrity nor of any semantic notion of correctness. The approach of [1] considers a Hoare logic directly relating pairs of states thereby being able to express non-interference properties in a natural manner. The development of [1] is proved sound, and while it deals with our value ranges in a rather advanced manner, there is no consideration of neither readers nor influencers nor of the difficulties of dealing with concurrency; indeed, one of the strong points of our work is that we are able to use security policies explicitly just as in the classical approaches and to do so in a concurrent setting. Focusing instead on strongest postconditions (in the form of a dynamic logic) the approach of [11] is able to directly formulate non-interference properties for a notion of mandatory information flow policies for confidentiality — but again without taking concurrency into account.

Many other papers deal with the Decentralized Label Model and information flow policies with aims that differ from ours. As an example, [26] aims at extending policies to give information about the availability of data (supplementing confidentiality and integrity of information), and [9] aims to connect the confidentiality and integrity dimensions by ensuring that data of low integrity cannot be used for deciding whether or not to declassify with respect to confidentiality. Considering a synchronous data flow language, [25] considers trace-based formulations of influencers and relates it to a non-interference property, and [13] considers the application of mandatory policies to avionics.

2 Motivating Example

Our development is motivated by the example illustrated in Fig. 1: two producers p_1 and p_2 send data to a multiplexer m over the channels in_1 and in_2, respectively. The multiplexer wraps the data up and forwards it over the channel ch to a demultiplexer d. The demultiplexer will then unwrap the data and forward it to the consumers c_1 and c_2 while adhering to the policy that data from p_1 is only allowed to reach c_1 and similarly data from p_2 is only allowed to reach c_2.

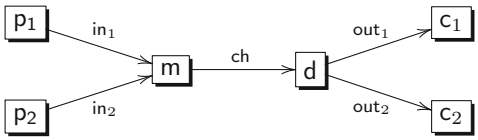

Fig. 1. The principals and channels of the multiplexer example.

We shall mainly be interested in the multiplexer and demultiplexer; we may write their code as follows:

$$\begin{array}{ll} \mathsf{m} : \mathsf{while\ true\ do} & \mathsf{d} : \mathsf{while\ true\ do} \\ \quad (\ \mathsf{in}_1?x_1; \mathsf{ch}!(1,x_1) & \quad (\ \mathsf{ch}?(y,z); \\ \quad \oplus\ \mathsf{in}_2?x_2; \mathsf{ch}!(2,x_2)\ ) & \quad\quad \mathsf{if}\ y = 1\ \mathsf{then}\ \mathsf{out}_1!z\ \mathsf{else}\ \mathsf{out}_2!z\ ) \end{array}$$

Here the channel ch is dyadic while the other channels are monadic. The multiplexer iterates through a loop, where it non-deterministically chooses to read from one of the channels in_1 or in_2 (as indicated by the operator $\oplus$), and then sends the data on the channel ch tagged with the constant 1 or 2 to record the source of the data. The demultiplexer also iterates through a loop; it will read a message from the channel ch and decides from the tag of the message whether the data itself has to be sent on the channel out_1 or out_2.

In the setting of the avionics gateway the principals may belong to the same security domain or they may belong to different security domains.

The Policies: We now associate policies with the channels; there will be a *confidentiality* part describing who is allowed to read the data sent on the channel, and there will be an *integrity* part describing who is allowed to have influenced the data sent on the channel.

Let us write P^i for the policy catering for data flowing from p_i to c_i (for $i = 1, 2$). The data will first be sent over the channel in_i and the integrity part of our policy will express that only p_i is allowed to influence the data sent on this channel. The confidentiality part of the policy is more complex: clearly m should be allowed to read the data, but we shall also allow d and c_i to read, since the data is to be passed from m to d and further on to c_i. We formalise this by specifying

$$\mathsf{P}^i_{\mathsf{i}}(\mathsf{in}_i) = \{\mathsf{p}_i\} \qquad \mathsf{P}^i_{\mathsf{r}}(\mathsf{in}_i) = \{\mathsf{m}, \mathsf{d}, \mathsf{c}_i\}$$

where we use the subscript i for the integrity part of the policy and the subscript r for the confidentiality part.

Let us next consider the policy for the channel out_i. Clearly d influences the data but since the data originates from p_i and has passed through m, we shall include all three as influencers. However, there is only one reader, namely c_i, so we specify

$$\mathsf{P}^i_{\mathsf{i}}(\mathsf{out}_i) = \{\mathsf{p}_i, \mathsf{m}, \mathsf{d}\} \qquad \mathsf{P}^i_{\mathsf{r}}(\mathsf{out}_i) = \{\mathsf{c}_i\}$$

We are now left with the challenging task of specifying the policy for the (dyadic) channel ch. We have policies for the tag field ($\mathsf{ch}.1$) as well as the payload ($\mathsf{ch}.2$) of the messages and we may want to record this as follows (for $i = 1, 2$):

$$\begin{array}{ll} \mathsf{P}^i_{\mathsf{i}}(\mathsf{ch}.1) = \{\mathsf{m}\} & \mathsf{P}^i_{\mathsf{r}}(\mathsf{ch}.1) = \{\mathsf{d}, \mathsf{c}_i\} \\ \mathsf{P}^i_{\mathsf{i}}(\mathsf{ch}.2) = \{\mathsf{p}_i, \mathsf{m}\} & \mathsf{P}^i_{\mathsf{r}}(\mathsf{ch}.2) = \{\mathsf{d}, \mathsf{c}_i\} \end{array} \tag{1}$$

In the case of the payload we state that p_i and m may be influencers of the data, whereas d and c_i will be the permitted readers of the data. For the tag field we can omit p_i from the set of influencers but otherwise the policy equals that of the payload.

Unfortunately, information flow policies that are *not* content-based (like DLM) do *not* allow us to have two distinct policies for the channel ch. This means that we would need to settle for a policy merging the policies P^1 and P^2. In particular, we would be forced to include both p_1 and p_2 as influencers of

both out_1 and out_2. This means that the policy would be *unable* to provide the required guarantees for the system.

Our Contribution: This motivates providing the desired security guarantees by introducing *disjunctive* policies into a concurrent language with synchronous communication. In this approach we allow the channel ch to have the disjunctive policy $\{\mathsf{P}^1, \mathsf{P}^2\}$. To reduce the uncertainty as to which of the policies P^1 and P^2 that actually applies we next incorporate a *value-range* component in our policies. We will extend the policies of (1) with a record of the value of the tag component of the messages (for $i = 1, 2$):

$$\mathsf{P}^i_{\mathsf{v}}(\mathsf{ch}.1) = \{i\}$$

Our subsequent analysis is based on a combined type system and Hoare logic that allows us to reason about the values of variables and hence the value of the tag field of the messages. In this way our analysis allows us to guarantee that data from p_1 only reaches c_1 and data from p_2 only reaches c_2.

3 Syntax and Instrumented Semantics

Preparing for the formal development we define the concurrent imperative language used and we develop its instrumented operational semantics.

Syntax of Processes and Systems: A system consists of a fixed number of *principals* running in parallel; each principal runs a process with its own local state and exchanges messages with other principals by synchronous communication over channels. The syntax of processes (or statements) S, arithmetic expressions a, boolean expressions b and systems Sys is:

$$\begin{aligned} S &::= \mathsf{skip} \mid x := a \mid S_1; S_2 \mid \mathsf{if}\ b\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2 \mid \mathsf{while}\ b\ \mathsf{do}\ S \\ &\quad \mid\ ch?x_1..x_k \mid ch!a_1..a_k \mid S_1 \oplus S_2 \mid \{X\}\, S \\ a &::= n \mid x \mid a_1\ op\ a_2 \\ b &::= \mathsf{true} \mid a_1\ rel\ a_2 \\ Sys &::= \mathsf{p}_1 : S_1 \mid\mid \cdots \mid\mid \mathsf{p}_n : S_n \end{aligned}$$

We write $x, y, z \in \mathbf{Var}$ for variables, $X \subseteq \mathbf{Var}$ for sets of variables, and $p \in \mathbf{Pr}$ for principals. We use ch for a polyadic channel name, n for unspecified constants, op for unspecified arithmetic operators, rel for unspecified relational operators, true for the boolean constant denoting truth, and we let u range over $\mathbf{Var} \cup \mathbf{Ch}$. We assume that $\mathbf{Pr} = \{\mathsf{p}_1, \cdots, \mathsf{p}_n\}$ is the set of principals, $\mathbf{Var} = \biguplus_{p \in \mathbf{Pr}} \mathbf{Var}_p$ is the union of mutually disjoint sets $\mathbf{Var}_p$ of variables, where each principal p is only allowed to use variables from $\mathbf{Var}_p$, and $\mathbf{Ch} = \{ch.1, \cdots, ch.k \mid ch \text{ is a polyadic channel name with arity } k\}$ is the set of *channel positions*. Arithmetic and boolean expressions may contain variables but neither channels nor principals. We denote by $\mathsf{fv}(\cdot)$ the free variables occurring inside arithmetic and boolean expressions.

The statements are mostly self-explanatory; $ch?x_1..x_k$ denotes the input of a k-tuple of variables over the channel ch and the assignment of the components to $x_1..x_k$, $ch!a_1..a_k$ denotes the output of a k-tuple of values over the channel ch, and $S_1 \oplus S_2$ denotes the "external" non-deterministic choice between S_1 and S_2. The statement $\{X\}\, S$ will be explained below; it will arise only during execution in the manner of Structural Operational Semantics [20].

Instrumented Semantics for Processes: The semantics is based on a standard Structural Operational Semantics [20] where the states are mappings from variables to values, i.e. $\sigma \in \mathbf{Var} \to \mathbf{Val}$. The instrumentation amounts to adding *flows* to the transitions; a flow F is a subset of pairs of variables, channels and principals

$$F \subseteq (\mathbf{Var} \cup \mathbf{Ch} \cup \mathbf{Pr}) \times (\mathbf{Var} \cup \mathbf{Ch} \cup \mathbf{Pr})$$

and it provides a *precise* record of the *explicit* and *implicit* information flow. The intuitive idea is that the value of the first component of a pair may influence the value of the second component; in case the component is a channel position $ch.i$ we refer to the value being communicated in the i'th position of the channel and in the case the component is a principal p it is instructive to think of it as the program counter for the process p.

In order to handle communication, the transitions are also annotated with the *action* taking place; an action α takes one of three forms:

$$\alpha ::= ch!v_1..v_k \mid ch?v_1..v_k \mid \tau$$

where the first two are for output and input over the channel ch and τ is an internal action; here $v_1..v_k$ denotes the sequence of values (from $\mathbf{Val}$) being communicated over the channel. We tacitly assume that arities match without having explicitly to require this in the semantics.

The general form of the transitions for processes is

$$\vdash_p \langle S; \sigma\rangle \xrightarrow[\alpha]{F} \langle S'; \sigma'\rangle$$

where the subscript p indicates the principal in which the process resides; here configurations of the form $\langle \mathsf{skip}; \sigma\rangle$ serve as terminal configurations. The definition is given in Fig. 2 (ignoring the two last rules concerned with systems) and the most interesting clauses are explained below.

First, in the clause for assignment the flow clearly should include $\mathsf{fv}(a) \times \{x\}$ as the values of the free variables of a are used to compute the value of x. Additionally we include (p, x) as the program counter of p also influences the value of x. Furthermore, the program counter is also influenced by the assignment so we also record the flow $(\mathsf{fv}(a) \cup \{p\}) \times \{p\}$. The clause for skip can be viewed as a special case only recording the flow $\{(p, p)\}$ to express that a process owned by principal p was active.

In the clause for conditionals and iteration we construct a block construct of the form $\{\mathsf{fv}(b)\}\, S$ for recording the implicit flow that result from passing the boolean condition b before embarking on the process S. This is in line with the

$$\vdash_p \langle \mathsf{skip}; \sigma\rangle \xrightarrow[\tau]{F} \langle \mathsf{skip}; \sigma\rangle \quad \text{if } F = \{p\} \times \{p\}$$

$$\vdash_p \langle x := a; \sigma\rangle \xrightarrow[\tau]{F} \langle \mathsf{skip}; \sigma[x \mapsto \mathcal{A}[\![a]\!]\sigma]\rangle \quad \text{if } F = (\mathsf{fv}(a) \cup \{p\}) \times \{x, p\}$$

$$\frac{\vdash_p \langle S_1; \sigma\rangle \xrightarrow[\alpha]{F} \langle S_1'; \sigma'\rangle}{\vdash_p \langle S_1; S_2; \sigma\rangle \xrightarrow[\alpha]{F} \langle S_1'; S_2; \sigma'\rangle} \text{ if } S_1' \neq \mathsf{skip} \qquad \frac{\vdash_p \langle S_1; \sigma\rangle \xrightarrow[\alpha]{F} \langle \mathsf{skip}; \sigma'\rangle}{\vdash_p \langle S_1; S_2; \sigma\rangle \xrightarrow[\alpha]{F} \langle S_2; \sigma'\rangle}$$

$$\vdash_p \langle \mathsf{if}\ b\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2; \sigma\rangle \xrightarrow[\tau]{F} \langle \{\mathsf{fv}(b)\}\, S_1; \sigma\rangle \quad \text{if} \quad \begin{array}{l}\mathcal{B}[\![b]\!]\sigma = \mathsf{true} \text{ and} \\ F = (\mathsf{fv}(b) \cup \{p\}) \times \{p\}\end{array}$$

$$\vdash_p \langle \mathsf{if}\ b\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2; \sigma\rangle \xrightarrow[\tau]{F} \langle \{\mathsf{fv}(b)\}\, S_2; \sigma\rangle \quad \text{if} \quad \begin{array}{l}\mathcal{B}[\![b]\!]\sigma = \mathsf{false} \text{ and} \\ F = (\mathsf{fv}(b) \cup \{p\}) \times \{p\}\end{array}$$

$$\vdash_p \langle \mathsf{while}\ b\ \mathsf{do}\ S; \sigma\rangle \xrightarrow[\tau]{F} \langle (\{\mathsf{fv}(b)\}\, S); \mathsf{while}\ b\ \mathsf{do}\ S; \sigma\rangle \quad \text{if} \quad \begin{array}{l}\mathcal{B}[\![b]\!]\sigma = \mathsf{true} \text{ and} \\ F = (\mathsf{fv}(b) \cup \{p\}) \times \{p\}\end{array}$$

$$\vdash_p \langle \mathsf{while}\ b\ \mathsf{do}\ S; \sigma\rangle \xrightarrow[\tau]{F} \langle \mathsf{skip}; \sigma\rangle \quad \text{if } \mathcal{B}[\![b]\!]\sigma = \mathsf{false} \text{ and } F = (\mathsf{fv}(b) \cup \{p\}) \times \{p\}$$

$$\vdash_p \langle ch!a_1..a_k; \sigma\rangle \xrightarrow[ch!v_1..v_k]{F} \langle \mathsf{skip}; \sigma\rangle \quad \text{if} \quad \begin{array}{l}v_i = \mathcal{A}[\![a_i]\!]\sigma \text{ (for all } i) \text{ and} \\ F = \bigcup_{i \le k} (\mathsf{fv}(a_i) \cup \{p\}) \times \{ch.i, p\}\end{array}$$

$$\vdash_p \langle ch?x_1..x_k; \sigma\rangle \xrightarrow[ch?v_1..v_k]{F} \langle \mathsf{skip}; \sigma[(x_i \mapsto v_i)_{i \le k}]\rangle \quad \text{if } F = \bigcup_{i \le k} \{ch.i, p\} \times \{x_i, p\}$$

$$\frac{\vdash_p \langle S_i; \sigma\rangle \xrightarrow[\alpha]{F} \langle S_i'; \sigma'\rangle}{\vdash_p \langle S_1 \oplus S_2; \sigma\rangle \xrightarrow[\alpha]{F} \langle S_i'; \sigma'\rangle} \text{ for } i = 1, 2$$

$$\frac{\vdash_p \langle S; \sigma\rangle \xrightarrow[\alpha]{F} \langle S'; \sigma'\rangle}{\vdash_p \langle \{X\}\, S; \sigma\rangle \xrightarrow[\alpha]{\{X\}\, F} \langle \{X\}\, S'; \sigma'\rangle} \text{ if } S' \neq \mathsf{skip} \qquad \frac{\vdash_p \langle S; \sigma\rangle \xrightarrow[\alpha]{F} \langle \mathsf{skip}; \sigma'\rangle}{\vdash_p \langle \{X\}\, S; \sigma\rangle \xrightarrow[\alpha]{\{X\}\, F} \langle \mathsf{skip}; \sigma'\rangle}$$

$$\frac{\vdash_{\mathsf{p}_i} \langle S_i; \sigma\rangle \xrightarrow[\tau]{F} \langle S_i'; \sigma'\rangle}{\langle \mathsf{p}_1 : S_1 \mid\mid \cdots \mid\mid \mathsf{p}_i : S_i \mid\mid \cdots \mid\mid \mathsf{p}_n : S_n; \sigma\rangle \overset{F}{\Longrightarrow} \langle \mathsf{p}_1 : S_1 \mid\mid \cdots \mid\mid \mathsf{p}_i : S_i' \mid\mid \cdots \mid\mid \mathsf{p}_n : S_n; \sigma'\rangle}$$

$$\frac{\vdash_{\mathsf{p}_i} \langle S_i; \sigma\rangle \xrightarrow[ch!v_1..v_k]{F_i} \langle S_i'; \sigma'\rangle \qquad \vdash_{\mathsf{p}_j} \langle S_j; \sigma'\rangle \xrightarrow[ch?v_1..v_k]{F_j} \langle S_j'; \sigma''\rangle}{\langle \cdots \mid\mid \mathsf{p}_i : S_i \mid\mid \cdots \mid\mid \mathsf{p}_j : S_j \mid\mid \cdots; \sigma\rangle \overset{\mathcal{F}}{\Longrightarrow} \langle \cdots \mid\mid \mathsf{p}_i : S_i' \mid\mid \cdots \mid\mid \mathsf{p}_j : S_j' \mid\mid \cdots; \sigma''\rangle}$$
$$\text{if } \mathcal{F} = (F_i \circ \mathbb{I}_{\mathbf{Ch}} \circ F_j) \cup (F_i \circ \mathbb{I}_{\mathbf{Var} \cup \mathbf{Pr}}) \cup (\mathbb{I}_{\mathbf{Var} \cup \mathbf{Pr}} \circ F_j) \text{ (and } i \neq j)$$

Fig. 2. Instrumented semantics of processes and systems.

treatment of implicit flows using block labels [12,17,18] and technically uses the technique of "local environments" developed in Structural Operational Semantics [20].

This then requires us to define the semantics of the block construct just created and in the clause for $\{X\}\, S$ we use the operation $\{X\}\, F$ defined by

$$\{X\}\, F = F \cup (X \times \mathsf{snd}(F))$$

where $\mathsf{snd}(F)$ is the projection on the second components of the pairs in F. In this way the implicit dependence on the variables of X is incorporated in the flow of the statement.

The flows constructed for input and output are easiest to understand if $ch!a$ is thought of as $ch := a$ and $ch?x$ is thought of as $x := ch$ with the obvious extensions to polyadic output and input. Note that these clauses introduce the channels in the flows and that this only happens when the action α is different from τ.

Example 1. *Consider the process* d *of Sect. 2 and assume that it performs the action* $\mathsf{out}_1!z$. *This will give rise to the flow*

$$F = \{(z, \mathsf{out}_1), (z, \mathsf{d}), (\mathsf{d}, \mathsf{out}_1), (\mathsf{d}, \mathsf{d})\}$$

However there is an implicit dependence on the variable y *of the test of the conditional so the resulting flow will be* $\{y\}F$ *which will add the two pairs* (y, out_1) *and* (y, d) *to* F.

Instrumented Semantics for Systems: The configurations now take the form $\langle \mathsf{p}_1 : S_1 \parallel \cdots \parallel \mathsf{p}_n : S_n; \sigma \rangle$. Since we assumed that the n processes have mutually disjoint sets of variables no confusion arises by using $\sigma : \mathbf{Var} \to \mathbf{Val}$ to denote the state of the combined system. The transitions have the form

$$\langle \mathsf{p}_1 : S_1 \parallel \cdots \parallel \mathsf{p}_n : S_n; \sigma \rangle \stackrel{\mathcal{F}}{\Longrightarrow} \langle \mathsf{p}_1 : S_1' \parallel \cdots \parallel \mathsf{p}_n : S_n'; \sigma' \rangle$$

where $\mathcal{F}$ is a *system flow* meaning that it is a flow that does not mention any channels and hence

$$\mathcal{F} \subseteq (\mathbf{Var} \cup \mathbf{Pr}) \times (\mathbf{Var} \cup \mathbf{Pr})$$

The semantics is defined by the last two rules in Fig. 2. The first rule embeds a process action not involving communication (as indicated by the τ annotation on the arrow) in the system level; since no communication takes place there will be no mentioning of channels in F so indeed $F \subseteq (\mathbf{Var} \cup \mathbf{Pr}) \times (\mathbf{Var} \cup \mathbf{Pr})$. The second rule takes care of communication between processes; here we need to combine the flows from the two processes taking part in the communication. First we have the flow resulting from the communication over the channel (written $F_i \circ \mathbb{I}_{\mathbf{Ch}} \circ F_j$), then we have the remaining flows from the two processes (written $F_i \circ \mathbb{I}_{\mathbf{Var} \cup \mathbf{Pr}}$ and $\mathbb{I}_{\mathbf{Var} \cup \mathbf{Pr}} \circ F_j$); here we write $\mathbb{I}_Y$ for the identity relation on the set Y (for Y being $\mathbf{Var} \cup \mathbf{Pr}$ or $\mathbf{Ch}$) and use this relation to select the relevant part of the flows F_i and F_j. Note that the resulting flow $\mathcal{F}$ is indeed a subset of $(\mathbf{Var} \cup \mathbf{Pr}) \times (\mathbf{Var} \cup \mathbf{Pr})$.

Example 2. *Returning to Sect. 2, suppose that* p_1 *performs the operation* $\mathsf{in}_1!v$ *for some value* v *and that* m *is ready to perform the operation* $\mathsf{in}_1?x_1$*. The flow constructed for one step of execution of* p_1 *and* m *will then be*

$$F_1 = \{(\mathsf{p}_1, \mathsf{in}_1), (\mathsf{p}_1, \mathsf{p}_1)\}$$
$$F_2 = \{(\mathsf{in}_1, \mathsf{x}_1), (\mathsf{in}_1, \mathsf{m}), (\mathsf{m}, \mathsf{x}_1), (\mathsf{m}, \mathsf{m})\}$$

These two flows are then combined into a flow for the overall communication:

$$\mathcal{F} = \{(\mathsf{p}_1, \mathsf{x}_1), (\mathsf{p}_1, \mathsf{m}), (\mathsf{p}_1, \mathsf{p}_1), (\mathsf{m}, \mathsf{x}_1), (\mathsf{m}, \mathsf{m})\}$$

Here the first two pairs come from the flow through the channel, the third pair comes from F_1 *and the last two pairs come from* F_2*.*

4 Security Policies

We now introduce our security policies. We start with so-called *basic policies* for influencers and readers, where we borrow ideas from [17,18]. We extend on DLM in that the basic policies are content-based thanks to a component for value ranges. We next introduce so-called *disjunctive policies* that are sets of basic policies; they are needed to deal with the challenges illustrated in the multiplexer example in Sect. 2.

Basic Policies: Our basic policies provide information for each variable and channel about the principals that might have *influenced* their values, about the principals that might be allowed to *read* their values, and about their actual *value range*. Formally, a basic policy P is given by three component mappings

$$P_{\mathsf{i}} : \mathbf{Pol}_{\mathsf{i}} = (\mathbf{Var} \cup \mathbf{Ch}) \to \mathbf{Lab}_{\mathsf{i}} \text{ influencers}$$
$$P_{\mathsf{r}} : \mathbf{Pol}_{\mathsf{r}} = (\mathbf{Var} \cup \mathbf{Ch}) \to \mathbf{Lab}_{\mathsf{r}} \text{ readers}$$
$$P_{\mathsf{v}} : \mathbf{Pol}_{\mathsf{v}} = (\mathbf{Var} \cup \mathbf{Ch}) \to \mathbf{Lab}_{\mathsf{v}} \text{ value range}$$

where $\mathbf{Lab}_{\mathsf{i}} = \wp(\mathbf{Pr})$, $\mathbf{Lab}_{\mathsf{r}} = \wp(\mathbf{Pr})^{op}$, and $\mathbf{Lab}_{\mathsf{v}} = \wp(\mathbf{Val})$.

The orderings $\sqsubseteq$ on $\mathbf{Lab}_{\mathsf{i}}$, $\mathbf{Lab}_{\mathsf{r}}$ and $\mathbf{Lab}_{\mathsf{v}}$ are obtained from those of the powersets: for $\mathbf{Lab}_{\mathsf{i}}$ it is the subset ordering $\subseteq$ on the powerset $\wp(\mathbf{Pr})$ of principals, for $\mathbf{Lab}_{\mathsf{r}}$ it is the superset ordering $\supseteq$ on $\wp(\mathbf{Pr})$ (because the notation $\wp(\mathbf{Pr})^{op}$ indicates that the natural ordering is the *opposite*, or *dual*, of the one for powersets), and for $\mathbf{Lab}_{\mathsf{v}}$ it is the subset ordering $\subseteq$ on $\wp(\mathbf{Val})$. The orderings are lifted to policy components and basic policies in a pointwise manner.

Flows Adhering to Basic Policies: We shall now define a predicate $\mathsf{sec}(P, F, P')$ that specifies when a flow F adheres to the basic policies P and P'; here P will be the policy that is relevant before the flow F whereas P' is the policy that is relevant after the flow F. As we shall see P will primarily be used to provide information about the permitted readers whereas P' will primarily be used to provide information about the permitted influencers.

We define the predicate $\mathsf{sec}(P, F, P')$ by

$$\begin{array}{l}\forall(p,u') \in F : p \in P'_{\mathsf{i}}(u') \wedge \\ \forall(u,u') \in F : (P_{\mathsf{i}}(u) \sqsubseteq P'_{\mathsf{i}}(u') \wedge P_{\mathsf{r}}(u) \sqsubseteq P'_{\mathsf{r}}(u')) \wedge \\ \forall(u,p') \in F : p' \in P_{\mathsf{r}}(u) \wedge \\ \forall y \in \mathbf{Var} \setminus (\mathsf{snd}(F)) : (P_{\mathsf{i}}(y) \sqsubseteq P'_{\mathsf{i}}(y) \wedge P_{\mathsf{r}}(y) \sqsubseteq P'_{\mathsf{r}}(y))\end{array}$$

The first line of the definition of $\mathsf{sec}(P, F, P')$ ensures that the principals p recorded as an influencer of the variable or channel u' is indeed permitted to be an influencer according to the resulting policy P'. The third line is analogous and ensures that the principal p' recorded as a reader of the variable or channel u is indeed permitted to be a reader according to the initial security policy P. The second line extends these considerations to the flow recorded between variables and channels. Note that the definition considers the constraints on influencers and readers to go in *opposite directions*: the partial order $\sqsubseteq$ amounts to $\subseteq$ in the case of influencers and to $\supseteq$ in the case of readers. This observation is central for the *duality* between the treatment of influencers and readers in our information flow type system; it expresses that it is always secure to remove readers and to add influencers. The fourth line merely ensures that we only make secure changes to the policies for variables *not* recorded in the flow: we may include more influencers and we may remove some readers for these variables.

The definition of $\mathsf{sec}(P, F, P')$ simplifies a bit when F is a system flow $\mathcal{F}$ in that u and u' now only need to range over variables rather than variables and channels.

Disjunctive Policies: We shall introduce *disjunctive policies* $\mathcal{P}$ to be finite sets $\{P^1, \cdots, P^m\}$ of basic policies each having the three components $P^i_{\mathsf{i}} \in \mathbf{Pol}_{\mathsf{i}}$, $P^i_{\mathsf{r}} \in \mathbf{Pol}_{\mathsf{r}}$, and $P^i_{\mathsf{v}} \in \mathbf{Pol}_{\mathsf{v}}$ as explained above. Intuitively, this corresponds to a disjunctive formula of basic policies where each basic policy only uses conjunction.

The state σ of a configuration in the semantics will determine whether or not a policy P of $\mathcal{P}$ applies in that configuration. We write $\sigma \models \overline{P_{\mathsf{v}}}$ to mean $\forall x \in \mathbf{Var} : \sigma(x) \in P_{\mathsf{v}}(x)$ and use $\overline{P_{\mathsf{v}}}$ for the logical formula $\bigwedge_{x \in \mathbf{Var}} x \in P_{\mathsf{v}}(x)$.

We do *not* require that for each σ there exists $P \in \mathcal{P}$ such that $\sigma \models \overline{P_{\mathsf{v}}}$ because there may be states that do not conform to the desired policy. Also we do *not* require that for each σ there exists at most one $P \in \mathcal{P}$ such that $\sigma \models \overline{P_{\mathsf{v}}}$ although this may be a natural property to arrange in many cases and may make the subsequent development more intuitive. In fact, the notation would come close to what could be expressed using a notion of dependent types which would constitute a more intuitive interface for the industrial programmer.

Example 3. *Returning to the motivating example we consider the disjunctive policy* $\{\mathsf{P}^1, \mathsf{P}^2\}$ *consisting of just two basic policies. For the channels the specification is given already in Sect. 2 and for the variables it is given by the following table (for* $i, j \in \{1, 2\}$*):*

	x_j	y	z
$\mathsf{P}^i_{\mathsf{r}}$	$\{\mathsf{c}_j, \mathsf{m}, \mathsf{d}\}$	$\{\mathsf{c}_1, \mathsf{c}_2, \mathsf{d}\}$	$\{\mathsf{c}_i, \mathsf{d}\}$
$\mathsf{P}^i_{\mathsf{i}}$	$\{\mathsf{p}_j, \mathsf{m}\}$	$\{\mathsf{m}, \mathsf{d}\}$	$\{\mathsf{p}_i, \mathsf{m}, \mathsf{d}\}$
$\mathsf{P}^i_{\mathsf{v}}$	$\mathbb{Z}$	$\{i\}$	$\mathbb{Z}$

When no explicit specification is given, the policy is the least restrictive, allowing no influencers, all readers and all values. Note that the policy for y (and indeed also ch.1*) allows* c_1 *as well as* c_2 *to learn the outcome of the test on the first component of the message exchanged over* ch*.*

Systems Adhering to Disjunctive Policies: We are now ready to explain when a system adheres to a disjunctive policy. We shall take a co-inductive approach and formulate a self-simulation condition in the manner of bi-simulation.

It will be useful to consider systems together with their preconditions. We shall write

$$\{\phi_1 \wedge \cdots \wedge \phi_n\}\mathsf{p}_1 : S_1 \parallel \cdots \parallel \mathsf{p}_n : S_n$$

for a system $\mathsf{p}_1 : S_1 \parallel \cdots \parallel \mathsf{p}_n : S_n$ that is intended only to be started in a state σ satisfying the logical formula $\phi_1 \wedge \cdots \wedge \phi_n$. We shall require that the free variables of the formula ϕ_i are contained in $\mathbf{Var}_{\mathsf{p}_i}$ (the variables belonging to the process p_i) thereby ensuring that ϕ_i only applies to S_i. The simplest choice of $\phi_1 \wedge \cdots \wedge \phi_n$ would be $\mathsf{true} \wedge \cdots \wedge \mathsf{true}$ and we sometimes abbreviate it to true; the usefulness of considering other choices of $\phi_1 \wedge \cdots \wedge \phi_n$ will emerge in the next section.

Definition 1. *A predicate* $\mathcal{R}$ *on systems with preconditions is a* self-simulation *with respect to* $\mathcal{P}$ *whenever*

$$\mathcal{R}(\{\phi_1 \wedge \cdots \wedge \phi_n\}\mathsf{p}_1 : S_1 \parallel \cdots \parallel \mathsf{p}_n : S_n)$$

implies that

$$\begin{array}{l}
\forall \sigma, \sigma', S'_1 \cdots S'_n, \mathcal{F} : \\
\quad \langle \mathsf{p}_1 : S_1 \parallel \cdots \parallel \mathsf{p}_n : S_n ; \sigma \rangle \stackrel{\mathcal{F}}{\Longrightarrow} \langle \mathsf{p}_1 : S'_1 \parallel \cdots \parallel \mathsf{p}_n : S'_n ; \sigma' \rangle \\
\Downarrow \\
\quad \exists \phi'_1, \cdots, \phi'_n : \\
\qquad \mathcal{R}(\{\phi'_1 \wedge \cdots \wedge \phi'_n\}\mathsf{p}_1 : S'_1 \parallel \cdots \parallel \mathsf{p}_n : S'_n) \wedge \\
\qquad \forall P \in \mathcal{P} : \ \sigma \models (\phi_1 \wedge \cdots \wedge \phi_n \wedge \overline{P_{\mathsf{v}}}) \\
\qquad\qquad \Downarrow \\
\qquad\qquad \exists P' \in \mathcal{P} : \sigma' \models (\phi'_1 \wedge \cdots \wedge \phi'_n \wedge \overline{P'_{\mathsf{v}}}) \ \wedge \ \mathsf{sec}(P, \mathcal{F}, P')
\end{array}$$

The self-simulation part of the definition expresses that for all states, whenever one system configuration evolves into another system configuration and the first system is in the relation $\mathcal{R}$ for a certain precondition, then there is an updated precondition for the resulting system ensuring that it also is in the relation $\mathcal{R}$. The last three lines of the definition put extra requirements on the relationship between the states, preconditions, policies and flows: whenever the initial state satisfies the preconditions and some policy from $\mathcal{P}$ applies, then

there must be some policy in $\mathcal{P}$ such that the next configuration satisfies the updated preconditions and the latter policy applies; furthermore, the observed flow $\mathcal{F}$ has to be acceptable with respect to the two policies. These requirements may be easiest to appreciate in the case where the policy set $\mathcal{P}$ satisfies that for each σ there is at most one (or perhaps exactly one) $P \in \mathcal{P}$ such that $\sigma \models \overline{P_{\mathsf{v}}}$.

As in the case of bi-simulation it is immediate to show that any union of self-simulations is itself a self-simulation. This allows us to define self-similarity in the same co-inductive manner as used for bi-similarity:

Definition 2. Self-similarity *(with respect to $\mathcal{P}$) is the largest self-simulation (with respect to $\mathcal{P}$) and it is denoted $\models^{\mathcal{P}}$ (or simply $\models$).*

A system $\mathsf{p}_1 : S_1 \mid\mid \cdots \mid\mid \mathsf{p}_n : S_n$ respects the disjunctive policy $\mathcal{P}$ whenever it is self-similar:

$$\models^{\mathcal{P}} \{\mathsf{true} \wedge \cdots \wedge \mathsf{true}\}\mathsf{p}_1 : S_1 \mid\mid \cdots \mid\mid \mathsf{p}_n : S_n$$

5 Type System and Correctness

Given a disjunctive policy we now specify a type system for ensuring that processes and systems obey the policy and we prove that a well-typed system respects the disjunctive policy. To deal with the value-range components of policies, the type system is combined with a Hoare logic for reasoning about the values of variables [3,4]. We already mentioned that using a Hoare logic allows to cleanly incorporate also the results of prior static analyses into the information flow type system.

Type System: The Hoare logic part of the type system is fairly simple because we use local variables and synchronous communication (in the manner of MILS [21] and ARINC-811 [10]) rather than shared variables between processes [23]. The judgement of the type system for processes has the form

$$X \vdash_p \{\phi\} S \{\phi'\}$$

where X is a set of implicitly used variables, p is the name of the principal in which the process S executes, and ϕ and ϕ' are the pre- and post-conditions of S in the form of logical formulae over program variables in $\mathbf{Var}_p$. (We shall assume that each $\mathbf{Var}_p$ is sufficiently big to account for all logical variables needed in the Hoare logic.)

The definition is given in Fig. 3 and requires some auxiliary notation. Given a set $X \subseteq \mathbf{Var}$ we then define the mappings $P_{\mathsf{i}}[X]$ and $P_{\mathsf{r}}[X]$ in $\mathbf{Lab}_{\mathsf{i}}$ and $\mathbf{Lab}_{\mathsf{r}}$, respectively, by taking least upper bounds over the variables in X; to be specific:

$$P_{\mathsf{i}}[X] = \bigcup_{x \in X} P_{\mathsf{i}}(x) \qquad P_{\mathsf{r}}[X] = \bigcap_{x \in X} P_{\mathsf{r}}(x)$$

In Fig. 3 we shall use $P_{\mathsf{r}}[a; X]$ as a shorthand for $P_{\mathsf{r}}[\mathsf{fv}(a) \cup X]$ and $P_{\mathsf{i}}[a; X]$ as a shorthand for $P_{\mathsf{i}}[\mathsf{fv}(a) \cup X]$ and similarly for boolean expressions b instead of arithmetic expressions a. This notation is often used in an expression of the form

$$X \vdash_p \{\phi\}\mathsf{skip}\{\phi'\} \quad \text{if } \forall P \in \mathcal{P} : \phi \wedge \overline{P_{\mathsf{v}}} \;\Rightarrow\; \phi' \wedge p \in P_{\mathsf{r}}[X]$$

$$X \vdash_p \{\phi\}x := a\{\phi'\} \quad \text{if } \forall P \in \mathcal{P} : \phi \wedge \overline{P_{\mathsf{v}}} \;\Rightarrow\; \exists P' \in \mathcal{P} : \left(\begin{array}{l} \phi'[a/x] \wedge \overline{P'_{\mathsf{v}}}[a/x] \wedge \\ P_{\mathsf{i}}[x \mapsto P_{\mathsf{i}}[a; X]] \sqsubseteq P'_{\mathsf{i}} \;\wedge\; p \in P'_{\mathsf{i}}(x) \wedge \\ P_{\mathsf{r}}[x \mapsto P_{\mathsf{r}}[a; X]] \sqsubseteq P'_{\mathsf{r}} \;\wedge\; p \in P_{\mathsf{r}}[a; X] \end{array} \right)$$

$$\frac{X \vdash_p \{\phi\}S_1\{\phi''\} \qquad X \vdash_p \{\phi''\}S_2\{\phi'\}}{X \vdash_p \{\phi\}S_1; S_2\{\phi'\}}$$

$$\frac{\begin{array}{l} X \cup \mathsf{fv}(b) \vdash_p \{\phi \wedge b\}S_1\{\phi'\} \\ X \cup \mathsf{fv}(b) \vdash_p \{\phi \wedge \neg b\}S_2\{\phi'\} \end{array}}{X \vdash_p \{\phi\}\mathsf{if}\ b\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\{\phi'\}} \quad \text{if } \forall P \in \mathcal{P} : \phi \wedge \overline{P_{\mathsf{v}}} \;\Rightarrow\; p \in P_{\mathsf{r}}[b; X]$$

$$\frac{X \cup \mathsf{fv}(b) \vdash_p \{\phi \wedge b\}S\{\phi\}}{X \vdash_p \{\phi\}\mathsf{while}\ b\ \mathsf{do}\ S\{\phi \wedge \neg b\}} \quad \text{if } \forall P \in \mathcal{P} : \phi \wedge \overline{P_{\mathsf{v}}} \;\Rightarrow\; p \in P_{\mathsf{r}}[b; X]$$

$$\frac{X \cup X_0 \vdash_p \{\phi\}S\{\phi'\}}{X \vdash_p \{\phi\}\{X_0\}\,S\{\phi'\}}$$

$$X \vdash_p \{\phi\}ch!a_1..a_k\{\phi'\} \quad \text{if } \forall P \in \mathcal{P} : \phi \wedge \overline{P_{\mathsf{v}}} \;\Rightarrow\; \exists P' \in \mathcal{P} : \left(\begin{array}{l} \phi' \wedge \overline{P'_{\mathsf{v}}} \wedge \bigwedge_{i \le k} a_i \in P'_{\mathsf{v}}(ch.i) \wedge \\ P_{\mathsf{i}}[(ch.i \mapsto P_{\mathsf{i}}[a_i; X])_{i \le k}] \sqsubseteq P'_{\mathsf{i}} \wedge \bigwedge_{i \le k} p \in P'_{\mathsf{i}}(ch.i) \\ P_{\mathsf{r}}[(ch.i \mapsto P_{\mathsf{r}}[a_i; X])_{i \le k}] \sqsubseteq P'_{\mathsf{r}} \wedge \bigwedge_{i \le k} p \in P_{\mathsf{r}}[a_i; X] \end{array} \right)$$

$$X \vdash_p \{\phi\}ch?x_1..x_k\{\phi'\} \quad \text{if } \forall P \in \mathcal{P} : \left(\begin{array}{l} (\exists x_1..x_k.\phi \wedge \overline{P_{\mathsf{v}}}) \quad \wedge \\ \bigwedge_{i \le k} x_i \in P_{\mathsf{v}}(ch.i) \end{array} \right) \Rightarrow\; \exists P' \in \mathcal{P} : \left(\begin{array}{l} \phi' \wedge \overline{P'_{\mathsf{v}}} \wedge \\ P_{\mathsf{i}}[(x_i \mapsto P_{\mathsf{i}}[ch.i; X])_{i \le k}] \sqsubseteq P'_{\mathsf{i}} \wedge \bigwedge_{i \le k} p \in P'_{\mathsf{i}}(x_i) \\ P_{\mathsf{r}}[(x_i \mapsto P_{\mathsf{r}}[ch.i; X])_{i \le k}] \sqsubseteq P'_{\mathsf{r}} \wedge \bigwedge_{i \le k} p \in P_{\mathsf{r}}[ch.i; X] \end{array} \right)$$

$$\frac{X \vdash_p \{\phi\}S_1\{\phi'\} \qquad X \vdash_p \{\phi\}S_2\{\phi'\}}{X \vdash_p \{\phi\}S_1 \oplus S_2\{\phi'\}} \quad \text{if } \forall P \in \mathcal{P} : \phi \wedge \overline{P_{\mathsf{v}}} \;\Rightarrow\; p \in P_{\mathsf{r}}[X]$$

$$\frac{X \vdash_p \{\psi\}S\{\psi'\}}{X \vdash_p \{\phi\}S\{\phi'\}} \quad \text{if } (\phi \Rightarrow \psi) \wedge (\psi' \Rightarrow \phi')$$

Fig. 3. Type system for processes.

$P_{\mathsf{r}}[x \mapsto P_{\mathsf{r}}[a;X]]$ that denotes P'_{r} defined by $P'_{\mathsf{r}}(u) = P_{\mathsf{r}}(u)$ whenever $u \neq x$ and $P'_{\mathsf{r}}(x) = P_{\mathsf{r}}[a;X]$ and similarly for $P_{\mathsf{i}}[x \mapsto P_{\mathsf{i}}[a;X]]$.

Most axiom schemes and rules in Fig. 3 strengthen a precondition of the form ϕ to the formula $\phi \wedge \overline{P_{\mathsf{v}}}$ that allows us to use the value-range information from the appropriate basic policy in $\mathcal{P}$. With the exception of assignment, input and output, most axiom schemes and rules demand that the strengthened precondition ensures that the principal p is correctly recorded as a reader of the variables whose values are either used implicitly (typically by being a member of the set X) or explicitly (typically $\mathsf{fv}(b)$). As an example, the axiom scheme for skip illustrates both points.

The axiom scheme for assignment is more complex because the state of the system changes. It therefore considers all basic policies P whose value-range component is consistent with the (strengthened) precondition and demands that there is a basic policy P' that appropriately records the state change. The pattern $\forall P \in \mathcal{P} : (\cdots) \Rightarrow \exists P' \in \mathcal{P} : (\cdots)$ takes care of this and is in line with the definition of self-simulation. The first line of requirements ensures that the strengthened precondition establishes the formulae obtained from the strengthened postcondition by mimicking the effect of the assignment; the use of a substitution $[a/x]$ on a logical formula is classical for Hoare logic (when using the weakest precondition approach). To be explicit, the notation $\overline{P_{\mathsf{v}}}[a/x]$ means $a \in P_{\mathsf{v}}(x) \wedge \bigwedge_{y \in \mathbf{Var} \setminus \{x\}} y \in P_{\mathsf{v}}(y)$. The second line of requirements ensures that the new policy P' records all the influencers of the variable assigned due to both implicit use of variables in X and explicit use of variables in a; additionally it ensures that the influence on the principal p is recorded in the new policy P'. The third line of requirements ensures that the new policy P' records all the readers of the variable assigned due to both implicit use of variables in X and explicit use of variables in a; additionally it ensures that the reading within p of X and a is recorded in the original policy P.

Once again note that the partial order for influencers is such that it is always secure to add influencers, that the partial order for readers is such that it is always secure to remove readers, and that the *semantic underpinning* of these statement is expressed by $\mathsf{sec}(P, F, P')$ (and Theorem 1 below).

The axiom schemes for output and input are easiest to understand if $ch!a$ is thought of as $ch := a$ and $ch?x$ is thought of as $x := ch$. The rule for input differs from assignment in that the pure Hoare logic component takes a strongest postcondition approach (as opposed to the weakest precondition approach used for assignment). The remaining axioms are rather standard from a Hoare logic point of view.

The type system is lifted to systems as follows:

$$\frac{\emptyset \vdash_{\mathsf{p}_1} \{\phi_1\} S_1 \{\mathsf{true}\} \quad \cdots \quad \emptyset \vdash_{\mathsf{p}_n} \{\phi_n\} S_n \{\mathsf{true}\}}{\vdash^{\mathcal{P}} \{\phi_1 \wedge \cdots \wedge \phi_n\} \mathsf{p}_1 : S_1 \mid\mid \cdots \mid\mid \mathsf{p}_n : S_n}$$

where we once more require that the free variables of the formula ϕ_i are contained in $\mathbf{Var}_{\mathsf{p}_i}$ (the variables belonging to the process p_i) thereby ensuring that ϕ_i only applies to S_i.

Example 4. *Returning to the motivating example we shall now highlight some of the steps in proving that the overall system guarantees the disjunctive security policy* $\mathcal{P} = \{\mathsf{P}^1, \mathsf{P}^2\}$. *To establish the judgement*

$$\emptyset \vdash_{\mathsf{m}} \{\mathsf{true}\}\ \mathsf{ch!}(1, x_1)\ \{\mathsf{true}\}$$

we will choose P' *to be* P^1 *independently of whether* P *is* P^1 *or* P^2. *To establish the judgement*

$$\emptyset \vdash_{\mathsf{d}} \{\mathsf{true}\}\ \mathsf{ch?}(y, z)\ \{\mathsf{true}\}$$

we will choose P' *to be equal to* P. *Finally, to establish the judgement*

$$\{y\} \vdash_{\mathsf{d}} \{y = 1\}\ \mathsf{out}_1!z\ \{y = 1\}$$

we use the precondition $y = 1$ *together with* $y \in P_{\mathsf{v}}(y)$ *to conclude that, even though we only seem to know that* $P \in \{\mathsf{P}^1, \mathsf{P}^2\}$ *it must be the case* $P = \mathsf{P}^1$, *and we can therefore choose* $P' = \mathsf{P}^1$ *and complete the proof.*

Correctness Results: Our overall correctness result shows that well-typed programs satisfy self-similarity:

Theorem 1. *If* $\vdash^{\mathcal{P}} \{\mathsf{true}\}Sys$ *then* $\models^{\mathcal{P}} \{\mathsf{true}\}Sys$.

The proof is by directly showing that typability is a self-simulation:

Proposition 1. $\vdash^{\mathcal{P}}$ *is a self-simulation with respect to* $\mathcal{P}$.

Among other things this establishes a *subject reduction* result (saying that typing is preserved under evaluation).

6 Conclusion and Future Work

We have extended basic *discretionary* information flow policies for readers (confidentiality) and influencers (integrity) to be dependent on *content* (values) and have introduced *disjunctive* information flow policies to facilitate the content-dependent shift between basic policies.

Our approach has been motivated by the challenges of the avionics gateway (as illustrated by the multiplexer example in Sect. 2) suggested by the avionics partners in the European Artemis Project SESAMO. Prior attempts at using DLM uncovered a number of weaknesses of information flow policies that are not able to incorporate content; however, the explicit use of security labels denoting readers and influencers were considered extremely relevant. This motivated our combination of Hoare logic assertions with classical security labels (unlike approaches like [1] that do not admit classical security labels) and our introduction of disjunctive policies. We are currently working on developing annotations for avionics software in C using dependent types and restricted logical formulae as an interface to the underlying disjunctive information flow policies.

We developed a *combined Hoare logic and type system* (Sect. 5) for verifying whether a system adheres to the specified disjunctive policies. Apart from the

technical convenience of using a Hoare logic as the basis of a type system it also facilitates incorporating the results of prior static analyses into the information flow type system; this is needed in order to interact with the approach of industrial users. To obtain a stronger type system it would be useful to strengthen the rule of consequence to admit analysis by cases

$$\frac{X \vdash_p \{\phi_1\}S\{\psi_1\} \qquad X \vdash_p \{\phi_2\}S\{\psi_2\}}{X \vdash_p \{\phi\}S\{\psi\} \quad \text{if } (\phi \Rightarrow \phi_1 \vee \phi_2) \wedge (\psi_1 \Rightarrow \psi) \wedge (\psi_2 \Rightarrow \psi)}$$

although we are not going to claim any (relative) completeness results for the combined Hoare logic and type system. For practical use one would need to limit the logical assertions to a restricted format so as to support efficient type inference.

The development has been performed for *concurrent systems* with synchronous communication and local memory as required by MILS [21] and ARINC-811 [10]. In addition to extensions with bypassing security policies it would be feasible to add polymorphism of annotations, add a principal hierarchy, incorporate procedures and methods, borrowing from DLM and other information flow policies.

Our semantic justification was based on an instrumented operational semantics in the manner used in static program analysis (Sect. 3). It provided a semantic interpretation that makes it clear that *opposite directions of flow* are appropriate for confidentiality and integrity, and hence agrees with the intuition about the *duality* of readers (always safe to remove some) and influencers (always safe to add some) in information flow type systems. Based on this we took a *co-inductive* approach to defining *self-similarity* (Sect. 4) borrowing from the development of bi-simulations. Our main correctness results showed that typability suffices for self-similarity (Sect. 5). Technically the proof amounted to showing that typability is itself a self-simulation.

This approach should not be seen as a dismissal of the value of a non-interference result (meaning that the system is contained in the reflexive part of a notion of bi-simulation). However, non-interference results are not easy to "get right" as is discussed at length in [7] and adding concurrency only adds to the complexities [14]: should non-interference be termination-sensitive, should it be timing-sensitive, etc. In particular, the approaches of [1,11] do not directly carry over because they do not deal with concurrent systems. While a non-interference result would be a welcome additional development, we would like to follow [7] in letting the non-interference result provide a stronger basis for the instrumented semantics rather than being the primary mechanism for ensuring the correctness of the type systems. This approach is in line with the research in programming languages where the vast majority of program analyses are formulated with respect to an understanding of program behaviour comparable to our instrumented semantics; looking for further justification is possible and considerations similar to those of non-interference are appropriate [19].

The exposition provided in this paper is a simplification of our formal development that has been checked using the Coq proof assistant (including the motivating multiplexer example of Sect. 2).

Acknowledgement. We are supported by IDEA4CPS (DNRF 86-10) and benefitted from discussions with Michael Paulitsch and Kevin Müller from Airbus.

References

1. Amtoft, T., Dodds, J., Zhang, Z., Appel, A., Beringer, L., Hatcliff, J., Ou, X., Cousino, A.: A certificate infrastructure for machine-checked proofs of conditional information flow. In: Degano, P., Guttman, J.D. (eds.) Principles of Security and Trust. LNCS, vol. 7215, pp. 369–389. Springer, Heidelberg (2012)
2. Andrews, G.R., Reitman, R.P.: An axiomatic approach to information flow in programs. ACM Trans. Program. Lang. Syst. **2**(1), 56–76 (1980)
3. Apt, K.R.: Ten years of Hoare's logic: A survey - part I. ACM Trans. Program. Lang. Syst. **3**(4), 431–483 (1981)
4. Apt, K.R.: Ten years of Hoare's logic: a survey part II: nondeterminism. Theoret. Comput. Sci. **28**, 83–109 (1984)
5. Bell, D.E., LaPadula, L.J.: Secure computer systems: a mathematical model. Technical report, MITRE Corporation (1973)
6. Biba, K.J.: Integrity considerations for secure computer systems. Technical report, MITRE Corporation (1977)
7. Boudol, G.: Secure information flow as a safety property. In: Guttman, J., Degano, P., Martinelli, F. (eds.) FAST 2008. LNCS, vol. 5491, pp. 20–34. Springer, Heidelberg (2009)
8. Broberg, N., Sands, D.: Paralocks: role-based information flow control and beyond. In: 37 th POPL, pp. 431–444. ACM (2010)
9. Chong, S., Myers, A.C.: Decentralized robustness. In: 19'th CSFW, pp. 242–256. IEEE Computer Society (2006)
10. Airlines Electronic Engineering Committee. ARINC 811: Commercial aircraft information security concepts of operation and process framework. Technical report (2005)
11. Darvas, Á., Hähnle, R., Sands, D.: A theorem proving approach to analysis of secure information flow. In: Hutter, D., Ullmann, M. (eds.) SPC 2005. LNCS, vol. 3450, pp. 193–209. Springer, Heidelberg (2005)
12. Denning, D.E., Denning, P.J.: Certification of programs for secure information flow. CACM **20**(7), 504–513 (1977)
13. Greve, D.: Data flow logic: Analyzing Information Flow Properties of C Programs. Rockwell Collins (2011)
14. Hedin, D., Sabelfeld, A.: A Perspective on Information-Flow Control. Marktoberdorf Summerschool (2011)
15. Montagu, B., Pierce, B.C., Pollack, R.: A theory of information-flow labels. In: 26th CSF, pp. 3–17. IEEE Computer Society (2013)
16. Müller, K., Paulitsch, M., Tverdyshev, S., Blasum, H.: MILS-related information flow control in the avionic domain: a view on security-enhancing software architectures. In: IEEE/IFIP International Conference on Dependable Systems and Networks Workshops, DSN 2012, pp. 1–6. IEEE (2012)

17. Myers, A.C., Liskov, B.: A decentralized model for information flow control. In: 16th ACM Symposium on Operating Systems Principles, pp. 129–142 (1997)
18. Myers, A.C., Liskov, B.: Protecting privacy using the decentralized label model. ACM Trans. Softw. Eng. Methodol. **9**(4), 410–442 (2000)
19. Nielson, F.: Program transformations in a denotational setting. ACM Trans. Program. Lang. Syst. **7**(3), 359–379 (1985)
20. Plotkin, G.D.: A structural approach to operational semantics. J. Logic Algebraic Program. **60–61**, 17–139 (2004)
21. Rushby, J.: Separation and Integration in MILS (The MILS Constitution). Technical report SRI-CSL-08-XX, SRI International, February 2008
22. Sabelfeld, A., Russo, A.: From dynamic to static and back: riding the roller coaster of information-flow control research. In: Virbitskaite, I., Voronkov, A., Pnueli, A. (eds.) PSI 2009. LNCS, vol. 5947, pp. 352–365. Springer, Heidelberg (2010)
23. Stirling, C.: A generalization of Owicki-Gries's Hoare logic for a concurrent while language. Theoret. Comput. Sci. **58**, 347–359 (1988)
24. Volpano, D.M., Irvine, C.E., Smith, G.: A sound type system for secure flow analysis. J. Comput. Secur. **4**(2/3), 167–188 (1996)
25. Whalen, M.W., Greve, D.A., Wagner, L.G.: Model checking information flow. In: Hardin, D.S. (ed.) Design and Verification of Microprocessor Systems for High-Assurance Applications, pp. 381–428. Springer, New York (2010)
26. Zheng, L., Myers, A.C.: End-to-end availability policies and noninterference. In: 18'th CSFW, pp. 272–286. IEEE Computer Society (2005)