Specifying Confidentiality in *Circus*

Michael J. Banks and Jeremy L. Jacob

Department of Computer Science, University of York, UK
{Michael.Banks,Jeremy.Jacob}@cs.york.ac.uk

Abstract. This paper presents an approach for extending the *Circus* formalism to accommodate information flow security concerns. Working with the semantics of *Circus*, we introduce a notation for specifying which aspects of *Circus* processes are confidential and should not be revealed to low-level users. We also describe a novel procedure for verifying that a process satisfies its confidentiality properties.
Keywords: *Circus*, information flow security, confidentiality properties, unifying theories of programming, verifying security

1 Introduction

How can software engineers obtain robust assurances that a system does not leak secret data to its users (and other entities) who lack an appropriate security clearance? We say that information flows from a system to a user if that user can analyse its interactions with the system to deduce details about the system's behaviour. The study of techniques for measuring and regulating information flow is a central topic in theoretical studies of computer security [1,2].

When building a system that handles secret data — such as cryptographic keys or classified documents — it is vital to ensure the system's design does not induce undesirable information flows about that data to low-level (untrusted or unprivileged) users. A *confidentiality property* prescribes an upper bound on information flow from a system to a low-level user, to prevent that user from deducing secret information from its interactions with the system. These properties are inherently non-functional, and so they cannot be specified using the facilities provided by conventional formal methods.

The rationale for combining functionality and confidentiality requirements within a formal framework is to simplify the task of building systems that are "secure by construction". Without systematic support for modelling confidentiality properties alongside functional specifications, we can argue that a system satisfies a given confidentiality property only by resorting to *ad hoc* reasoning. In this paper, we outline how the *Circus* formalism [3,4,5] can be extended with facilities for specifying the confidentiality properties that a system should satisfy, in addition to a specification of the system's desired behaviour.

The contributions of this paper are as follows. First, in Section 3, we present a syntax for specifying confidentiality properties over *Circus* processes and define its semantics in Section 4. Second, in Section 5, we describe a method for verifying that a *Circus* process satisfies the confidentiality properties encoded in its specification. Third, in Section 6, we identify how *Circus* processes can be refined while preserving confidentiality properties. In Section 7, we compare our approach with existing frameworks for integrating confidentiality properties into formal software development. We summarise our work in Section 8.

2 *Circus*

Circus is a formal specification language which integrates the CSP process algebra with the state-based specification facilities of Z to achieve a cohesive framework for modelling state-rich concurrent and reactive systems.

A *Circus* process specifies an internal (private) state, a state invariant and a collection of named *actions*, which can be grouped into Z schema expressions and guarded commands (representing operations on the state), invocations of other actions (by name) and CSP constructs (modelling interaction with the environment). The behaviour of a *Circus* process is defined by a distinguished nameless main action, which follows the declarations of the other actions.

Example 1. The following *Circus* process, *Cell*, represents a memory cell that stores an integer value from the *hin* channel and broadcasts it on the *hout* channel. The cell can be switched between two modes. In public mode, the value currently stored in the cell is also broadcast on the *lout* channel; whereas in private mode, an arbitrary value is broadcast on *lout*.

$$
\begin{array}{l}
MODE == \{PUB, PRV\} \\
\textbf{channel } on, off \\
\textbf{channel } hin, hout : \mathbb{N} \\
\textbf{channel } lout : MODE \times \mathbb{N} \\
\textbf{process } Cell \mathrel{\widehat=} \textbf{begin} \\
\quad \textbf{state } Mem \mathrel{\widehat=} [\, val : \mathbb{N}, m : MODE \,] \\
\quad Init \mathrel{\widehat=} [\, Mem' \mid val' = 0 \land m' = PUB \,] \\
\quad Read \mathrel{\widehat=} \left(\begin{array}{l} \quad m = PUB \ \& \ lout!(PUB, val) \rightarrow Skip \\ \Box\ m = PRV \ \& \ \sqcap_{n \in \mathbb{N}} lout!(PRV, n) \rightarrow Skip \\ \Box\ hout!val \rightarrow Skip \end{array} \right) \\
\quad Write \mathrel{\widehat=} hin?n \rightarrow val := n? \\
\quad Switch \mathrel{\widehat=} (on \rightarrow m := PRV) \Box (off \rightarrow m := PUB) \\
\quad \bullet\ Init \,;\, \mu X \bullet (Read \Box Write \Box Switch) \,;\, X \\
\textbf{end}
\end{array}
$$

The denotational semantics of *Circus* is defined using Hoare and He's *Unifying Theories of Programming* (UTP) [6]. We describe this semantics here briefly, but we encourage the reader to consult the definitive account by Oliveira et al. [5].

Each *Circus* action A is defined as a *reactive design* of the form $\mathbf{R}(Pre \vdash Post)$, where Pre is a condition over the state variables that must hold for A to commence proper execution; and $Post$ describes a relation between initial states of A (satisfying Pre) and all later states that A may reach at a stable intermediate or final point in its execution.

In addition to the state variables of A, *Circus* features eight distinguished observational variables to model the behaviour of A as visible to the environment: ok records that A has been properly started; ok' records that A has reached a stable (observable) intermediate or final state; $wait$ records that A is waiting to commence its execution; $wait'$ indicates that A is awaiting interaction with the environment, while $\neg\, wait'$ indicates that A has terminated; tr records the process trace up to the point when A is started; tr' records the trace up to the intermediate or final state reached by A; ref' gives the set of events refused by A at the state it has reached; and ref is included for consistency.

3 Specifying Confidentiality Properties

In this section, we outline a lightweight notation for specifying confidentiality properties over *Circus* processes; we formalise its semantics in the next section.

Consider a *Circus* process P which interacts with a low-level user Low. We assume that Low may possess two sources of information about the behaviour of P: namely, (i) its own interactions with P (constituting partial observations of P's behaviour); and (ii) its *a priori* knowledge of P's design.

We model Low's interface to P in terms of the observational variables of P. We expect that Low can perceive whether the process is running normally (the ok and ok' variables) and whether the process is waiting for interaction with the environment (the $wait$ and $wait'$ variables). However, Low cannot observe P's state variables, because they are hidden from the environment [4].

We define Low's *window*, $\mathcal{L}$, to be the set of events communicated by P to the environment that are visible to Low through its interface. We model each event in $\mathcal{L}$ as a pair (c, i), where c is the channel name and i is the value transmitted on the channel [5]. The portion of the process trace $tr' - tr$ that is visible to Low is given by $(tr' - tr) \upharpoonright \mathcal{L}$. Moreover, the set of events refused by P that Low may perceive is given by $ref' \cap \mathcal{L}$.

We say that two behaviours of a process P are *Low-indistinguishable* if their valuations of ok, ok' and $wait, wait'$ are equal, the projections of their respective traces through $\mathcal{L}$ are identical and, if these behaviours are non-terminating, the projections of their refusal sets through $\mathcal{L}$ are identical.

Definition 1 (Indistinguishability). *The UTP predicate $I(\mathcal{L})$ captures the indistinguishability of two behaviours Φ and $\widetilde{\Phi}$ of P — where $\widetilde{\Phi}$ is expressed over a renaming of P's observational variables — as viewed through window $\mathcal{L}$:*

$$I(\mathcal{L}) \triangleq \begin{pmatrix} ok = \widetilde{ok} \wedge ok' = \widetilde{ok'} & \wedge\ (tr' - tr) \upharpoonright \mathcal{L} = (\widetilde{tr'} - \widetilde{tr}) \upharpoonright \mathcal{L} \\ \wedge\ wait = \widetilde{wait} \wedge wait' = \widetilde{wait'} & \wedge\ (wait' \Rightarrow ref' \cap \mathcal{L} = \widetilde{ref'} \cap \mathcal{L}) \end{pmatrix} \quad (1)$$

The notion of Low-indistinguishable behaviours is central to our formulation of confidentiality. A confidentiality property stipulates that, for each behaviour Φ of P involving a confidential activity ψ, it must be possible for P to exhibit alternative "cover story" behaviours that are Low-indistinguishable to Φ but do not involve ψ. The purpose of these non-confidential cover stories is to conceal occurrences of ψ from Low. If P may exhibit these cover story behaviours, then Low cannot deduce from its interaction with P that ψ has occurred, because Low is unable to rule out the possibility that any of the cover story behaviours associated with Φ may have occurred instead.

We introduce a confidentiality annotation, or *κ-annotation* for short, as our template for specifying confidentiality properties over *Circus* processes.

Definition 2 (κ-annotation). *A κ-annotation is a tuple $\langle C, O, \mathcal{L}, D \rangle$, where $\mathcal{L}$ denotes the window of Low, C and D are Circus actions and O is a set of Z schemata known*[1] *as* obligations*.*

The C action represents activities of the process over which a confidentiality property applies. Each obligation $\theta \in O$ is used in combination with C to specify — in terms of the initial and final states of C — which C activities of a process are classed as confidential and which C activities serve as cover stories for those confidential activities. θ is composed of two parts:

– The declaration part of θ specifies a *frame* of initial and final state variables of C (and their types), together with any input or output variables associated with events performed by C. These variables are partitioned into two classes: the confidential variables are denoted by v and v'; while the cover story variables are denoted using a renaming $(\widetilde{v}, \widetilde{v'})$ of v and v'.
– The predicate part of θ describes a relation between activities of C: for each confidential C activity expressed in terms of v and v', a range of cover story C activities are expressed using the $\widetilde{v}$ and $\widetilde{v'}$ variables.

An obligation expresses a closure condition over the behaviours of P. For each behaviour Φ of P featuring a C activity ψ that θ marks as confidential, θ demands that P exhibits *at least one* alternative Low-indistinguishable behaviour $\widetilde{\Phi}$ featuring a C activity $\widetilde{\psi}$ that is marked as a cover story for ψ by θ. Since θ may offer multiple cover stories, the designer of P has the flexibility to choose which of those cover story activities are exhibited by P.

Given two obligations θ_1 and θ_2 with the same frame, we say θ_2 is *at least as strong* as θ_1 if and only if (i) every activity ψ marked as confidential by θ_1 is also marked as confidential in θ_2; and (ii) every cover story activity for ψ required by θ_2 is also required by θ_1. Formally:

$$\theta_1 \leq \theta_2 \triangleq [\, \mathsf{conf}(\theta_1) \Rightarrow \mathsf{conf}(\theta_2) \,] \wedge [\, (\mathsf{conf}(\theta_1) \wedge \theta_2) \Rightarrow \theta_1 \,] \tag{2}$$

where $\mathsf{conf}(\theta)$ is defined to be $(\exists\, \widetilde{v}, \widetilde{v'} \bullet \theta)$ and $[\, X \,]$ denotes the universal closure of X over all variables [6].

The D parameter of a κ-annotation is a *Circus* action specifying activities of P that serve to declassify any confidential C activities that took place previously in P's execution. Hence, a κ-annotation becomes active when a C activity is

[1] The term “obligation” is borrowed from Seehusen and Stølen [7].

performed and then persists until it expires when a D activity is completed. To specify that declassification does not takes place, we can write $D = \mathit{Stop}$, since the Stop action (representing deadlock) never completes [5].

Example 2. Suppose we insist that whenever the *Cell* process (from Example 1) is operating in private mode, no information about the value written to the cell may be revealed to Low, until the cell reverts to public mode. We can specify a κ-annotation $\kappa_{pr} = \langle C_{pr}, O_{pr}, \mathcal{L}, D_{pr} \rangle$ to capture this requirement, where:

$$\begin{array}{ll} C_{pr} = \mathit{hin}?n \rightarrow \mathit{Skip} & O_{pr} = \bigcup_{x \in \mathbb{N}} \left\{ [\, m, \widetilde{m} : \{\mathit{PRV}\}, \widetilde{n?} : \mathbb{N} \mid \widetilde{n?} = x \,] \right\} \\ D_{pr} = \mathit{off} \rightarrow \mathit{Skip} & \mathcal{L} \quad = (\{\mathit{on}, \mathit{off}\} \times \{\mathit{Sync}\}) \cup (\{\mathit{lout}\} \times (\mathit{Mode} \times \mathbb{N})) \end{array}$$

By selecting the parameters of a κ-annotation carefully, we can encode a wide range of security requirements over the state and behaviour of *Circus* processes.

We can compare the strength of κ-annotations by lifting the $\leq$ ordering over their sets of obligations. Given two κ-annotations $\kappa_1 = \langle C, O_1, \mathcal{L}, D \rangle$ and $\kappa_2 = \langle C, O_2, \mathcal{L}, D \rangle$ over the same window, we write $\kappa_1 \preceq \kappa_2$ if and only if each obligation $\theta_1 \in O_1$ can be matched by an obligation $\theta_2 \in O_2$ such that $\theta_1 \leq \theta_2$.

Definition 3 (κ-ordering). *κ_2 is at least as strong as κ_1 if and only if:*

$$\kappa_1 \preceq \kappa_2 \triangleq \forall \theta_1 \in O_1 \bullet \exists \theta_2 \in O_2 \bullet \theta_1 \leq \theta_2 \tag{3}$$

4 The Semantics of κ-Annotations

In previous work, we have described a generic framework for expressing confidentiality properties over models of systems in the UTP [8]. We now adapt this framework to define the semantics of κ-annotations over *Circus* processes.

The observational variables of a *Circus* process model the process's behaviour in terms of all interactions that it may make with its environment. However, these variables do not record the multiple intermediate states that the process may pass through during its execution. We need to extend the semantics of *Circus* actions to capture these details.

A *snapshot* of an action A records the values of the observational variables of a process (abbreviated to x) immediately prior to an invocation of A at some point in the process's execution, together with the values of the observational variables of the process (abbreviated to x') at the intermediate or final state reached by A. We record each snapshot $i \geq 1$ by extending the alphabet of A with new lists of (fresh) observational variables x_i and x_i' that are isomorphic to x and x' respectively. We define a UTP healthiness condition $\mathbf{S}_i$ to extend *Circus* actions with the semantics of snapshots:

$$\mathbf{S}_i(A) \triangleq \begin{pmatrix} A \wedge (s_i \Rightarrow s_i') \wedge (x_i' = x_i \lhd s_i' = s_i \rhd x_i' = x' \wedge x_i = x) \\ \wedge\, (s_i' \wedge i > 1 \Rightarrow s_{i-1}') \wedge (s_i' \wedge \mathit{wait} \Rightarrow s_i) \end{pmatrix} \tag{4}$$

The Boolean variables s_i and s_i' are used to record whether snapshot i is triggered by the action. Each $\mathbf{S}_i$-healthy action triggers snapshot i only if s_i is unset and s_i' is set. Otherwise, the x_i and x_i' variables are kept constant by the action.

The order in which snapshots are triggered is monotonically increasing: for each $i > 1$, a $\mathbf{S}_i$-healthy action does not trigger snapshot i before snapshot $i-1$ is triggered. Finally, the $(s_i' \wedge \mathit{wait} \Rightarrow s_i)$ condition ensures that an action does not trigger a snapshot if the action is waiting to commence execution.

At the level of *Circus* processes, we write P_k to denote the process P, where each action of P is made $\mathbf{S}_i$-healthy for each $i \in 1..k$. We restrict P_k to exactly k snapshots by specifying $\neg\, s_i \wedge s_i'$ for each $i \in 1..k$ and hiding those variables from the environment:

$$P_k^+ \triangleq \exists\, s_1, s_1', \ldots, s_k, s_k' \bullet P_k \wedge \mathit{ok} \wedge (\forall\, i \in 1..k \bullet \neg\, s_i \wedge s_i') \tag{5}$$

In addition, the ok variable ensures that P_k^+ is properly started.

Given a κ-annotation $\langle C, O, \mathcal{L}, D\rangle$, let θ denote an obligation in O and ψ denote an activity of C marked as confidential by $\mathsf{conf}(\theta)$. Suppose that Low makes an interaction ϕ with P that is consistent with a behaviour $\varPhi$ of P featuring an instance of ψ. The predicate encoding all such behaviours of P is:

$$\mathsf{Sec}(P, C, \theta) \triangleq P_2^+ \wedge (C \wedge \neg\, C_f^f \wedge \neg\, \mathit{wait}' \wedge \mathsf{conf}(\theta))[x_1, x_2'/x, x'] \tag{6}$$

where $C \wedge \neg\, C_f^f \wedge \neg\, \mathit{wait}' \wedge \mathsf{conf}(\theta)$ denotes all non-diverging[2] and terminating activities of C that involve an activity described by $\mathsf{conf}(\theta)$. The x_1 and x_2' variables of P_2^+ record the state of P before and after such activities.

We say that $\varPhi$ *fulfils* θ if ψ also corresponds to at least one alternative Low-indistinguishable behaviour $\widetilde{\varPhi}$ of P featuring a cover story $\widetilde{\psi}$ given by θ that is related to ψ by θ. In other words, $\varPhi$ fulfils θ if Low cannot establish that ψ *must* have occurred from its observation of $\varPhi$, because Low cannot distinguish $\varPhi$ from $\widetilde{\varPhi}$ and so cannot rule out that $\widetilde{\psi}$ may have occurred instead. For this to be the case, each confidential behaviour of P encoded by Sec must also satisfy the predicate $\exists\, \widetilde{x}, \widetilde{x'} \bullet \mathsf{Cov}(P, \mathcal{L}, C, \theta)$:

$$\mathsf{Cov}(P, \mathcal{L}, C, \theta) \triangleq \begin{pmatrix} \exists\, \widetilde{x_1}, \widetilde{x_1'}, \widetilde{x_2}, \widetilde{x_2'} \bullet \widetilde{P_2^+} \wedge I(\mathcal{L}) \wedge J(\mathcal{L}) \\ \wedge\, (\widetilde{C} \wedge \neg\, \widetilde{C_f^f} \wedge \neg\, \widetilde{\mathit{wait}'} \wedge \theta)[x_1, x_2', \widetilde{x_1}, \widetilde{x_2'}/x, x', \widetilde{x}, \widetilde{x'}] \end{pmatrix} \tag{7}$$

where $\widetilde{A} = A[\widetilde{x}, \widetilde{x'}, \widetilde{x_1}, \widetilde{x_2}/x, x', x_1, x_2]$ and $J(\mathcal{L})$ is a predicate over the $\mathit{tr}_1, \mathit{tr}_2'$ and $\widetilde{\mathit{tr}_1}, \widetilde{\mathit{tr}_2'}$ snapshot trace variables that is satisfied only if the confidential and cover story activities of C take place at the same point of P's execution, as observed through the $\mathcal{L}$ window. $J(\mathcal{L})$ is defined as:

$$J(\mathcal{L}) \triangleq (\mathit{tr}_1 - \mathit{tr}) \upharpoonright \mathcal{L} = (\widetilde{\mathit{tr}_1} - \widetilde{\mathit{tr}}) \upharpoonright \mathcal{L} \wedge (\mathit{tr}_2 - \mathit{tr}_1) \upharpoonright \mathcal{L} = (\widetilde{\mathit{tr}_2} - \widetilde{\mathit{tr}_1}) \upharpoonright \mathcal{L} \tag{8}$$

Alternatively, $\varPhi$ fulfils θ if $\varPhi$ also features a declassification activity encoded by D that takes place in $\varPhi$ after ψ is performed:

$$\mathsf{Dec}(P, D) \triangleq \forall\, x_3, x_4' \bullet (D \wedge \neg\, D_f^f \wedge \neg\, \mathit{wait}')[x_3, x_4'/x, x'] \Rightarrow \exists\, x_3', x_4' \bullet P_4^+ \tag{9}$$

[2] A_c^b denotes $A[b, c/\mathit{ok}', \mathit{wait}]$ [5]. For each *Circus* action A we have $A = \mathbf{R}(\neg\, A_f^f \vdash A_f^t)$ [5,6], so $A \wedge \neg\, A_f^f$ denotes all behaviours of A where A's precondition is met.

A process P *satisfies* a κ-annotation $\kappa = \langle C, O, \mathcal{L}, D\rangle$ if and only if, for each obligation $\theta \in O$, each behaviour of P conforming to $\mathsf{Sec}(P, C, \theta)$ fulfils θ.

Definition 4 (Satisfaction). *P satisfies κ if and only if $P \models \kappa$ holds:*

$$P \models \langle C, O, \mathcal{L}, D\rangle \triangleq \forall \theta \in O \bullet \left[\begin{array}{r} \mathsf{Sec}(P, C, \theta) \wedge \neg\, \mathsf{Dec}(P, D) \\ \Rightarrow \exists\, \widetilde{x}, \widetilde{x'} \bullet \mathsf{Cov}(P, \mathcal{L}, C, \theta) \end{array}\right] \tag{10}$$

Example 3. The *Cell* process satisfies the κ_{pr} property. Informally, each behaviour of *Cell* featuring a *hin* event when $m = PRV$ cannot be distinguished by Low from every other *hin* event, so Low cannot rule out any of the cover stories specified by κ_{pr} until after the declassification event *off* takes place.

Lemma 1 states that the condition for verifying that P satisfies a κ-annotation is monotonic with respect to the κ-ordering; that is, if P satisfies κ, then it also satisfies all κ-annotations weaker than κ.

Lemma 1 (Monotonicity of $\models$). *If $P \models \kappa_2$ holds, then $P \models \kappa_1$ also holds for all κ_1 such that $\kappa_1 \preceq \kappa_2$.*

5 Propagation: Divide and Conquer!

Since the $\models$ condition is defined over the whole space of a process's behaviours, applying it to any non-trivial process could be extremely difficult in practice. In this section, we describe a procedure for verifying κ-annotations against a restricted, but useful, class of *Circus* processes.

This procedure requires a process's main action to consist of an initialisation action followed by a recursive loop over a generalised external choice of labelled guarded compound actions that we call *blocks*. Henceforth, we say that a process following this form is a *block-structured process* (BSP). We can divide a BSP into its component blocks and then identify proof obligations over individual blocks that imply the $\models$ condition.

Definition 5 (Block-structured process). *The main action of a Circus BSP with label set L is structured as follows:*

$$Init\ ;\ \left(\mu\, X \bullet \Box_{l \in L}(g.l\ \&\ A.l)\ ;\ X\right) \tag{11}$$

where the Init action initialises the process state and, for each $l \in L$, $g.l$ is a guard (on the v variables) and $A.l$ is a divergence-free (compound) Circus action. Each block $B.l$ behaves as $(g.l \wedge A.l)$ if $g.l$ holds and as Stop otherwise [5].

If process P is block-structured, we can safely assume (by Definition 5) that each block $B.l$ of P may only be started in states satisfying $g.l$.

Let $\delta \subseteq L \times L$ denote a relation between block labels that maps i to j if and only if P may perform $B.j$ immediately following $B.i$; that is, P may invoke $B.i$ in a state such that $B.i$ terminates in a state that satisfies $B.j$'s guard:

$$\delta \triangleq \left\{ i \mapsto j \,\middle|\, \exists\, x, x' \bullet \left(Init\, ;\, \mu\, X \bullet \left(\begin{array}{c} \Box_{l \in L}\, B.l\, ;\, X \\ \Box\ (B.i \wedge \neg\, wait') \end{array} \right)\right) \wedge g.j[v'/v] \right\} \tag{12}$$

In addition, let $\mathsf{AllDec}(B.l, D)$ denote a predicate that holds only if every terminating behaviour of $B.l$ started from a state satisfying $g.l$ involves a declassification activity specified by D:

$$\mathsf{AllDec}(B, D) \triangleq \begin{pmatrix} \forall\, x, x' \bullet B \wedge \neg\, B^f_f \wedge \neg\, wait' \\ \qquad \Rightarrow \exists\, x_1, x_1', x_2, x_2' \bullet B_2^+ \wedge (D \wedge \neg\, D^f_f)[x_1, x_2'/x, x'] \end{pmatrix} \tag{13}$$

5.1 Block-Level Verification

Given a BSP P and a κ-annotation κ, we analyse each block of P individually to verify that no sequence of blocks that P may perform can leak confidential information to Low. In effect, this analysis assumes that Low is able to observe the block sequence performed by P. This assumption is pessimistic but it is sound — as it over-approximates Low's observational abilities — and it enables us to reason about information flow to Low on a block-by-block basis.

Throughout this section, we assume that C and D actions specified by κ are *enclosed* by the blocks of P.

Definition 6 (Enclosure). *An action A is enclosed by a BSP P if and only if, whenever P can perform an activity described by $A \wedge \neg\, A^f_f \wedge \neg\, wait'$, that activity is performed within a single block of P.*

For each block $B.l$ of P, we need to identify all behaviours of $B.l$ featuring confidential activities that are *not* subsequently declassified (as specified by D) within $B.l$. These behaviours are given by the predicate $\mathsf{R}(\kappa, \theta, B.l)$:

$$\mathsf{R}(\kappa, \theta, B) \triangleq \exists\, x_1, x_1', x_2, x_2' \bullet \mathsf{Sec}(B, C, \theta) \wedge \neg\, \mathsf{Dec}(B, D) \tag{14}$$

For each of these behaviours of $B.l$ featuring a confidential activity ψ, the predicate $\mathsf{Q}(\kappa, \theta, B.l)$ relates that behaviour to all Low-indistinguishable behaviours of $B.l$ featuring cover story activities related to ψ by θ:

$$\mathsf{Q}(\kappa, \theta, B) \triangleq \forall\, x_1, x_1', x_2, x_2' \bullet \begin{pmatrix} \mathsf{Sec}(B, C, \theta) \wedge \neg\, \mathsf{Dec}(B, D) \\ \Rightarrow \mathsf{Cov}(B, \mathcal{L}, C, \theta) \end{pmatrix} \tag{15}$$

Observe that, if no C activity conforming to $\mathsf{conf}(\theta)$ can take place within $B.l$, then $\mathsf{R}(\kappa, \theta, B.l)$ will yield false.

It follows from the semantics of $\models$ that, in order for each behaviour Φ of $B.l$ given by $\mathsf{R}(\kappa, \theta, B.l)$ to fulfil θ within the context of $B.l$, there must exist at least one behaviour $\widetilde{\Phi}$ of $\widetilde{B.l}$ that $\mathsf{Q}(\kappa, \theta, B.l)$ associates with Φ. We define a proof obligation po to capture this requirement:

$$\mathsf{po}(B, (R, Q)) \triangleq \Big[\, B \wedge R \Rightarrow \exists\, \widetilde{x}, \widetilde{x'} \bullet \widetilde{B} \wedge Q \,\Big] \tag{16}$$

$\mathsf{po}(B.l, (\mathsf{R}(\kappa, \theta, B.l), \mathsf{Q}(\kappa, \theta, B.l)))$ treats $B.l$ in isolation from the other blocks of P. Hence, discharging this proof obligation does not guarantee that all behaviours of P fulfil θ, because Low may analyse its full interaction with P to obtain knowledge about the state of P before and after an invocation of $B.l$.

To ascertain that each P behaviour as a whole fulfils θ, we also need to verify that all sequences of blocks that P can perform do not reveal confidential information to Low about the behaviour of $B.l$. Hence, we introduce a procedure for *propagating* the R and Q predicates across the blocks of P, to enable us to verify in a piece-wise fashion that no possible execution of P can violate θ.

5.2 Forwards Propagation

Given a block $B.i$ where $R.i = \mathsf{R}(\kappa, \theta, B.i)$ and $Q.i = \mathsf{Q}(\kappa, \theta, B.i)$, we can calculate a pair of predicates $(R', Q') = \mathsf{fw}\,(B.i, (R.i, Q.i))$ encoding all final states of $B.i$ that can be reached by all terminating behaviours of $B.i$ classed as confidential by $R.i$, together with the final states of all Low-indistinguishable (and terminating) behaviours of $B.i$ classed as cover stories by $Q.i$:

$$\mathsf{fw}\,(B, (R, Q)) \triangleq \begin{pmatrix} (\exists\, x \bullet B \wedge R \wedge \neg\, wait')[x/x'], \\ (\exists\, x, \widetilde{x} \bullet B \wedge \widetilde{B} \wedge Q \wedge I^*(\mathcal{L}) \wedge \neg\, wait')[x, \widetilde{x}/x', \widetilde{x'}] \end{pmatrix} \quad (17)$$

The $I^*(\mathcal{L})$ predicate denotes $I(\mathcal{L})$ extended with Low's ability to perceive deadlock of P when $B.i$ terminates, which arises only if $B.i$ reaches a final state in which none of the guards of the blocks of P are enabled. Hence, if a behaviour of $B.i$ involving a confidential activity terminates in a deadlocking state, then the associated cover story behaviours of $B.i$ should also terminate in a deadlocking state, to preserve Low-indistinguishability:

$$I^*(\mathcal{L}) \triangleq I(\mathcal{L}) \wedge (\neg\, wait' \wedge \forall\, l \in L \bullet \neg\, g.l[v'/v] \Rightarrow \neg\, \forall\, l \in L \bullet \neg\, \widetilde{g.l}[\widetilde{v'}/\widetilde{v}]) \quad (18)$$

We have $I^*(\mathcal{L}) = I(\mathcal{L})$ if every final state that a block of P may reach enables one or more guards of P.

For each block $B.j$ where $i \mapsto j \in \delta$, we need to verify that Low's interactions with $B.j$ do not provide Low with information about the behaviour of $B.i$ that allows Low to retrospectively rule out all cover stories given by $Q.i$. This is guaranteed by $\mathsf{po}(B.j, \mathsf{fw}\,(B.i, (R.i, Q.i)))$, which implies that if $B.j$ is started from any final state of $B.i$ marked as confidential, then each interaction with $B.j$ that Low may make could have instead been generated by $B.j$ started from any final state of $B.i$ marked as a cover story.

Moreover, we need to prove that each sequence of blocks that P can perform following $B.i$ does not leak confidential information to Low. This can be done by verifying the last block in the sequence against the (R, Q) pair obtained by recursively propagating $(R.i, Q.i)$ forwards through each block in the sequence.

We can incorporate declassification into forwards propagation by altering the δ relation. For each block $B.t$ where $\mathsf{AllDec}(B.t, D)$ holds, we need not propagate a confidentiality requirement further than $B.t$, because it is relaxed when $B.t$ terminates. Thus, we can remove from δ all transitions leading from $B.t$:

$$\delta_D \triangleq \delta \setminus \{t \mapsto j \mid t, j \in L \wedge \mathsf{AllDec}(B.t, D)\} \quad (19)$$

We now formulate a verification condition for the forwards propagations of κ over all blocks of a process. For each block $B.l$, the set of all forwards propagations of the obligations contained within κ of all blocks of P that may lead to an invocation of $B.l$ is given by $\overrightarrow{\rho}(\kappa, l)$:

$$\overrightarrow{\rho}(\kappa, l) \triangleq \{\rho \mid i \in L \wedge \theta \in O \wedge (l, \rho) \in \mathsf{fwds}(\{(i, \mathsf{RQ}(\kappa, \theta, B.i))\})\} \tag{20}$$

$$\mathsf{fwds}(K) \triangleq K \cup \mathsf{fwds}\left(\{(j, \mathsf{fw}\,(B.i, \rho)) \mid i \mapsto j \in \delta_D \wedge (i, \rho) \in K\}\right) \tag{21}$$

where ρ denotes (R, Q) and $\mathsf{RQ}(\kappa, \theta, B)$ is shorthand for $(\mathsf{R}(\kappa, \theta, B), \mathsf{Q}(\kappa, \theta, B))$. We calculate $\mathsf{fwds}(K)$ by iterating until a fixed point is reached.

The set $\overrightarrow{\rho}(\kappa, l)$ contains all forwards propagations of κ through all sequences of blocks from $B.i$ up to $B.l$. This set represents a sound approximation of κ through the blocks of the process up to $B.l$. Hence, it is sufficient to discharge $\mathsf{po}(B.l, \rho)$ for each $\rho \in \overrightarrow{\rho}(\kappa, l)$ in order to verify that $B.l$ does not reveal information about the state of P to Low that could violate any instance of κ applicable to the blocks that may take place prior to $B.l$.

5.3 Backwards Propagation

Forwards propagation is capable of verifying that confidential information about the behaviour of each block $B.i$ is not disclosed by any sequence of blocks that may follow $B.i$. However, this procedure leaves open the possibility that a process may have performed a sequence of blocks leading up to a state in which $B.i$ may perform a confidential activity but cannot instead perform the requisite cover story activities required by κ. Again, it may be possible for Low to infer confidential information about the behaviour of $B.i$ by analysing its interaction with the process to identify information about the initial state of $B.i$.

To ensure that Low cannot rule out cover stories for $B.i$ based on its knowledge about the behaviour of the previous blocks executed by a process, it suffices to propagate each $\mathsf{RQ}(\kappa, \theta, B.i)$ pair backwards to all blocks that may precede $B.i$. The confidentiality requirement on the process state immediately prior to an invocation of $B.i$ is given by $\mathsf{bw}\,(B.i, \mathsf{RQ}(\kappa, \theta, B.i))$:

$$\mathsf{bw}\,(B, (R, Q)) \triangleq \begin{pmatrix} (\exists\, x' \bullet B \wedge R \wedge \neg\, wait)[x'/x], \\ (\exists\, x', \widetilde{x'} \bullet B \wedge \widetilde{B} \wedge Q \wedge I(\mathcal{L})) \wedge \neg\, wait)[x', \widetilde{x'}/x, \widetilde{x}] \end{pmatrix} \tag{22}$$

$$\overleftarrow{\rho}(\kappa, l) \triangleq \{\rho \mid j \in L \wedge \theta \in O \wedge (l, \rho) \in \mathsf{bwds}(\{(j, \mathsf{RQ}(\kappa, \theta, B.j))\})\} \tag{23}$$

$$\mathsf{bwds}(K) \triangleq K \cup \mathsf{bwds}\left(\{(i, \mathsf{bw}\,(B.j, \rho)) \mid i \mapsto j \in \delta \wedge (j, \rho) \in K\}\right) \tag{24}$$

$\overleftarrow{\rho}(\kappa, l)$ gives the set of all backwards propagations of κ from the blocks of the process to $B.l$. Again, discharging $\mathsf{po}(B.l, \rho)$ for each $\rho \in \overleftarrow{\rho}(\kappa, l)$ suffices to establish that $B.l$ does not reveal information about the state of P to Low that may violate the κ-annotations of the blocks following $B.l$.

It is also necessary to verify that all initial states of P satisfy the backwards-propagated κ-annotations of the blocks that may be performed immediately after *Init*. This is assured by discharging $\mathsf{po}(Init, \overleftarrow{\rho}(\kappa, Init))$, where:

$$\overleftarrow{\rho}(\kappa, Init) \triangleq \bigcup_{l \in L} \{\mathsf{bw}(B.l, \rho) \mid \rho \in \overleftarrow{\rho}(\kappa, l) \wedge (\exists\, x, x' \bullet Init \wedge g.l[v'/v])\} \tag{25}$$

5.4 Verifying Confidentiality

Forwards and backwards propagation can be applied together to a BSP P to verify each block of P against κ. We now sketch a proof that this procedure — with one important caveat — is sufficient to demonstrate that P satisfies κ.

First, we relate the po proof obligation to the $\models$ condition. Given a κ-annotation $\langle C, O, \mathcal{L}, D \rangle$ and an individual block B, if we can demonstrate that each behaviour of B conforming to $\mathsf{Sec}(B, C, \theta)$ fulfils θ for each $\theta \in O$, then we can conclude that B satisfies κ in isolation. This is formalised in Lemma 2.

Lemma 2 (po entails $\models$). *For any block B and κ-annotation $\kappa = \langle C, O, \mathcal{L}, D \rangle$:*

$$\forall\, \theta \in O \bullet \mathsf{po}(B, \mathsf{RQ}(\kappa, \theta, B)) \quad \textit{implies} \quad B \models \kappa \tag{26}$$

We now extend the scope of Lemma 2 to enclose multiple blocks. Consider any two blocks $B.i$ and $B.j$ of P where $i \mapsto j \in \delta$ and $B.i$ or $B.j$ may exhibit an activity marked as confidential by κ. To justify that $(B.i\,;B.j) \models \kappa$ holds, we need to discharge four proof obligations over $B.i$ and $B.j$ for each $\theta \in O$:

$$\begin{array}{rl} \mathsf{po}(B.i, \mathsf{RQ}(\kappa, \theta, B.i)) & \quad \mathsf{po}(B.j, \mathsf{RQ}(\kappa, \theta, B.j)) \\ \mathsf{po}(B.j, \mathsf{fw}\,(B.i, \mathsf{RQ}(\kappa, \theta, B.i))) & \quad \mathsf{po}(B.i, \mathsf{bw}\,(B.j, \mathsf{RQ}(\kappa, \theta, B.j))) \end{array}$$

Together, these four proof obligations imply that each behaviour of $(B.i\,;B.j)$ marked as confidential by C and θ is concealed by at least one alternative Low-indistinguishable behaviour of $(B.i\,;B.j)$ marked as a cover story.

In order to generalise this result to arbitrary sequences of blocks, we need to restrict our attention to κ-annotations where each obligation features *exactly one* cover story activity for each confidential activity. (This restriction ensures the same cover story is propagated forwards and backwards through the process.)

Definition 7 ($\widehat{\kappa}$-annotation). *A $\widehat{\kappa}$-annotation $\langle C, O, \mathcal{L}, D \rangle$ is a κ-annotation where the condition $\forall\, v, v' \bullet \mathsf{conf}(\theta) \Rightarrow \exists_1\, \widetilde{v}, \widetilde{v'} \bullet \theta$ holds for each $\theta \in O$.*

Naturally, a $\widehat{\kappa}$-annotation can always be obtained from a κ-annotation κ by strengthening each obligation of κ.

We say that P is *$\widehat{\kappa}$-safe* if we can prove the blocks of P uphold the respective forwards and backwards propagations of $\widehat{\kappa}$ through P.

Definition 8 ($\widehat{\kappa}$-safety). *A BSP P is safe w.r.t. $\widehat{\kappa} = \langle C, O, \mathcal{L}, D \rangle$ if and only if P encloses C and D; $\mathsf{po}(\mathit{Init}, \overleftarrow{\rho}(\widehat{\kappa}, \mathit{Init}))$ holds and, for each $l \in L$, we have:*

$$\mathsf{po}(B.l, \mathsf{RQ}(\widehat{\kappa}, \theta, B.l)) \quad \textit{and} \quad \forall\, \rho \in \overrightarrow{\rho}(\widehat{\kappa}, l) \cup \overleftarrow{\rho}(\widehat{\kappa}, l) \bullet \mathsf{po}(B.l, \rho) \tag{27}$$

If process P is $\widehat{\kappa}$-safe, then no sequence of blocks that P may perform may leak confidential information to Low. It follows that, if P is $\widehat{\kappa}$-safe, then P must satisfy κ. This result is encapsulated by Theorem 1.

Theorem 1. *If a BSP P is $\widehat{\kappa}$-safe, then $P \models \widehat{\kappa}$ holds.*

A trivial consequence of Theorem 1 and Lemma 1 that, if P is $\widehat{\kappa}$-safe and $\kappa \preceq \widehat{\kappa}$ holds, then P must satisfy κ as well as $\widehat{\kappa}$.

6 Confidentiality-Preserving Refinement

Behavioural refinement maintains the functionality of *Circus* processes, but may not preserve confidentiality properties [9]. This so-called "refinement paradox" arises because naïvely refining away non-determinism within a process P may remove behaviours from P that are required as cover stories by the κ-annotations of P, without also removing the associated confidential behaviours. Such refinement steps violate the $\models$ condition.

Example 4. Consider the following refinement of the *Read* block of *Cell*:

$$Read \triangleq \left(\begin{array}{l} \quad m = PUB \;\&\; lout!(PUB, val) \rightarrow Skip \\ \Box\; m = PRV \;\&\; lout!(PRV, val \;\mathsf{mod}\; 2) \rightarrow Skip \\ \Box\; hout!val \rightarrow Skip \end{array} \right)$$

This refinement is manifestly insecure with respect to κ_{pr} specified in Example 2, because if Low observes a *lout* event when $m = PRV$, it can deduce one bit of information about the value of *val*, in violation of half of the obligations of κ_{pr}.

When developing a process by stepwise refinement, it would be wasteful to reach a concrete process design that violates its κ-annotations. This problem can be overcome by strengthening process refinement in *Circus* to uphold κ-annotations.

Definition 9 (Secure process refinement). *For processes P_1 and P_2 we say that P_2 is a* secure refinement *of P_1 w.r.t. a κ-annotation κ — written $P_1 \sqsubseteq_{\mathcal{P}}^{\kappa} P_2$ — if P_2 is a process refinement of P_1 and P_2 satisfies κ.*

Observe that $P_1 \sqsubseteq_{\mathcal{P}}^{\kappa} P_2$ requires only that P_2 satisfies κ, whereas P_1 itself need not be secure. However, an insecure refinement may result in a process that cannot be refined securely, so we propose that a BSP should be verified to satisfy its κ-annotations by applying propagation at an early stage of its development. Thereafter, each refinement step should maintain those κ-annotations. This can be achieved by retaining the $\overrightarrow{\rho}$ and $\overleftarrow{\rho}$ sets generated by propagation and reusing them at each refinement step to verify that it preserves $\widehat{\kappa}$-safety.

Consider a BSP P that has been proved to be $\widehat{\kappa}$-safe. By Definition 8, we know that each block $B.l$ of P upholds each member of $\overrightarrow{\rho}(\widehat{\kappa}, l)$ and $\overleftarrow{\rho}(\widehat{\kappa}, l)$, as well as $\mathsf{RQ}(\kappa, \theta, B.l)$. Suppose we refine P to obtain a BSP P' by replacing $B.l$ in P with $B.l'$, where $B.l \sqsubseteq B.l'$. If we have $g.l = g.l'$, then P' can only feature transitions between blocks that are possible for P (i.e. $\delta_{P'} \subseteq \delta_P$). In Theorem 2, we present conditions on $B.l'$ that are sufficient to establish that P' is $\widehat{\kappa}$-safe.

Theorem 2. *If P is $\widehat{\kappa}$-safe and P' equals P except with $B.l'$ in place of $B.l$ (where $g.l = g.l'$ and $B.l \sqsubseteq B.l'$), then P' is $\widehat{\kappa}$-safe if for* every *cover story behaviour of $B.l$ specified by $\mathsf{RQ}(\widehat{\kappa}, \theta, B.l)$ and each member of $\overrightarrow{\rho}(\widehat{\kappa}, l) \cup \overleftarrow{\rho}(\widehat{\kappa}, l)$ where $B.l'$ features an associated confidential behaviour, the same cover story behaviour is present in $B.l'$. These conditions are formalised as follows:*

$$\forall (R, Q) \in \{\mathsf{RQ}(\widehat{\kappa}, \theta, B.l)\} \cup \overrightarrow{\rho}(\widehat{\kappa}, l) \cup \overleftarrow{\rho}(\widehat{\kappa}, l) \bullet \mathsf{saferef}(B.l, B.l', (R, Q))$$

$$\textit{where} \quad \mathsf{saferef}(B, B', (R, Q)) \triangleq \left[B' \wedge R \wedge \widetilde{B} \wedge Q \Rightarrow \widetilde{B'} \right]$$

7 Related Work

In software engineering, it is conventional to implement security policies by building access control into the system design. However, the notion of information flow security is more generally applicable than access control: while confidentiality properties specify *what* information should not be disclosed to low-level users, access control mechanisms describe *how* that information should be protected. Furthermore, access control does not account for Low inferring secret information indirectly from its interaction with a system [1].

Our κ-annotations share the spirit of Jacob's *security specifications* [9,10], which are functions from low-level observations of a system to the minimal set of system behaviours that a low-level user must be unable to distinguish from those observations. The same idea underlies our earlier work on encoding confidentiality properties in the UTP [8], where we define an abstract formulation of confidentiality properties across the spectrum of UTP theories. By specialising this formulation to the semantics of *Circus*, we have achieved a framework where these properties can be integrated directly into formal software developments.

In many of the existing frameworks for expressing confidentiality properties, such as Mantel's MAKS [11], the occurrence (or non-occurrence) of particular high-level events is taken to be confidential. We abstract from this event-centric style by taking *Circus* actions over the state of processes as the basis of our confidentiality encoding. In addition, our model of Low's observational abilities is based on the failures-divergences semantics of CSP — as encoded in the UTP semantics of *Circus* — and is therefore richer than the trace-based models of Low's observations frequently employed in these frameworks.

The confidentiality properties encoded by our κ-annotations are (in general) weaker than the *noninterference* property, which stipulates that no input from a high-level user can influence any output to a low-level user [12]. We contend that a more fine-grained approach to specify limits on information flow to low-level users is beneficial to software engineers, because it affords greater flexibility in designing systems to meet their functionality and security requirements.

Unlike noninterference, our notion of confidentiality does not stop a trusted (yet treacherous) high-level user from actively leaking secrets to low-level users by influencing their interactions with the system using a pre-arranged signalling protocol. While deliberate disclosure of secret data by malicious high-level users is troublesome in security-critical environments, we contend that no technical measures can prevent such users from leaking data outside the system domain.

Our propagation procedure is related to the *unwinding* technique [12], which aims to simplify the task of verifying a system against a confidentiality property. Unwinding transforms a global confidentiality property (typically expressed in terms of trace sets) over a system into conditions over its individual state transitions, which can then be discharged using traditional proof methods [11].

Our work shares some ideas with Morgan's recent *shadow semantics* [13,14], which extends the refinement calculus for sequential programs to ensure that refinement does not introduce new information flows about secret data to Low. The shadow semantics assumes that Low can observe a program's control flow;

likewise, we assume that Low can deduce the sequence of blocks performed by a process. However, the shadow semantics goes a step further, by distinguishing between *atomic* and *composite* non-determinism and allowing Low to monitor how composite non-determinism is resolved in a program's execution. This means that refinement of composite non-determinism is security-preserving. We do not grant Low that ability, because κ-annotations do not cleanly partition the whole process state into secret and non-secret variables. Moreover, applying the shadow semantics for refinement in our framework would require *Circus* to be extended with an alternative semantics for non-determinism.

We have covered many of the topics discussed here in greater depth in earlier work [8]. A fuller survey of the various approaches for formalising and reasoning about information flow security can be found elsewhere [1,2].

8 Conclusions

In this paper, we have presented a framework for specifying confidentiality properties over *Circus* processes and a procedure for verifying that a *Circus* process satisfies those properties. The close integration of our framework with the specification facilities of *Circus* is original and is supported by the UTP foundations of *Circus*. While we have taken *Circus* as the formal foundation of our framework, the underlying principles are general and could be translated to other formalisms besides *Circus* (especially those with a UTP semantics).

Our ongoing research aims to elevate confidentiality properties to the status of "first-class citizens" in *Circus* developments, with suitable techniques and automated tools to support the verification of process designs against confidentiality properties and for checking the correctness of refinement steps. We hypothesise that the work presented in this paper, together with the underlying *Circus* platform, may provide the foundations of a viable engineering approach for developing software in tandem with information flow security concerns. We are currently working on a case study project to evaluate this hypothesis.

We have left several pertinent topics unexplored in this paper, such as the consequences of concurrency and probabilistic behaviour for information flow and confidentiality. We leave the investigation of these topics to future work.

Finally, taking a formal approach to security engineering can increase confidence that a system does not leak secrets to low-level users, but it is unwise to assume that any system implementation is secure in all circumstances. In particular, our framework does not address sources of information leakage that arise at the hardware level, such as its responsiveness or power consumption [15]. The task of extending formal methods to address these factors is likely to be challenging, but would be a significant step towards engineering secure systems.

Acknowledgements. Michael Banks is supported by a UK EPSRC DTA studentship. We are grateful to the anonymous referees for their helpful comments and to Matthew Naylor for proofreading.

References

1. McLean, J.: Security models. In Marciniak, J., ed.: Encyclopedia of Software Engineering. Volume 2. John Wiley & Sons, Inc. (1994) 1136–1145
2. Ryan, P.: Mathematical models of computer security. In: Foundations of Security Analysis and Design. Volume 2171 of Lecture Notes in Computer Science., Springer Berlin / Heidelberg (2001) 1–62
3. Cavalcanti, A., Sampaio, A., Woodcock, J.: A refinement strategy for Circus. Formal Aspects of Computing **15**(2-3) (November 2003) 146–181
4. Oliveira, M.V.: Formal Derivation of State-Rich Reactive Programs using Circus. PhD thesis, Department of Computer Science, University of York (2005)
5. Oliveira, M., Cavalcanti, A., Woodcock, J.: A UTP semantics for Circus. Formal Aspects of Computing **21**(1) (February 2009) 3–32
6. Hoare, C.A.R., He, J.: Unifying Theories of Programming. Prentice Hall International Series in Computer Science. Prentice Hall Inc. (1998)
7. Seehusen, F., Stølen, K.: Information flow security, abstraction and composition. IET Information Security **3**(1) (2009) 9–33
8. Banks, M.J., Jacob, J.L.: Unifying theories of confidentiality. In Qin, S., ed.: 3rd International Symposium on Unifying Theories of Programming (UTP 2010). Volume 6445 of Lecture Notes in Computer Science., Springer Berlin / Heidelberg (2010) 120–136
9. Jacob, J.L.: On the derivation of secure components. In: Proceedings of the 1989 IEEE Symposium on Security and Privacy, IEEE Computer Society (1989) 242–247
10. Jacob, J.L.: Security specifications. In: Proceedings of the 1988 IEEE Symposium on Security and Privacy. (1988) 14–23
11. Mantel, H.: A Uniform Framework for the Formal Specification and Verification of Information Flow Security. PhD thesis, Universität Saarbrücken (July 2003)
12. Goguen, J.A., Meseguer, J.: Unwinding and inference control. In: Proceedings of the 1984 IEEE Symposium on Security and Privacy, IEEE Computer Society (1984) 75–86
13. Morgan, C.: The shadow knows: Refinement and security in sequential programs. Science of Computer Programming **74**(8) (June 2009) 629–653
14. Morgan, C.: Compositional noninterference from first principles. Formal Aspects of Computing (To appear).
15. Clark, J.A., Stepney, S., Chivers, H.: Breaking the model: Finalisation and a taxonomy of security attacks. Electronic Notes in Theoretical Computer Science **137**(2) (July 2005) 225–242

A Proofs

Formal proofs of the theorems and lemmas presented in this paper are available from `http://www-users.cs.york.ac.uk/~mbanks/`.