# On Information Flow Control in Event-B and Refinement

Chunyan Mu
*LIAFA, Université Paris Diderot- Paris 7, France*
*Email: chunyan.mu@liafa.univ-paris-diderot.fr*

***Abstract*—This paper investigates the problem of preserving information flow security in Event-B specification models and during the process of refining an abstract specification to be more concrete. A typed Event-B model is presented to enforce information flow security. We then present an approach to the problem of preserving information flow properties under abstraction refinement. The novelty of the approach is that we formalise refinement transformation in terms of the mathematical concept of Galois connection for the purpose of information-flow analysis and control. That is, the state-invariant and state-transition predicates of the models are used to generate the Galois connection. We show how the refinement transformation ensures to preserve the security properties during the development steps from the beginning abstract-level specification to a concrete implementation.**

***Keywords*-Flow; Security; Specification; Event-B; Type System; Refinement**

## I. Introduction

Information flow analysis is a fundamental issue in preserving confidentiality and discovering security leaks in software systems. Information flow control analyses the programs and try to discover potential illegal flows and guarantee that no unauthorised flow can occur. In software development, it is important to concern security related issues from the very beginning, *e.g.* abstract-level specifications. Refinement is a relation on specifications that formalises the process of stepwise development to achieve a more concrete version. A security-concern stepwise development requires to preserve the security properties of interest during the development steps, until a concrete specification is achieved. It is therefore required to have a corresponding theory to ensure that the transformation from the abstract level to the more concrete level preserves the security properties of the abstract levels.

In this paper, we consider secure information flow analysis for systems in a specification language, Event-B. Event-B [1], [2] is a general purpose specification language used for the abstract design and detailed development of safety systems. We present security typing rules for a core Event-B model which can be used to control secure information flow. The model variables are typed into different security levels in an information flow lattice [3]. The typing rules for model events can prevent unauthorised information to flow in an abstract machine given a flow security policy. We focus on systems in which secrets are stored in model variables. Security labels are associated with variables indicating the intended secrecy of the contents. The security labels form a finite flow lattice $\mathcal{L}$ with a partial ordering $\sqsubseteq_{\mathtt{sec}}$. For example, the simplest case is that the security labels partition into two sets: high ($H$) denoting secret and low ($L$) denoting public. A partial ordering, $L \sqsubseteq_{\mathtt{sec}} H$, implies that the only permitted information flow is from $L$ to $H$, *i.e.,* no-flowing-down. We say a system is secure if it satisfies an applied security flow policy, for example, the well known non-interference property [4]. A system is *non-interfering* if it is not possible for an outside observer to gain any knowledge about the presence of $high$ labels in the original run (the observer only sees $low$ labels). The security problem is therefore to verify that there is no dependency between the initial value of the high labels and the final value of the low labels. Specifically, every reachable *state* of the system during the development steps should satisfy the basic non-interference property. In this paper, we are generally talking about a no-flowing-down security property denoted as $\phi_{\mathtt{sec}}$ rather than a specified property such as non-interference.

Furthermore, in Event-B, systems are developed using refinement: the system is first described as a simple abstract specification and then is refined stepwise to be a more concrete implementation. Such hierarchical specification approach may introduce information security violations during the stepwise development, even if each abstract-level specification of the system keeps sound information flow control. In order to avoid this, it is required to investigate how the transitions from the specification (higher level of abstraction) to the implementation (lower level of abstraction) of a system, *i.e.* refinements between abstraction levels, can be done in order to preserve the flow security properties. We introduce a novel method to formalise the refinement transformation between the abstract model and a relatively concrete one in terms of the mathematical concept of Galois connection. Such connection between the abstract models ensures that the stepwise refinement preserves the security property $\phi_{\mathtt{sec}}$.

The contributions of the paper are summarised as follows: we present typing rules for the core Event-B model with information flow control; we give semantic interpretation of transformations and propose a mechanism to preserve the security properties of interests under stepwise refinements by making Galois connections between abstract machines; we also study the security relationships between different levels of abstract machines during vertical *refinements*.

The rest of the paper is organised as follows. Section II presents the core Event-B model and the typing rules for the model with secure information flow control. In Section III, we investigate the problem of preserving security properties under refinement, and present some formal results w.r.t. the secure refinement. Section IV presents the related work in information flow control for specification and programming languages. The final Section summarises our work and presents directions for future research.

## II. The Event-B Model with Information Flow Control

In general, the development of a B model follows an incremental procedure validated by the refinement. A system is modelled by a sequence of models related by the refinement. In this paper, we use Event-B as our basis to investigate secure information flow analysis in specifications and refinement. Specifically, we define the security type environment as: $\Gamma : \text{VAR} \rightarrow \mathcal{L}$, where VAR is a set of model variables, and $\mathcal{L}$ is the security flow lattice. Intuitively, we assign a security type to each model variable. The set of the security types forms a security flow lattice denoted by $\mathcal{L}$. The actions of the model events can cause information flows from variables with a higher level to (observed) variables with a lower level. Clearly, such flows can cause information leakage and should not be allowed. We therefore define typing rules to the predicates and expression operators of the model to ensure such unauthorised flows are forbidden, and argue a basic predicate must satisfy its security typing rules.

### A. The Model and its Security Type System

An Event-B model normally consists of a set of *variables* constituting the state of the model, an *invariant* that describes the valid states, and a set of *events* that describes the transformation of the state of the model. The model can also use a *context machine* with *constant* and *set* definitions. We focus on a core structure of Event B in this paper.

**The model.** The core Event-B model is defined as a tuple $M = (m, \text{VAR}, I, \text{EVT})$ where $m$ is the *name* of the model, VAR is a sequence of distinct model *variables*, $I$ is a model *invariant* limiting the possible states of VAR, EVT is a set of model *events*.

An *event* is a *guarded command* and composed by a *guards* $G(\text{VAR})$ and a generalised substitution *action* $A(\text{VAR})$. We consider the events in the following form:

$$\text{EVT} \triangleq \texttt{when}\ G(\text{VAR})\ \texttt{then}\ A(\text{VAR})$$

where action denoted as $A$ are used to change the state of a machine. Intuitively, the *substitution* actions in the semantics of events can be viewed as a set of state-based predicate transformers. The effect of an action defines a *before-after predicate* which describes the relationship between the state before and after the action has occurred. A state is a map from VAR to their values. In an Event-B model, a state consists of a sequence of variables that are modified by events. The state space of an Event B model is therefore modelled as the Cartesian product of the state of the variables, *i.e.,* variables $v_1, \ldots, v_n$ having state $\sigma_1, \ldots, \sigma_n$ give the state space: $\Sigma = \sigma_1 \times \ldots \times \sigma_n$, where $\sigma_i \triangleq v_i \mapsto \mathbb{N}$. The substitution actions of the core Event-B model include:

$$\text{ACTION} ::= \texttt{skip} \;\Big|\; x := E \;\Big|\; x :\in S \;\Big|\; z : |P \;\Big|\; x, y := E, F$$

The event actions can be viewed as transformation functions which update the model state. Specifically,

- Null action: `skip` denotes the empty set of actions for an event, the state is unchanged under the *skip* action;
- Assignment: $x := E$ denotes the assignment, *i.e.,* state is updated by replacing free occurrences of $x$ by $E$: where $x \subseteq$ VAR is a sequence of variables, and $E$ denotes a number of set-theoretic expressions corresponding to each of the variables in $x$;
- Choice from the set: $x :\in S$ denotes that we update the state by arbitrarily choosing values from the set $S$ for the variables in $x$, *i.e.,* $x$ becomes a member of $S$;
- Choice by predicate: $x : |P$ denotes that we update the state by arbitrarily choosing values for the variables in $x$ that satisfy the predicate $P$, *i.e.,* $x$ becomes such that the predicate $P$ holds;
- Multiple action: $x, y := E, F$ denotes concurrent assignment of the values $E$ and $F$ to the variable sequences $x$ and $y$ respectively.

**Type checking for flow security.** At each abstract level, all entities are built from a set of events. Each event is defined in terms of substitution actions on model variables. Each variable is assigned a security level. The powerset of model variables therefore forms a complete lattice $\mathcal{L}$, where the partial ordering is regarding to the security levels of the model variables: $\forall v_1, v_2 \in \text{VAR},\ v_1 \preceq v_2$ *iff* $\tau_1 \sqsubseteq_{\texttt{sec}} \tau_2$, where $\preceq$ denotes the partial ordering on VAR, and $\tau_1, \tau_2 \in \mathcal{L}$ denote the security levels of variables $v_1$ and $v_2$ respectively. Event action causes information to flow among variables. Secure information flow can be described by a secure information flow predicate in a way of typing rules. Let $\mathcal{L}$ be the finite flow lattice. $\tau \subseteq \mathcal{L}$ denotes a sequence of security levels in $\mathcal{L}$ regarding to a sequence $x$ of variables with same length: $|\tau| = |x|$. Following the style of type system presented in [5], [6], we define the security typing environment as: $\Gamma : \text{VAR} \rightarrow \mathcal{L}$. In a model event, for an action $A$, judgements have the form: $\tau \vdash$, where the type $\tau$ denotes the counter level of the variable sequence $x$ regarding to the action $A(x)$ being executed to eliminate implicit flows from the *guard*, $\Gamma$ and $\Gamma'$ describe the security levels of the identifiers which hold before and after the execution of $A$. Furthermore, model events can influence each other and introduce information flows. It is also necessary

to look at the sequential ordering between model events, which might introduce data dependence between events and therefore introduce information flows as a result. Intuitively, the dependency between two events can be introduced in the following situation. The *before-after predicate* of an event $\text{EVT}_1$ can enable another event $\text{EVT}_2$ if the state of the model after executing $\text{EVT}_1$ makes the guard of $\text{EVT}_2$ be true and the action of $\text{EVT}_2$ will be executed in succession.

*Definition 1:* [Dependency between events.] Let $\text{EVT}_1$ and $\text{EVT}_2$ be two model events, and: $\text{EVT}_1 \triangleq \texttt{when}\ G_1(\text{VAR})\ \texttt{then}\ A_1(\text{VAR})$, $\text{EVT}_2 \triangleq \texttt{when}\ G_2(\text{VAR})\ \texttt{then}\ A_2(\text{VAR})$. For $i \in \{1, 2\}$, let $I$ denote the model invariant, $G_i$ denotes the guard of event $\text{EVT}_i$, $A_i$ denotes the action of event $\text{EVT}_i$, and $\Psi_i$ denotes the *before-after predicate* relating $\text{EVT}_i$. Say $\text{EVT}_2$ depends on $\text{EVT}_1$, written as: $(\text{EVT}_1 \rhd \text{EVT}_2)$, iff:

$$\begin{aligned} &\textit{if} \quad I(\text{VAR}) \wedge G_1(\text{VAR}) \quad \textit{then}: \\ &\quad [\![A_1]\!]\big(\Psi_1(\sigma(\text{VAR}), \sigma'(\text{VAR})\big) \Leftrightarrow \\ &\quad \big(G_2(\text{VAR}) = \texttt{true} \wedge [\![A_2]\!]\big(\Psi_2(\sigma(\text{VAR}), \sigma'(\text{VAR}))\big)\big). \end{aligned}$$

Predicate $\Psi_i(\sigma(\text{VAR}), \sigma'(\text{VAR}))$ means that the before and after state defined of $\text{EVT}_i$ are $\sigma$ and $\sigma'$ respectively. $[\![A]\!]\Psi$ means that executing action $A$ produces a predicate $\Psi$. We present the security typing rules for specifying security information flow predicates of events in Table I. Notation $\Gamma \vdash E : t$ denotes that under type environment $\Gamma$, expression $E$ has type $t$. The type of an expression (including guard expression) is defined by taking the least upper bound of the types of its free variables:

$$\Gamma \vdash E : t \ \ \textit{iff} \ \ t = \bigsqcup_{v \in \mathsf{fv}(E)} \Gamma(v)$$

The notation $\tau \sqcup \tau'$ is defined as follows.

*Definition 2:* For a sequence of variables denoted as $x = \langle v_1, \ldots, v_n \rangle$, $\tau = \langle t_1, \ldots, t_n \rangle$ defines a sequence of security levels of $x$, *i.e.,* $\tau(v_1) = t_1, \ldots, \tau(v_n) = t_n$ where $\tau(v_i)$ denotes the security level of variable $v_i$ for any $i \in \{1, \ldots, n\}$. Similarly, $\tau' = \langle t'_1, \ldots, t'_n \rangle$ defines a sequence security levels of $x$ as: $\tau'(v_1) = t'_1, \ldots, \tau'(v_n) = t'_n$. We therefore define $\tau \sqcup \tau'$ as: $\tau \sqcup \tau' = \langle t_1 \sqcup t'_1, \ldots t_n \sqcup t'_n \rangle$ regarding to the variable sequence: $x = \langle v_1, \ldots, v_n \rangle$.

*Definition 3:* We define that the basic predicate of EVT: $\Psi \triangleq \Gamma\{\text{EVT}\}\Gamma'$, satisfies the flow security properties, written as: $\Gamma\{\text{EVT}\}\Gamma' \vdash \phi_{\mathtt{sec}}$, *i.e.,* $\Psi \vdash \phi_{\mathtt{sec}}$ *iff* for all $\sigma'$, $\sigma$:

$$\forall v \in \text{VAR}. \Big(\Psi(\sigma(v), \sigma'(v)) \implies \textit{either}\ \Gamma'(v) \sqsubset_{\mathtt{sec}} \Gamma(v)\Big) \\ \textit{or}\ \sigma'(v) = \sigma(v)$$

where $\Psi(\sigma(v), \sigma'(v))$ denotes that the predicate by performing EVT will update the current state $\sigma(v)$ to be $\sigma'(v)$ regarding to a variable $v$.

*Definition 4:* For any $E_1 \rhd E_2 \rhd \ldots \rhd E_n$ ($n \in \mathbb{N}$), consider the relevant sequential of predicates $\langle \Psi_i(\sigma_{i-1}(\text{VAR}), \sigma_i(\text{VAR})) | i \in \{1 \ldots n\} \rangle$ regarding to $\langle E_i | i \in \{1 \ldots n\} \rangle$. We say a model $M$ is secure, written as: $M \vdash \phi_{\mathtt{sec}}$, *iff*, for each predicate $\Psi$ of $M$:

$$\Psi(\sigma_0(\text{VAR}), \sigma_n(\text{VAR})) \vdash \phi_{\mathtt{sec}} \ \text{ holds.}$$

*Theorem 1:* The security type system presented in Table I for the Event-B model is sound w.r.t. the security condition defined in Definition 3, 4.

*Proof:* We need to prove for any event EVT and their sequences with predicates $\Psi$ of the model, the following holds: $\forall v \in \text{VAR}, \Psi(\sigma(v), \sigma'(v)) \Rightarrow \Gamma'(v) \sqsubseteq_{\mathtt{sec}} \Gamma(v)$. $\Gamma'(v) \sqsubseteq_{\mathtt{sec}} \Gamma(v)$ implies that EVT contains no substitution to $v$ with a higher security level expression. Specifically, the rules ensures that for each semantic action, we take the least upper bound of the security settings before and after of the defined variables, *i.e.,* $\Gamma' \sqsupseteq_{\mathtt{sec}} \Gamma$. Together with the security condition, this ensure that for all $\Psi(\sigma'(v), \Psi(\sigma(v)))$, either $\Gamma'(v) \sqsubset_{\mathtt{sec}} \Gamma(v)$ or $\sigma'(v) = \sigma(v)$. ∎

*Example 1:* Let us look at a simple example. Assume we have two events defined as follows:

$$\begin{aligned} \text{EVT}_1 &\triangleq \texttt{when true then}\ l, h := 0, 2\ \texttt{end} \\ \text{EVT}_2 &\triangleq \texttt{when}\ l \leq h\ \texttt{then}\ l := l + 1\ \texttt{end} \end{aligned}$$

where model variables $\{l, h\} \subseteq \text{VAR}$, the security levels of $l, h$ are $\tau_1, \tau_2$ respectively, and $\tau_1 \sqsubseteq_{\mathtt{sec}} \tau_2$. Clearly, $\Psi_1(\sigma(l \mapsto 0, h \mapsto 0), \sigma'(l \mapsto 0, h \mapsto 2))$, and $\Psi_2(\sigma'(l \mapsto 0, h \mapsto 2), \sigma'(l \mapsto 1, h \mapsto 2))$. According to Definition 1, it is clear that $\text{EVT}_1$ can enable the execution of $\text{EVT}_2$, we have: $\text{EVT}_1 \rhd \text{EVT}_2$, the guard $(l \leq h)$ causes an implicit flow with a counted level $\tau_1 \sqcup \tau_2$, and therefore:

$$\frac{\Gamma\{\text{EVT}_1\}\Gamma'(l \mapsto \tau_1, h \mapsto \tau_2) \quad \Gamma'\{\text{EVT}_2\}\Gamma''(l \mapsto \tau_1 \sqcup \tau_2, h \mapsto \tau_2)}{\Gamma\{\text{EVT}_1 \rhd \text{EVT}_2\}\Gamma''(l \mapsto \tau_2, h \mapsto \tau_2)}$$

Note that $\Gamma''(l) \not\sqsubseteq_{\mathtt{sec}} \Gamma(l)$ implies that the system violates the flow property.

## III. Preserving Security Properties under Refinement

In general, an Event-B component concerns the development of software models and includes *abstract machine, refinement, concrete machine*. In this section, we investigate the problem of how to check the specification is consistent with the flow policy and the security properties are preserved under the refinement. Specifically, we are interested in the vertical refinement between a higher level abstract machine and a relatively concrete one.

As standard, we say a system specification model $M'$ is a refinement of a system specification model $M$ if and only if $M' \sqsubseteq_{\mathtt{ref}} M$. Intuitively, $M'$ is more accurate or less abstract than $M$. $M$ is secure if the events satisfy the security properties which are guarded by the typing rules

(TSub) $$\frac{\tau_1 \vdash \Gamma_1\{\textsc{Evt}\}\Gamma_1'}{\tau_2 \vdash \Gamma_2\{\textsc{Evt}\}\Gamma_2'} \quad \tau_2 \sqsubseteq_{\mathsf{sec}} \tau_1,\ \Gamma_2 \sqsubseteq_{\mathsf{sec}} \Gamma_1,\ \Gamma_1' \sqsubseteq_{\mathsf{sec}} \Gamma_2'$$

(TDepEvts) $$\frac{\Gamma_0\{\textsc{Evt}_1\}\Gamma_1 \quad \Gamma_1\{\textsc{Evt}_2\}\Gamma_2 \quad \ldots \quad \Gamma_{n-1}\{\textsc{Evt}_n\}\Gamma_n}{\Gamma\{\textsc{Evt}_1 \rhd \textsc{Evt}_2 \rhd \ldots \rhd \textsc{Evt}_n\}\Gamma_n}$$

(TSkip) $$\frac{}{\tau \vdash \Gamma\{\mathtt{skip}\}\Gamma}$$ (TAssign) $$\frac{\Gamma \vdash E : \tau'}{\tau \vdash \Gamma\{x := E\}\Gamma'(x \mapsto \tau \sqcup \tau')}$$

(TChoiceFromSet) $$\frac{t = \bigsqcup_{s \in S} \Gamma(s) \quad \tau' = \langle t, \ldots, t\rangle \wedge |\tau'| = |x|}{\tau \vdash \Gamma\{x :\in S\}\Gamma'(x \mapsto \tau \sqcup \tau')}$$

(TChoiceByPredicate) $$\frac{\Gamma \vdash P : \tau' = \bigsqcup_{v \in \mathtt{fv}(P)} \Gamma(v)}{\tau \vdash \Gamma\{x : |P|\}\Gamma'(x \mapsto \tau \sqcup \tau')}$$

(TMultipleAction) $$\frac{\Gamma \vdash E : \tau_1 \quad \Gamma \vdash F : \tau_2}{\tau \vdash \Gamma\{x, y := E, F\}\Gamma'(x \mapsto \tau \sqcup \tau_1, y \mapsto \tau \sqcup \tau_2)} \quad (x \cap y = \emptyset)$$

$$\frac{\Gamma \vdash E : \tau_1 \quad \Gamma \vdash F : \tau_2}{\begin{array}{l}\tau \vdash \quad \Gamma\{x, y := E, F\} \\ \qquad \Gamma'\Big(\{v_i\} \mapsto \tau(\{v_i\}) \sqcup \tau_1(\{v_i\}) \sqcup \tau_2(\{v_i\}), \\ \qquad\ x \setminus \{v_i\} \mapsto \tau(x \setminus \{v_i\}) \sqcup \tau_1(x \setminus \{v_i\}), \\ \qquad\ y \setminus \{v_i\} \mapsto \tau(y \setminus \{v_i\}) \sqcup \tau_2(y \setminus \{v_i\})\Big)\end{array}} \quad (x \cap y = \{v_i\})\ \textit{for}\ 1 \le i \le n$$

Table I
TYPING RULES FOR MODEL EVENTS WITH INFORMATION FLOW CONTROL.

defined in Table I. However, by the definition of refinement there is no guarantee that a refinement of $M$ will preserve the security properties since the refined model might introduce new or merge existing events. Therefore, secure information flow properties are not always preserved by refinement. One can imagine that the reason for this is that the secure flow properties and the relevant typing rules depends on the semantics of events and the type environment, which can not guarantee that the refinement transformation preserve the security properties of interest. One can argue that the security requirement can be viewed as security predicates, however, some of the events and their actions that satisfies the rules may be removed or merged during the refinement. Therefore, proving the security property at one abstract level is not enough in general. The lower level may introduce behaviours that do not have any abstract equivalent at a higher level. Relevant security properties must be proven again at the concrete level or ensured via the refinement transformation to guarantee that these additional behaviours introduced via refinement do not violate security flow policy. In this Section, we propose to study the problem of preserving security properties under refinement transformation.

### A. Secure refinement rules

In the first step, we propose to present the secure refinement rules of events in order to prove that a refinement is both secure and correct. Refinement provides a way to construct stronger invariants and add details in a way to enrich it step by step. This is generally achieved by extending the list of state variables, by merging and refining existing events, and by adding new events. Specifically, assume the lower level model denoted by $M_{\mathcal{C}}$ has a collection of state variables denoted by $v_{\mathcal{C}}$, which is distinct from the collection of variables denoted by $v_{\mathcal{A}}$ in the abstract model denoted by $M_{\mathcal{A}}$. The abstract variables $v_{\mathcal{A}}$ and the concrete ones $v_{\mathcal{C}}$ are linked together by means of *gluing invariant* $J(v_{\mathcal{A}}, v_{\mathcal{C}})$: it "glues" the state of the concrete model $M_{\mathcal{C}}$ to that of its abstraction $M_{\mathcal{A}}$. For any variable in abstract (concrete) machine if there is no glued variable in the relevant concrete (abstract) machine, we introduce a virtual variable denoted by $\perp$ to represent it, *i.e.,* we extend the set of the variables such that: $\textsc{Var}^*_{\mathcal{A}} = \textsc{Var}_{\mathcal{A}} \cup \{\perp\}$, and $\textsc{Var}^*_{\mathcal{C}} = \textsc{Var}_{\mathcal{C}} \cup \{\perp\}$. Fig. 1 shows the idea of it. Furthermore, a variant $V$ is introduced to prevent executions "non-terminating": $V$ has to decreased by every *convergent event* and must not be increased by *anticipated events*. Assume the type environments for abstract event and concrete event are denoted as $\Gamma_{\mathcal{A}}$ and $\Gamma_{\mathcal{C}}$ respectively. We extend the refinement laws to incorporate rules w.r.t. security properties in Table II. For clear cases, we use $\Psi(v, v')$ to denote the before and after states of variable(s) $v$ instead of writing $\Psi(\sigma(v), \sigma'(v))$.

- Rules (SubSec-REF) - (InvSec-REF) define the secure refinement for existing events. We assume abstract event $E_{\mathcal{A}}$ with guard $G_{\mathcal{A}}(v_{\mathcal{A}})$ and before-and-after predicate $\Psi_{\mathcal{A}}(v_{\mathcal{A}}, v'_{\mathcal{A}})$ in $M_{\mathcal{A}}$ is refining to a relevant concrete event $E_{\mathcal{C}}$ with guard $G_{\mathcal{C}}(v_{\mathcal{C}})$ and before-and-

| | |
|---|---|
| (SubSec-REF) | $I(v_\mathcal{A}) \ \wedge \ J(v_\mathcal{A}, v_\mathcal{C}) \ \wedge \ \Gamma_\mathcal{A} \vdash v_\mathcal{A} : \tau_\mathcal{A} \ \wedge \ \Gamma_\mathcal{C} \vdash v_\mathcal{C} : \tau_\mathcal{C} \implies \tau_\mathcal{C} \sqsubseteq_{\mathsf{sec}} \tau_\mathcal{A}$ |
| (FisSec-REF) | $I(v_\mathcal{A}) \ \wedge \ J(v_\mathcal{A}, v_\mathcal{C}) \ \wedge \ G_\mathcal{C}(v_\mathcal{C}) \ \wedge \ \Gamma_\mathcal{C} \vdash v_\mathcal{C} : \tau_\mathcal{C} \ \wedge \ \Gamma'_\mathcal{C} \vdash v'_\mathcal{C} : \tau'_\mathcal{C} \implies \exists v'_\mathcal{C}.\Psi_\mathcal{C}(v_\mathcal{C}, v'_\mathcal{C}) \ \wedge \ (\tau'_\mathcal{C} \sqsubseteq_{\mathsf{sec}} \tau_\mathcal{C})$ |
| (GrdSec-REF) | $I(v_\mathcal{A}) \ \wedge \ J(v_\mathcal{A}, v_\mathcal{C}) \ \wedge \ G_\mathcal{C}(v_\mathcal{C}) \ \wedge \ \Gamma_\mathcal{C} \vdash G_\mathcal{C}(v_\mathcal{C}) : \tau_\mathcal{C} \ \wedge \ \Gamma_\mathcal{A} \vdash G_\mathcal{A}(v_\mathcal{A}) : \tau_\mathcal{A} \implies G_\mathcal{A}(v_\mathcal{A}) \ \wedge \ (\tau_\mathcal{C} \sqsubseteq_{\mathsf{sec}} \tau_\mathcal{A})$ |
| (InvSec-REF) | $I(v_\mathcal{A}) \ \wedge \ J(v_\mathcal{A}, v_\mathcal{C}) \ \wedge \ G_\mathcal{C}(v_\mathcal{C}) \ \wedge \ \Psi_\mathcal{C}(v_\mathcal{C}, v'_\mathcal{C}) \ \wedge (\Gamma_\mathcal{C} \vdash v_\mathcal{C} : \tau_\mathcal{C}, \Gamma'_\mathcal{C} \vdash v'_\mathcal{C} : \tau'_\mathcal{C}) \ \wedge \ (\Gamma_\mathcal{A} \vdash v_\mathcal{A} : \tau_\mathcal{A}, \Gamma'_\mathcal{A} \vdash v'_\mathcal{A} : \tau'_\mathcal{A})$ $\implies \exists v'_\mathcal{A}.(\Psi_\mathcal{A}(v_\mathcal{A}, v'_\mathcal{A}) \wedge J(v'_\mathcal{A}, v'_\mathcal{C})) \ \wedge \ (\tau'_\mathcal{C} \sqsubseteq_{\mathsf{sec}} \tau_\mathcal{C} \sqsubseteq_{\mathsf{sec}} \tau_\mathcal{A}) \ \wedge \ (\tau'_\mathcal{C} \sqsubseteq_{\mathsf{sec}} \tau'_\mathcal{A} \sqsubseteq_{\mathsf{sec}} \tau_\mathcal{A})$ |
| (MergeSec-REF) | $E \triangleq \texttt{when}\ G(v)\ \texttt{then}\ S(v)\ \texttt{end} \ \wedge \ F \triangleq \texttt{when}\ H(v)\ \texttt{then}\ S(v)\ \texttt{end} \ \wedge \ (\tau \vdash \Gamma\{E\}\Gamma') \ \wedge \ (\tau \vdash \Gamma\{F\}\Gamma'')$ $\implies EF \triangleq \texttt{when}\ G(v) \vee H(v)\ \texttt{then}\ S(v)\ \texttt{end} \ \wedge \ \tau \vdash \Gamma \ \{EF\} \ \Gamma' \sqcup \Gamma'' \ \wedge \Gamma' \sqcup \Gamma'' \sqsubseteq_{\mathsf{sec}} \Gamma$ |
| (NewEvtSec-REF) | $I(v_\mathcal{A}) \ \wedge \ J(v_\mathcal{A}, v_\mathcal{C}) \ \wedge \ G_\mathcal{C}(v_\mathcal{C}) \ \wedge \ \Psi_\mathcal{C}(v_\mathcal{C}, v'_\mathcal{C}) \ \wedge \ (\Gamma_\mathcal{C} \vdash v_\mathcal{C} : \tau_\mathcal{C} \ \wedge \ \Gamma'_\mathcal{C} \vdash v'_\mathcal{C} : \tau'_\mathcal{C})$ $\implies J(v_\mathcal{A}, v'_\mathcal{C}) \ \wedge \ (V(v'_\mathcal{C}) \in \mathbb{N} \wedge (V(v'_\mathcal{C}) < V(v_\mathcal{C}))) \ \wedge \ \tau'_\mathcal{C} \sqsubseteq_{\mathsf{sec}} \tau_\mathcal{C}$ |

Table II
SECURE REFINEMENT RULES FOR EXISTING EVENTS.

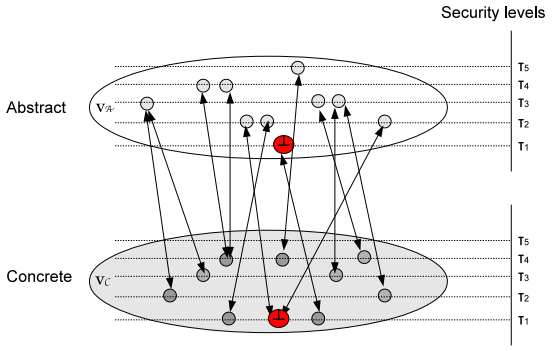


Figure 1. Gluing Invariant

after predicate $\Psi_\mathcal{C}(v_\mathcal{C}, v'_\mathcal{C})$ in $M_\mathcal{C}$. Let $\tau_\mathcal{A} \in \mathcal{L}_\mathcal{A}$ be a sequence of security levels of the sequence of variables $v_\mathcal{A}$ and $\tau_\mathcal{C} \in \mathcal{L}_\mathcal{C}$ be a sequence of security levels of the sequence of variables $v_\mathcal{C}$. we say $\tau_\mathcal{C} \sqsubseteq_{\mathsf{sec}} \tau_\mathcal{A}$, *iff*:

$$\forall v_a \in v_\mathcal{A}, \ v_c \in v_\mathcal{C}, \ J(v_a, v_c) \in J(v_\mathcal{A}, v_\mathcal{C}) \implies \tau_c \sqsubseteq_{\mathsf{sec}} \tau_a.$$

Specifically, SubSec-REF ensures that for glued variables, the security level of variables in the concrete event will not be higher than that of their glued variables in the abstract event. FisSec-REF ensures that the refined event is feasible and there is no unauthorised flow introduced by the before-after predicate. GrdSec-REF and InvSec-REF ensure that the concrete event is a correctly refined event regarding to the abstract one and the security level of the corresponding variables will not be higher than that of the glued ones in the abstract event. Note that for any variable $v$, the security level of an "empty" variable $\bot$ is never higher than its "glued" variable $v$, *i.e.*, $\tau(\bot) \sqsubseteq_{\mathsf{sec}} \tau(v)$.

- Rule (MergeSec-REF) specifies the secure refinement for merging existing events. Several abstract events can be merged being refined to a single concrete event. For event $E$ and $F$, $EF$ defines the refinement of merging $E$ and $F$, and the security typing rule ensures the security environment of $EF$ (after the refinement of merging $E$ and $F$) be the least upper bound of that of $E$ and $F$. The notation $\Gamma' \sqcup \Gamma''$ is defined as:

$$\forall v \in \textsc{Var}, \ (\Gamma' \vdash v : \tau') \wedge (\Gamma'' \vdash v : \tau'') \Rightarrow \ (\Gamma' \sqcup \Gamma'' \vdash v : \tau' \sqcup \tau'')$$

- New events might be introduced in a refinement. The rule (NewEvSec-REF) ensures that adding new event during the refinement will not introduce secure information flow and be correct. As usual, each new event refines an implicit `skip` event which satisfies the security typing rules, and the non-divergence rule.

### B. Secure Refinements as Galois Connections

In the second step, we propose to introduce a method to define *secure transformation function* between abstract models under refinement. To investigate the common refinement relation that can be obtained between an abstract model and a relevant concrete model, we explore the idea of making Galois connection between them. First let us review the basic definition of Galois connection [7].

*Definition 5:* Let $\mathcal{P} = (P, \preceq_P)$ and $\mathcal{Q} = (Q, \preceq_Q)$ be posets, and suppose $f_* : P \to Q$ and $f^* : Q \to P$ are a pair of functions such that $\forall p \in P$ and $\forall q \in Q$,

$$f_*(p) \preceq_Q q \ \textit{iff} \ p \preceq_P f^*(q)$$

then the pair $(f_*, f^*)$ form a *Galois connection* between $\mathcal{P}$ and $\mathcal{Q}$. If $(f_*, f^*)$ is such a connection, $f_*$ is said to be the *left adjoint* of the corresponding $f^*$, and $f^*$ is the *right adjoint* of $f_*$.

Definition 6 presents a relationship between the abstract model and a relevant refined model regarding to the predicates and security properties.

*Definition 6:* For a given Event-B model $M$, let $\varphi$ be a predicate w.r.t. a variable $v$ of $M$, $\Psi$ be the set of predicates w.r.t. the set of the model variables VAR of $M$, $M'$ be a refined model of $M$: $M' \sqsubseteq_{\mathtt{ref}} M$, $\varphi'$ be a predicate w.r.t. variable $v'$ of $M$ which is gluing to $v$: $J(v, v')$, $\Psi'$ be the set of predicates of the set of the model variables VAR$'$ of $M'$. Then:

- if $\varphi \vdash \phi_{\mathtt{sec}} \Rightarrow \varphi' \vdash \phi_{\mathtt{sec}}$, then we write $\varphi' \models \varphi$;
- we say $\Psi' \models \Psi$ *iff*

$$\forall (v \in \text{VAR} \wedge v' \in \text{VAR}' \wedge J(v, v')) : \ \varphi' \models \varphi.$$

**The Galois Connection.** We propose to make a Galois connection between an abstract model $M_{\mathcal{A}}$ and a relevant refined concrete model $M_{\mathcal{C}}$. For cases without introducing confusion, we say the Galois connection between the models $M_{\mathcal{A}}$ and $M_{\mathcal{C}}$ rather than between their sate spaces of variables. Specifically, the refinement between $M_{\mathcal{A}}$ and $M_{\mathcal{C}}$ can be viewed as a paired function: $M_{\mathcal{A}}(f^*, f_*)M_{\mathcal{C}}$, where $f^*$ denotes a concretisation function from $M_{\mathcal{A}}$ to $M_{\mathcal{C}}$ and defines the interpretation from $M_{\mathcal{A}}$ to lower level $M_{\mathcal{C}}$; while $f_*$ denotes the abstraction function from $M_{\mathcal{C}}$ to $M_{\mathcal{A}}$, and defines the interpretation from $M_{\mathcal{C}}$ to $M_{\mathcal{A}}$. Remember that each model is a state transformer via transformation functions. Let $[\![\cdot]\!]_{\mathcal{A}}$ denote the semantic substitution function under abstract machine $M_{\mathcal{A}}$, and $[\![\cdot]\!]_{\mathcal{C}}$ denote the semantic substitution function under concrete machine $M_{\mathcal{C}}$. Diagrammatically, Fig. 2 shows an intuitive connection between $M_{\mathcal{A}}$ and $M_{\mathcal{C}}$.

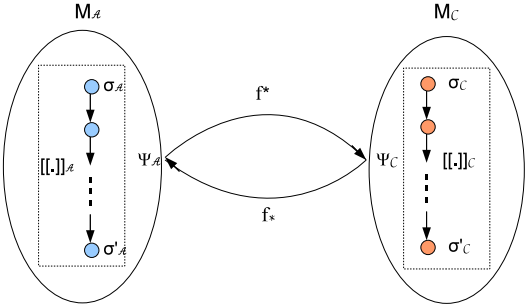


Figure 2. Connection between abstract models under refinement

Formally, we present the definition of the connection as follows.

*Definition 7:* Let $M_{\mathcal{A}}$ be a higher level model, $M_{\mathcal{C}}$ be a lower level model, $\Psi_{\mathcal{A}}$ be the set of predicates of $M_{\mathcal{A}}$, $\Psi_{\mathcal{C}}$ be the set of predicates of $M_{\mathcal{C}}$, and notation $\mathcal{P}(X)$ denotes powerset of set $X$ as standard. Then put:

- the powerset of the predicates set of $M_{\mathcal{A}}$ forms a lattice with a partial ordering on subset relations $\subseteq_{\mathcal{A}}$, we write: $M_{\mathcal{A}} = (\mathcal{P}(\Psi_{\mathcal{A}}), \subseteq_{\mathcal{A}})$, where $\subseteq_{\mathcal{A}}$ denotes the *subset* relation on $\mathcal{P}(\Psi_{\mathcal{A}})$;
- similarly, we write: $M_{\mathcal{C}} = (\mathcal{P}(\Psi_{\mathcal{C}}), \subseteq_{\mathcal{C}})$, where $\subseteq_{\mathcal{C}}$ denotes the *subset* relation on $\mathcal{P}(\Psi_{\mathcal{C}})$;
- for $\alpha \subseteq \Psi_{\mathcal{A}}$, define:

$$f^*(\alpha) = \{\varphi_c \mid \forall \varphi_a (\varphi_a \in \alpha \rightarrow \varphi_c \models \varphi_a)\};$$

- for $\beta \subseteq \Psi_{\mathcal{C}}$, define:

$$f_*(\beta) = \{\varphi_a \mid \forall \varphi_c (\varphi_c \in \beta \rightarrow \varphi_c \models \varphi_a)\}.$$

Paired function $(f^*, f_*)$ thus build a connection between an abstract model $M_{\mathcal{A}}$ and a refined model $M_{\mathcal{C}}$.

*Theorem 2:* Refinement function $(f^*, f_*)$ defined in Definition 7 forms a Galois connection between $M_{\mathcal{A}}$ and $M_{\mathcal{C}}$.

*Proof:* According to the definition of Galois connection, to prove $(f^*, f_*)$ forming a Galois connection is equivalent to prove the following equivalence:

$$\forall \beta \subseteq \Psi_{\mathcal{C}}, \alpha \subseteq \Psi_{\mathcal{A}} : \ f_*(\beta) \subseteq_{\mathcal{A}} \alpha \ \textit{iff} \ \beta \subseteq_{\mathcal{C}} f^*(\alpha).$$

Let $\equiv$ denote "equivalent to", we have:

$$\begin{aligned} \beta \subseteq_{\mathcal{C}} f^*(\alpha) &\equiv \{\varphi_c \mid \forall \varphi_a (\varphi_a \in \alpha \rightarrow \varphi_c \models \varphi_a)\} \supseteq_{\mathcal{C}} \beta \\ &\equiv (\forall \varphi_c \in \beta)(\forall \varphi_a \in \alpha)\varphi_c \models \varphi_a \\ &\equiv (\forall \varphi_a \in \alpha)(\forall \varphi_c \in \beta)\varphi_c \models \varphi_a \\ &\equiv \{\varphi_a \mid \forall \varphi_c (\varphi_c \in \beta \rightarrow \varphi_c \models \varphi_a)\} \subseteq_{\mathcal{A}} \alpha \\ &\equiv f_*(\beta) \subseteq_{\mathcal{A}} \alpha \end{aligned}$$

Therefore, the refinement connection $(f^*, f_*)$ between $M_{\mathcal{A}}$ and $M_{\mathcal{C}}$ forms a Galois connection.

■

Galois connections has important properties which relate them to the underlying ordered structures. The following theorem shows that any three abstract models under refinement built via two Galois connections with matching pre-orders can be composed.

*Theorem 3:* Let us consider the following abstract models under refinement: $M_1 = (\mathcal{P}(\Psi_1), \subseteq_1)$, $M_2 = (\mathcal{P}(\Psi_2), \subseteq_2)$, and $M_3 = (\mathcal{P}(\Psi_3), \subseteq_3)$, and: $M_3 \sqsubseteq_{\mathtt{ref}} M_2 \sqsubseteq_{\mathtt{ref}} M_1$. Let $(f^*, f_*)$ be the Galois connection between $M_1$ and $M_2$, $(g^*, g_*)$ be the Galois connection between $M_2$ and $M_3$. Then $(f^* \circ g^*, g_* \circ f_*)$ is a Galois connection between $M_1$ and $M_3$.

*Proof:* By assumption and Definition 5, for $\alpha \subseteq \Psi_1$, $\beta \subseteq \Psi_2$, and $\gamma \subseteq \Psi_3$, we have:

$$G_1 : f_*(\beta) \subseteq_1 \alpha \ \ \textit{iff} \ \ \beta \subseteq_2 f^*(\alpha),$$

and

$$G_2 : g_*(\gamma) \subseteq_2 \beta \ \ \textit{iff} \ \ \gamma \subseteq_3 g^*(\beta).$$

By $G_2$, we have:

$$g^*(\gamma) \subseteq_2 \beta \subseteq_2 f^*(\alpha) \ \ \textit{iff} \ \ \gamma \subseteq_3 g^*(\beta) \subseteq_3 g^*(f^*(\alpha)) \quad (1)$$

By $G_1$, we have:

$$f_*(g_*(\gamma)) \subseteq_1 f_*(\beta) \subseteq_1 \alpha \ \ \textit{iff} \ \ g_*(\gamma) \subseteq_2 \beta \subseteq_2 f^*(\alpha) \quad (2)$$

By (1) and (2), we obtain what we want:

$$f_*(g_*(\gamma)) \subseteq_1 \alpha \ \ \textit{iff} \ \ \gamma \subseteq_3 g^*(f^*(\alpha))$$

■

*Theorem 4:* Let $M_{\mathcal{C}}$ and $M_{\mathcal{A}}$ be two abstract models, and $M_{\mathcal{C}} \sqsubseteq_{\mathtt{ref}} M_{\mathcal{A}}$, then: $M_{\mathcal{A}} \vdash \phi_{\mathtt{sec}} \implies M_{\mathcal{C}} \vdash \phi_{\mathtt{sec}}$.

*Proof:* By Definition 4, $M_{\mathcal{A}} \vdash \phi_{\mathtt{sec}}$ implies for any predicate set $\Psi_{\mathcal{A}}$ of $M_{\mathcal{A}}$: $\Psi_{\mathcal{A}} \vdash \phi_{\mathtt{sec}}$.

By Definition 3, we have:

$$\forall(v_a \in \textsc{Var}_{\mathcal{A}}, v_c \in \textsc{Var}_{\mathcal{C}}, J(v_a, v_c)).\varphi_c \models \varphi_a \qquad (3)$$

where $\varphi_c \in \Psi_{\mathcal{C}}$ denotes the predicate w.r.t. $v_c$, and $\varphi_a \in \Psi_{\mathcal{A}}$ denotes the predicate w.r.t. $v_a$.

On the other hand, for $\alpha \subseteq \Psi_{\mathcal{A}}$, according to Definition 7:

$$f^*(\alpha) = \{\varphi_c \mid \forall\varphi_a(\varphi_a \in \alpha \rightarrow \varphi_c \models \varphi_a)\} \qquad (4)$$

By $(3), (4)$, we have:

$$\forall\varphi_a \in \Psi_{\mathcal{A}}.f^*(\varphi_a) \vdash \phi_{\mathtt{sec}},\ \textit{i.e.,}\ \forall\varphi_c \in \Psi_{\mathcal{C}}.\varphi_c \vdash \phi_{\mathtt{sec}}.$$

This implies, for any predicate set $\Psi_{\mathcal{C}}$ of $M_{\mathcal{C}}$, we have: $\Psi_{\mathcal{C}} \vdash \phi_{\mathtt{sec}}$. By Definition 4, we then obtain: $M_{\mathcal{C}} \vdash \phi_{\mathtt{sec}}$. ■

Theorem 4 implies that the refinement transformation preserve the security properties of interests, *i.e.,* if the abstract machine $M_{\mathcal{A}}$ is secure and $M_{\mathcal{A}}(f^*, f_*)M_{\mathcal{C}}$, then the relevant $M_{\mathcal{C}}$ is secure as well.

*Corollary 1:* Assume $M_n \sqsubseteq_{\mathtt{ref}} \ldots \sqsubseteq_{\mathtt{ref}} M_2 \sqsubseteq_{\mathtt{ref}} M_1$, $n \in \mathbb{N}$, then: $M_1 \vdash \phi_{\mathtt{sec}} \implies M_n \vdash \phi_{\mathtt{sec}}$.

*Proof:* The proof follows directly from Theorem 3 and 4 by induction. ■

## IV. Related Work

This paper relates to the topic of security information flow control in specification languages. The notion of secure information flow specifies the security requirements of the system where should be no information flow from the datum to the observer. Information flow control in programming languages for security concerns has been an active research topic for a long time. Denning and Denning [8] first use program analysis to investigate if the information flow properties of a program satisfy a specified multi-level security policy. Security type systems had been substantially used to formulate the analysis of secure information flow in programs. Specifically, our work is initially inspired by the flow sensitive type system for programs in a simple While language for multi-level security by Hunt and Sands [6]. In [6], sensitive information was stored in programming variables, the powerset of program variables thus forms the universal lattice. The flow-sensitive types system was defined by a family of inference systems which is forced to satisfy a simple non-interference property. Their recent work [9] showed how flow-sensitive multi-level security typing can be achieved in polynomial time. In addition to type-based treatments of secure information flow analysis for programs, Clark *et. al* presented a flow logic approach in [10]. Hammer and Snelting [11] presented an approach for information flow control in program analysis based on program dependence graphs (PDG). Based on [11], [12] extended the PDG-based flow analysis by incorporating refinement techniques via path conditions to improve the precision of the flow control. Such PDG-based information flow control is more precise but more expensive than type-based approaches. These works did not include treatments on specification language and secure abstraction refinements.

A number of papers addresses flow analysis in Event-B. Iliasov [13] introduced a method for control flow properties in Event-B models. The *flow* analysis in this paper focused on expressions with event ordering and looked at the interference between events introduced by a set of conditions formulated on a machine. It can be used to express flow properties for a model and to verify them using proofs. Bendisposto *et. al* [14] proposed an automatic flow analysis by deriving a flow graph structure from an Event-B model specification. The derived graph contained information about dependence and independence of events which can be used for flow analysis and model comprehension.

In the context of refinement, Jocob [15] first pointed out that secure information flow properties were not preserved by the standard notion of refinement in general. There are a number of papers addressed information flow security and refinement. Heisel *et. al* developed a *condition* for confidentiality-preserving renement in [16], [17]. The basic idea was that the information allowed to be revealed by the concrete system should also be allowed to be revealed regarding to the abstract one. Alur *et. al* [18] presented a framework for preservation of secrecy in *labelled transition system*, and introduced a simulation-based proof technique for preserving secrecy under refinement. Mantel [19] proposed a method for preserving information flow properties under CSP-style renements regarding to an *event system*. The event system was considered as a tuple of a set of input/output events and a set of traces. The idea here was two fold: introducing refinement operators to refine specifications and then constructing secure refined event system based on low equivalence relations. Mantel showed how tailored renement operators for information flow properties can construct a renement in which the resulting renement preserves the given flow property. Bossi *et. al* [20] studied the problem of preservation of information flow properties under *action refinement* in the context of *process algebra*. Seehusen and Stølen [21] introduced a schema to specify and preserve secure information flow properties in the semantics of STAIRS [22]. Jürjens [23] presented a framework for preserving secrecy under refinement operators in specification framework FOCUS [24]. In FOCUS, a process was modelled by a total stream-processing function which mapped input streams to sets of output streams. A process was considered preserving the secrecy if without eventually outputting secrets. [23] presented a set of *conditions* w.r.t. FOCUS under which the refinements preserved proposed secrecy properties. However, these works did not propose techniques to define the program and refinement transformation which

ensures the preservation of secure flow properties as involved in our paper. Comparing with these works, we have introduced a sound and more general approach for secure flow control in a *specification language* and for preservation of secure flow properties under stepwise refinement by building transformation and connection functions. We have presented a framework to reason about security properties and relevant relations of the stepwise refinement transformations.

## V. Conclusions and future works

This paper investigates the problem of preserving information flow security in Event-B specification models and in the process of refinement. We have introduced a framework for reasoning about the secure flow property in Event-B and under refinement. We present a security type system for a core Event-B model with secure information flow control. We then introduce a novel mechanism to preserve secure information flow properties under abstraction refinement. We make Galois connections between the abstract model and the refined concrete ones for the purpose of flow analysis and control. The Galois connection is generated by the converse of the relations w.r.t. the predicates of the models. The refinement function via Galois connection ensures that the refinements preserve the security properties during the development steps from the beginning abstract-level specification to a concrete implementation. We also study a set of security relationships between different levels of abstract machines during the vertical *refinements*.

We have presented a general framework for formal reasoning about preserving flow secrecy properties in a specification language and the process of abstraction refinements. We believe that the results presented in this paper is a promising starting point for a comprehensive formal treatment of security issues in specification languages and standard stepwise refinements. For future work, we plan to investigate a logic for the flow security properties and explore the relationships between them. Another possibility is to define program transformations in a way that guarantees the preservation of flow security based on our framework.

## References


[1] J. Abrial and L. Mussat, "Introducing dynamic constraints in b," in *B*, 1998, pp. 83–128.

[2] J. Abrial, M. Butler, S. Hallerstede, T. S. Hoang, F. Mehta, and L. Voisin, "Rodin: an open toolset for modelling and reasoning in Event-B," *STTT*, vol. 12, no. 6, pp. 447–466, 2010.

[3] J. Landauer and T. Redmond, "A lattice of information," in *CSFW*, 1993, pp. 65–70.

[4] J. Goguen and J. Meseguer, "Security policies and security models," in *S & P*, 1982, pp. 11–20.

[5] R. P. Reitman and G. R. Andrews, "Certifying information flow properties of programs: An axiomatic approach," in *POPL*, 1979, pp. 283–290.

[6] S. Hunt and D. Sands, "On flow-sensitive security types," in *POPL*. ACM Press, January 2006, pp. 79–90.

[7] M. Erne, J. Koslowski, A. Melton, and G. Strecker, "A primer on galois connections," in *York Academy of Science*, 1992.

[8] D. E. Denning and P. J. Denning, "Certification of programs for secure information flow," *Commun. ACM*, vol. 20, no. 7, pp. 504–513, 1977.

[9] S. Hunt and D. Sands, "From exponential to polynomial-time security typing via principal types," in *ESOP*, 2011, pp. 297–316.

[10] D. Clark, C. Hankin, and S. Hunt, "Information flow for algol-like languages," *Comput. Lang.*, vol. 28, no. 1, pp. 3–28, 2002.

[11] C. Hammer and G. Snelting, "Flow-sensitive, context-sensitive, and object-sensitive information flow control based on program dependence graphs," *Int. J. Inf. Sec.*, vol. 8, no. 6, pp. 399–422, 2009.

[12] M. Taghdiri, G. Snelting, and C. Sinz, "Information flow analysis via path condition refinement," in *FAST*, 2010, pp. 65–79.

[13] A. Iliasov, "On event-b and control flow," School of Computing Science, Newcastle University, Tech. Rep. CS-TR-1159, 2009.

[14] J. Bendisposto and M. Leuschel, "Automatic flow analysis for event-b," in *FASE*, 2011, pp. 50–64.

[15] J. Jacob, "On the derivation of secure components," in *S & P*, 1989, pp. 242–247.

[16] M. Heisel, A. Pfitzmann, and T. Santen, "Confidentiality-preserving refinement," in *CSFW*, 2001, pp. 295–306.

[17] T. Santen, M. Heisel, and A. Pfitzmann, "Confidentiality-preserving refinement is compositional - sometimes," in *ESORICS*, 2002, pp. 194–211.

[18] R. Alur, P. Cerný, and S. Zdancewic, "Preserving secrecy under refinement," in *ICALP (2)*, 2006, pp. 107–118.

[19] H. Mantel, "Preserving information flow properties under refinement," in *S & P*, 2001, pp. 78–92.

[20] A. Bossi, C. Piazza, and S. Rossi, "Action refinement in process algebra and security issues," in *LOPSTR*, 2007, pp. 201–217.

[21] F. Seehusen and K. Stølen, "Maintaining information flow security under refinement and transformation," in *FAST*, 2006, pp. 143–157.

[22] Ø. Haugen and K. Stølen, "Stairs - steps to analyze interactions with refinement semantics," in *UML*, 2003, pp. 388–402.

[23] J. Jürjens, "Secrecy-preserving refinement," in *FME*, 2001, pp. 135–152.

[24] M. Broy and K. Stølen, *Specification and development of interactive systems: focus on streams, interfaces, and refinement*. Springer-Verlag New York, Inc., 2001.