Unifying Theories of Confidentiality

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 a framework for reasoning about the security of confidential data within software systems. A novelty is that we use Hoare and He's *Unifying Theories of Programming* (UTP) to do so and derive advantage from this choice. We identify how information flow between users can be modelled in the UTP and devise conditions for verifying that system designs may not leak secret information to untrusted users. We also investigate how these conditions can be combined with existing notions of refinement to produce refinement relations suitable for deriving secure implementations of systems.
Keywords: UTP, computer security, information flow, confidentiality properties, confidentiality-preserving refinement, MAKS.

1 Introduction

When designing a computer system that stores or manipulates confidential data, it is vital to define a security policy for this data and to obtain reliable assurances that the system never discloses this data to untrusted users in violation of the security policy. In software engineering, a security policy is usually implemented by combining access control mechanisms with user authentication schemes and cryptographic measures [1]. However, access control does not restrict how data can be manipulated once it has been released to users, so it cannot prevent confidential data from propagating indirectly to low-level users [2].

A radically different approach to computer security is to specify constraints on the *information flow* between a system and its users. We say that information flows from a high-level user ($\mathcal{H}$) to a low-level user ($\mathcal{L}$) whenever $\mathcal{L}$'s observations of a system are perturbed by the activities performed by $\mathcal{H}$. To ensure that a system does not leak sensitive or valuable data to $\mathcal{L}$, it is necessary to design the system with built-in restrictions on information flow [1]. These constraints are known as *confidentiality properties*, since they codify an upper limit on the flow of information about $\mathcal{H}$ activities classed as confidential to $\mathcal{L}$. Confidentiality properties specify *what* information must not be disclosed to $\mathcal{L}$, but unlike models of access control, they allow the designers of a system to decide *how* that information should be protected.

The confidentiality property known as *noninterference* [3] mandates that $\mathcal{H}$'s actions must have no effect on $\mathcal{L}$'s observations of the system; effectively, $\mathcal{L}$ cannot determine whether or not $\mathcal{H}$ has interacted with the system at all. It follows that a system satisfying noninterference cannot disclose confidential

information to $\mathcal{L}$. However, noninterference is too strong a requirement for many kinds of software products, because it is often necessary for $\mathcal{H}$ to communicate some kinds of data to $\mathcal{L}$. We contend that software developers should be able to specify custom confidentiality properties that are tailored to the intricacies of the system domain, rather than choosing from a limited range of ready-made noninterference-like confidentiality properties.

The central topic of this paper is a novel encoding of confidentiality properties in Hoare and He's Unifying Theories of Programming (UTP) [4]. Our encoding is inspired by Mantel's *Modular Assembly Kit for Security Properties* (MAKS) [5,6]. The MAKS is designed for expressing a wide range of confidentiality properties in the security literature in a uniform trace-based style. We generalise the foundations of the MAKS to the UTP, which allows us to define a class of confidentiality properties in a predicate style across the spectrum of existing UTP theories. Moreover, these predicates can be applied to verify that a software specification (written in a language with a UTP semantics) satisfies a given confidentiality property by means of formal proof.

This paper is structured as follows. We provide an overview of the UTP in Section 2. In Section 3, we describe the foundations of our approach for modelling the observations of users in the UTP. This approach provides the basis of our encoding of confidentiality properties in the UTP, which we present in Section 4. We investigate how refinement can be extended to accommodate confidentiality properties in Section 5. We examine the relationship between our work and existing techniques for applying confidentiality properties in rigorous software developments in Section 6 and summarise our work in Section 7.

2 Unifying Theories of Programming

The UTP provides an abstract framework for modelling the denotational semantics of a wide range of programming paradigms and the features of programming languages [4]. The UTP features a powerful standalone notation for specifying the behaviour of systems and reasoning about program correctness.

The mathematical foundation of the UTP is the alphabetised relational calculus. In the UTP, specifications and program constructs alike are expressed as predicates over an alphabet (a set) of observational variables. The observational variables in a predicate record all of the information about a program's execution that is visible when an observation of the program is made. These variables are grouped into two classes: by convention, undashed variables ($x, y, \ldots$) record the initial observation of the program state taken before execution commences, while dashed variables ($x', y', \ldots$) record an intermediate observation of the program state taken during (or after) the program's execution [4].

Example 1 gives a UTP specification of a simple system that we use later.

Example 1. Consider a simple guessing game played by two users Laurel ($\mathcal{L}$) and Hardy ($\mathcal{H}$). $\mathcal{H}$ chooses a number $n \in 0..9$, which is concealed from $\mathcal{L}$. $\mathcal{L}$ makes a guess g of the value of n. $\mathcal{L}$ is informed whether its guess was correct or

greater or smaller than n by setting t to 0, a positive value or a negative value respectively.

We model all acceptable runs of the game by defining a UTP predicate G_0:

$$G_0 \triangleq n \in 0..9 \land (g = n \Rightarrow t = 0) \land (g > n \Rightarrow t > 0) \land (g < n \Rightarrow t < 0) \quad (1)$$

While G_0 represents the game as seen by the environment as a whole, we expect that $\mathcal{L}$ is unable to observe the value of n. However, this expectation is not recorded in G_0. If we wish to reason about $\mathcal{L}$'s observations of G_0, we need to define a mapping from G_0's behaviours to the aspects of G_0 visible to $\mathcal{L}$.

3 Modelling User Observations

This section outlines a UTP approach for modelling the observational abilities of users. We cover this approach in greater depth elsewhere [7]; here, we present only the details that are essential for studying information flow in the UTP.

Observations. An observation of a UTP predicate S is any predicate that maps each variable in S's alphabet to a single value, such that S is satisfied by that mapping. (For example, $n = 3 \land g = 7 \land t = 42$ is a valid observation of G_0.)

Given a predicate S representing a system, we distinguish between two classes of observations of S. A *system-level* observation of S describes the behaviour of S in its entirety. The list of undashed variables $(s_1, \ldots, s_i)$ of a system-level observation is denoted by s and the corresponding list of dashed variables by s'. Whenever Φ is a system-level observation of S, we have:

$$(\exists_1 s, s' \bullet \Phi) \land [\Phi \Rightarrow S] \quad (2)$$

The first condition requires that Φ is satisfied by exactly one valuation of the system-level variables in S's alphabet. The second condition states that Φ is an actual observation of S.

It is often reasonable to expect that individual users cannot observe all of a system's behaviour; rather, each user is provided with an interface to the system, through which it can observe some elements of the system's behaviour. We say that a user's observation of a system is an *interface-level* observation. To record interface-level observations, we introduce separate lists of interface-level observational variables $u = (u_1, \ldots, u_j)$ and $u' = (u'_1, \ldots, u'_j)$, which must be disjoint from s and s'.

Views. A view is a predicate that formalises a user's interface to a system by defining a total relation from system-level observations to interface-level observations[1]. The interface-level observational variables in a view's alphabet are

[1] The theory presented in Section 4 is unchanged if views are required to be functional. However, by relaxing this requirement, we allow for the possibility that a user's interface gives a non-deterministic (noisy) representation of a system's behaviour.

decorated with the view's identifier: for example, x_V refers to an interface-level variable associated with view V, whereas x refers to a system-level variable.

Given a system-level observation Φ and a view V, the set of interface-level observations corresponding to Φ is given by calculating the image of Φ as projected through V. A view should not restrict the domains of the system-level observational variables in any way; that is, predicates such as $x < y$, $x' = 0$ and **false** are not views. Moreover, we exclude "time-travelling" predicates such as $x_V = x'$ (which demands a user with view V can observe the value of x' *before* it is computed) from the space of views. (These constraints on views may be formalised in the UTP as healthiness conditions [7].)

Example 2. Returning to Example 1, we may expect that $\mathcal{H}$ can observe the values of all variables in G_0's alphabet, but the value of n is hidden from $\mathcal{L}$. The observational abilities of $\mathcal{L}$ and $\mathcal{H}$ are modelled by the following views:

$$L \triangleq g_L = g \land t_L = t \tag{3}$$

$$H \triangleq g_H = g \land n_H = n \land t_H = t \tag{4}$$

The view H gives $\mathcal{H}$ total information about the state of G_0. The view L allows $\mathcal{L}$ to observe its own guess and the outcome of that guess, but it does not directly reveal the value of n to $\mathcal{L}$.

We insist that views associated with different users are *interface-disjoint*; that is to say, they share no interface-level observational variables.

Calculating Interface-Level Predicates. Given a system-level predicate S and a view V, we can derive a predicate that encodes the space of all interface-level observations that can be made of S when viewed through V. We define a predicate transformer $\mathbf{P}$ (for "project") to calculate this interface-level predicate:

$$\mathbf{P}(V, S) \triangleq \exists s, s' \bullet V \land S \tag{5}$$

The predicate $\mathbf{P}(V, S)$ is the image of S as projected through V. Hence, $\mathbf{P}(V, S)$ is satisfied by exactly those interface-level observations that can be made by monitoring the behaviour of S through V.

Given two interface-disjoint views $V_1(s_1, u_1)$ and $V_2(s_2, u_2)$ of the same system (i.e. $s_1 = s_2$), the view $V_1 \land V_2$ represents the combination of the interfaces corresponding to V_1 and V_2. Hence, $\mathbf{P}(V_1 \land V_2, S)$ is a predicate with alphabet $u_1 \cup u_2$ describing the relation between interface-level observations of S as made through V_1 and V_2.

Example 3. Applying $\mathbf{P}(L \land H)$ to G_0 yields the predicate:

$$\begin{aligned} \mathbf{P}(L \land H, G_0) &= \exists g, n, t \bullet \begin{pmatrix} g_L = g \land t_L = t \\ \land\ g_H = g \land n_H = n \land t_H = t \end{pmatrix} \land G_0 \\ &= \begin{pmatrix} n_H \in 0..9 \land g_L = g_H \land t_L = t_H \\ \land\ (g_H = n_H \Rightarrow t_L = 0) \\ \land\ (g_H > n_H \Rightarrow t_L > 0) \land (g_H < n_H \Rightarrow t_L < 0) \end{pmatrix} \end{aligned} \tag{6}$$

This predicate represents all compatible $\mathcal{H}$- and $\mathcal{L}$-observations of G_0.

4 Encoding Confidentiality Properties

When designing a system S, we may wish to prevent $\mathcal{L}$ from acquiring information about confidential features of $\mathcal{H}$'s interactions with S. Hence, our goals are to specify which aspects of $\mathcal{H}$'s observations of S are classed as confidential and to verify that $\mathcal{L}$ cannot use its observations of S to deduce information about those aspects.

In the worst case, $\mathcal{L}$ may possess complete knowledge of the implementation of S and the structure of its own view and that of $\mathcal{H}$. Hence, for each observation ϕ that $\mathcal{L}$ can make of S, $\mathcal{L}$ can deduce the set of $\mathcal{H}$ observations consistent with ϕ. The predicate that encodes this subset of $\mathcal{H}$ observations is given by:

$$\Omega_L^H(S, \phi) \triangleq \exists\, u_L, u_L' \bullet \mathbf{P}(L \wedge H, S) \wedge \phi \tag{7}$$

(Quantifying over u_L and u_L' hides $\mathcal{L}$'s observational variables, leaving a predicate over $\mathcal{H}$'s observational variables.) From $\mathcal{L}$'s perspective, the $\mathcal{H}$-observations encoded by $\Omega_L^H(S, \phi)$ are *indistinguishable*; that is, $\mathcal{L}$ is unable to identify which of those observations corresponds to $\mathcal{H}$'s actual observation of S.

For S to be considered secure, $\mathcal{L}$ must not be able to deduce confidential information about $\mathcal{H}$ for any observation of S that it can make. Following the MAKS approach [5,6], we encode this requirement in terms of two parameters:

- The *restriction* R is a predicate over $\mathcal{H}$'s observational variables (u_H, u_H') that represents the space of $\mathcal{H}$ observations featuring confidential activities. A $\mathcal{H}$ observation is classed as confidential if and only if it features in R.
- The *closure requirement* Q is a predicate that relates confidential $\mathcal{H}$ observations (in R) to alternative $\mathcal{H}$ observations that are not classed as confidential. Whenever Q relates a confidential activity ψ to a non-confidential activity $\widetilde{\psi}$ such that ψ and $\widetilde{\psi}$ are indistinguishable to $\mathcal{L}$, we say that $\widetilde{\psi}$ represents a *cover story* for ψ. The presence of $\widetilde{\psi}$ in a system ensures that if $\mathcal{H}$ performs ψ, then $\mathcal{L}$ is unable to deduce whether ψ (rather than $\widetilde{\psi}$) has occurred.

The cover stories in Q are encoded over a renaming $(\widetilde{u_H}, \widetilde{u_H'})$ of the variables in u_H and u_H', in order to distinguish between confidential $\mathcal{H}$ activities and non-confidential cover stories. We assume that the domain and co-domain of the relation encoded by Q are disjoint, so that no cover story observations are themselves classed as confidential.

We are now ready to present a formal definition of the space of properties that we call confidentiality properties.

Definition 1 (Confidentiality property). *A confidentiality property is a tuple of the form $\pi = \langle H, L, R, Q \rangle$, where H and L denote the views of $\mathcal{H}$ and $\mathcal{L}$, R is a restriction and Q a closure requirement over H.*

We do not claim that all confidentiality properties described in the literature can be expressed as a $\langle H, L, R, Q \rangle$ tuple (or a combination of such tuples). Nevertheless, Mantel has demonstrated that many confidentiality properties can

indeed be encoded as combinations of pairs of restrictions and closure requirements, assuming a trace-based semantic model for observations [5,6].

Of course, the purchaser of a system may wish to specify particular confidentiality requirements that the system implementation must satisfy. These requirements may be encoded by defining a custom confidentiality property, as we illustrate in the following example.

Example 4. Suppose that the operator of the system G_0 (described in Example 1) insists that $\mathcal{L}$ cannot identify the exact value of n if it guesses incorrectly. We define a confidentiality property $\pi_G = \langle H, L, R_G, Q_G \rangle$ to express this requirement, where R_G and Q_G are as follows:

$$R_G \triangleq g_H \neq n_H \tag{8}$$

$$Q_G \triangleq \widetilde{g_H} = g_H \wedge \widetilde{n_H} \neq n_H \wedge \widetilde{t_H} = t_H \tag{9}$$

R_G indicates the confidentiality property applies only to those system-level observations in G_0 where $\mathcal{L}$'s guess is incorrect. Q_G mandates that, for each such observation, there exists another observation in G_0 with a different value of n but the same values for g and t. Hence, Q_G requires that, when R_G is fulfilled, information giving the exact value of n does not flow to $\mathcal{L}$.

A confidentiality property can be strengthened by weakening R (i.e. making more $\mathcal{H}$ activities confidential) or by strengthening Q (i.e. removing acceptable cover stories). The weakest confidentiality property $\bot$ is obtained when $R =$ **false** (i.e. no $\mathcal{H}$ activities are confidential). Likewise, the strongest confidentiality property $\top$ arises when $R =$ **true** and $Q =$ **false**.

The predicate that relates $\mathcal{H}$ observations of S that are classed as confidential to $\mathcal{L}$ observations of S is given by:

$$\mathbf{P}\,(L \wedge H, S) \wedge R \tag{10}$$

Let ϕ denote a $\mathcal{L}$ observation and ψ denote a confidential $\mathcal{H}$ observation, where ϕ and ψ together satisfy Equation 10. To ensure that $\mathcal{L}$ cannot deduce that ψ has occurred with certainty, Q must relate ψ to a cover story $\widetilde{\psi}$ such that $\widetilde{\psi}$ is a valid $\mathcal{H}$ observation of S consistent with ϕ. Formally:

$$\phi \wedge \psi \Rightarrow \exists\, \widetilde{u_H}, \widetilde{u'_H} \bullet \mathbf{P}\left(L \wedge \widetilde{H}, S\right) \wedge Q \tag{11}$$

$\widetilde{H}$ denotes the view H where each variable in u_H and u'_H is renamed to its counterpart in $\widetilde{u_H}$ and $\widetilde{u'_H}$. Following Mantel [5,6], we combine Equation 10 and Equation 11 to obtain a "schema" for confidentiality properties in the UTP, which we present in Definition 2.

Definition 2 (Confidentiality property schema). *We say a system S* satisfies *a confidentiality property $\pi = \langle H, L, R, Q \rangle$ if and only if $\mathbf{C}(\pi, S)$ holds:*

$$\mathbf{C}(\pi, S) \triangleq \left[\mathbf{P}\,(L \wedge H, S) \wedge R \Rightarrow \left(\exists\, \widetilde{u_H}, \widetilde{u'_H} \bullet \mathbf{P}\left(L \wedge \widetilde{H}, S\right) \wedge Q\right)\right] \tag{12}$$

Hence, a system satisfies a confidentiality property if and only if the system permits only those information flows from $\mathcal{H}$ to $\mathcal{L}$ which are acceptable to the property. It follows from Definition 2 that the weakest property $\bot$ is satisfied by all systems, but that no implementable system specification can satisfy $\top$.

Example 5. We can determine whether G_0 satisfies π_G by calculating the constituent predicates of $\mathbf{C}$ corresponding to R_G and Q_G:

$$\mathbf{P}\,(L \wedge H, G_0) \wedge R_G \;=\; \begin{pmatrix} n_H \in 0..9 \wedge g_L = g_H \wedge t_L = t_H \\ \wedge\; g_H \neq n_H \wedge (g_H > n_H \Rightarrow t_L > 0) \\ \wedge\; (g_H < n_H \Rightarrow t_L < 0) \end{pmatrix} \tag{13}$$

$$\begin{array}{l} \exists\, \widetilde{u_H}, \widetilde{u'_H} \bullet \\ \mathbf{P}\left(L \wedge \widetilde{H}, G_0\right) \wedge Q_G \end{array} \;=\; \begin{pmatrix} g_L = g_H \wedge t_L = t_H \\ \wedge\; \exists\, \widetilde{n_H} \bullet \begin{pmatrix} \widetilde{n_H} \in 0..9 \wedge \widetilde{n_H} \neq n_H \\ \wedge\; (g_H = \widetilde{n_H} \Rightarrow t_L = 0) \\ \wedge\; (g_H > \widetilde{n_H} \Rightarrow t_L > 0) \\ \wedge\; (g_H < \widetilde{n_H} \Rightarrow t_L < 0) \end{pmatrix} \end{pmatrix} \tag{14}$$

When $g_L = g_H = 1 \wedge n_H = 0 \wedge t_H = t_L \wedge t_H > 0$, Equation 13 is satisfied but Equation 14 is not. In this scenario, $\mathcal{L}$ observes (guesses) $g_L = 1$ and learns from t_L that $g_L > n$. Since $n_H \in 0..9$, $\mathcal{L}$ can deduce from its observation of g_L and t_L that $n_H = 0$. (A similar situation arises when $n_H = 9$ and $g_L = 8$.) It follows that Equation 13 does not imply Equation 14 in all circumstances, and so we conclude that G_0 does not satisfy π_G.

The discrepancy between G_0 and π_G may be resolved by relaxing R_G to exclude system-level observations where $n_H = 0$ or $n_H = 9$. Hence, we define a weaker confidentiality property $\pi'_G = \langle H, L, R'_G, Q_G \rangle$, where:

$$R'_G \;\triangleq\; g_H \neq n_H \wedge n_H \neq 0 \wedge n_H \neq 9 \tag{15}$$

Of course, the discrepancy could be resolved by weakening G_0: for instance, by using t to indicate only whether the guess was correct or incorrect.

This example highlights an important trade-off between functionality and security requirements in software development. While confidentiality properties place an upper bound on information flow to $\mathcal{L}$, functionality requirements may be interpreted as placing a *lower* bound on information flow to $\mathcal{L}$. Should these bounds conflict (as in our example), then no system could ever satisfy both kinds of requirements and so these requirements should be re-evaluated before proceeding with the design of a system.

5 Confidentiality-Preserving Refinement

A system specification is *refined* (improved) by adding implementation details to the specification to remove non-determinism. In the UTP, the classical notion of refinement is characterised by implication between predicates: [4]

$$S \sqsubseteq T \;\triangleq\; [\, T \Rightarrow S \,] \tag{16}$$

$S \sqsubseteq T$ holds if and only if every system-level observation of T is a (possible) system-level observation of S. A significant feature of the UTP is that the notion of refinement is the same across the various UTP theories [4].

Refinement guarantees that a program satisfies all the functionality properties in its specification. Unfortunately, the $\sqsubseteq$ relation is not strong enough to preserve confidentiality properties in specifications, since it may introduce new conduits of information flow to low-level users [8,9]. This result can be established in our framework by appealing to the following lemma (from [7]).

Lemma 1 (P is order-preserving). *If $S \sqsubseteq T$ holds, then every pair of $\mathcal{L}$ and $\mathcal{H}$ observations of T must correspond to a pair of $\mathcal{L}$ and $\mathcal{H}$ observations of S:*

$$S \sqsubseteq T \Rightarrow \mathbf{P}(L \wedge H, S) \sqsubseteq \mathbf{P}(L \wedge H, T) \tag{17}$$

Lemma 1 indicates that whenever $S \sqsubseteq T$ holds, then for each $\mathcal{L}$ observation ϕ of T, the set of $\mathcal{H}$ observations of T consistent with ϕ is a subset of the $\mathcal{H}$ observations of S consistent with ϕ. Hence, $\mathcal{L}$ can deduce at least as much knowledge of $\mathcal{H}$'s activities by observing T as it can deduce by observing S. In turn, T may violate confidentiality properties that S satisfies, if T does not provide all of the cover stories that S provides. In the absence of these cover stories from T, $\mathcal{L}$ can deduce extra knowledge about $\mathcal{H}$'s activities that may enable it to establish that a certain confidential $\mathcal{H}$ activity has taken place. Hence, $S \sqsubseteq T$ is not sufficient to guarantee that T satisfies any given confidentiality property that S satisfies.

Example 6. The guessing game of Example 1 may be implemented as follows:

$$G_1 \triangleq n \in 0..9 \wedge t = g - n \tag{18}$$

While $G_0 \sqsubseteq G_1$ holds, it is clear that G_1 is less secure than G_0, since the values of g and t allow $\mathcal{L}$ to deduce the exact value of n in G_1. Indeed, G_1 violates the confidentiality property π'_G from Example 5, whereas G_0 satisfies this property.

When a system specification incorporates confidentiality properties, it is desirable for refinement steps to uphold these properties, to save the effort of re-verifying each confidentiality property after each refinement step and the expense of carrying out refinement steps that reach insecure system designs. In this section, we demonstrate how notions of refinement can be strengthened to preserve confidentiality properties of the form $\pi = \langle H, L, R, Q \rangle$.

A *confidentiality ordering* is a pre-order over the space of systems [10]. We define a confidentiality ordering $\preceq_\pi$ — parameterised by the encoding of confidentiality properties presented in Section 4 — to relate predicates according to what $\mathcal{L}$ can deduce about $\mathcal{H}$'s confidential activities as specified by π. [2]

[2] The confidentiality ordering presented here is conceptually related to Jacob's security ordering [7,10]. However, the orderings are different in their details: the security ordering is concerned only with what $\mathcal{L}$ can deduce about $\mathcal{H}$'s activities and does not discriminate between confidential and non-confidential activities.

Definition 3 (Confidentiality ordering). *Let $\pi = \langle H, L, R, Q\rangle$. T provides no more information about $\mathcal{H}$-observations in R to $\mathcal{L}$ than does S if and only if:*

$$S \preceq_\pi T \triangleq S \lesssim_\pi T \wedge S \trianglelefteq_\pi T \tag{19}$$

holds, where $\lesssim_\pi$ and $\trianglelefteq_\pi$ are defined as follows:

$$S \lesssim_\pi T \triangleq \mathbf{P}(L \wedge H, S) \sqsubseteq \mathbf{P}(L \wedge H, T) \wedge R \tag{20}$$

$$S \trianglelefteq_\pi T \triangleq \mathbf{P}\left(L \wedge \widetilde{H}, T\right) \sqsubseteq \mathbf{P}(L \wedge H, T) \wedge R \wedge \mathbf{P}\left(L \wedge \widetilde{H}, S\right) \wedge Q \tag{21}$$

$S \lesssim_\pi T$ mandates that every confidential $\mathcal{H}$ observation ψ present in T must also be present in S and, moreover, all $\mathcal{L}$ observations of T consistent with ψ must also be present in S. $S \trianglelefteq_\pi T$ requires that, for every $\mathcal{H}$ activity ψ in T classed as confidential by R, all of the cover stories related to ψ by Q that are present in S must also be present in T. When both of these conditions are satisfied, T provides no more information flow about $\mathcal{H}$'s confidential activities to $\mathcal{L}$ than S; in other words, $\mathcal{L}$ can deduce no more confidential information from any observation of T than it can deduce by observing S.

(Note that T may provide more information flow from $\mathcal{H}$ to $\mathcal{L}$ than S without violating π, so long as for each confidential $\mathcal{H}$ observation ψ in T, there is at least one cover story compatible with ψ also in T.)

Lemma 2 formalises the relationship between confidentiality properties and the confidentiality ordering.

Lemma 2 (π is closed under $\preceq_\pi$). *Whenever S satisfies the confidentiality property π and $S \preceq_\pi T$ holds, then T also satisfies π.*

It follows from Lemma 2 that a refinement relation that preserves π may be constructed as the least upper bound of the $\sqsubseteq$ and $\preceq_\pi$ orderings:

$$S \sqsubseteq^{cp}_\pi T \triangleq S \sqsubseteq T \wedge S \preceq_\pi T \tag{22}$$

Since $S \sqsubseteq T$ implies $S \lesssim_\pi T$ (by Lemma 1), it is sufficient to prove that $S \sqsubseteq T$ and $S \trianglelefteq_\pi T$ hold in order to establish that $S \preceq_\pi T$ holds. Hence, we can simplify the definition of $\sqsubseteq^{cp}_\pi$ as follows:

$$S \sqsubseteq^{cp}_\pi T = S \sqsubseteq T \wedge S \trianglelefteq_\pi T \tag{23}$$

Corollary 1 ($\sqsubseteq^{cp}$ preserves confidentiality properties). *Lemma 2 implies that, if $S \sqsubseteq^{cp}_\pi T$ holds and S satisfies π, then T also satisfies π.*

Example 7. An alternative implementation of the guessing game is:

$$G_2 \triangleq n \in 0..9 \wedge (g = n \Rightarrow t = 0) \wedge (g > n \Rightarrow t = 1) \wedge (g < n \Rightarrow t = -1) \tag{24}$$

We have $G_0 \sqsubseteq G_2$ and $G_0 \preceq_{\pi'_G} G_2$, so it follows that $G_0 \sqsubseteq^{cp}_{\pi'_G} G_2$. From Corollary 1, we can conclude that G_2 preserves π'_G.

Lemma 3 (Ordering of $\preceq_\pi$ and $\sqsubseteq$). *The $\preceq_\pi$ and $\sqsubseteq$ orderings are neither monotonic nor anti-monotonic w.r.t. each other.*

Lemma 3 suggests that, during a stepwise system development that employs the $\sqsubseteq_\pi^{cp}$ refinement relation, one may reach a system design where no useful refinement steps can be made without violating the $\preceq_\pi$ ordering (for instance, by removing cover stories from the design without also removing all confidential observations related to those cover stories). Since no progress towards a correct implementation of the specification can be made from such a design, the system's designers must either weaken π, or backtrack to an earlier design in their development and try a different series of refinement steps. This potential for costly backtracking suggests that the $\sqsubseteq_\pi^{cp}$ refinement relation may be impractical for developing software by stepwise refinement.

It is usually assumed that all behaviours of a system are distinguishable by its environment; hence, no new behaviours can be added to a system design without violating the system's specification. However, if a system is known to operate in an environment consisting of multiple users (each with limited observational abilities) then we may relax the definition of refinement by allowing certain system-level observations to be *added* to a system design without compromising its functionality, so long as these observations do not induce new interface-level observations. Given a set of interface-disjoint views $\mathcal{W}$ representing a group of users, we say T is a *co-operating refinement* of S (w.r.t. $\mathcal{W}$) [11] if, for all sets of observations of T that can be made through the views in $\mathcal{W}$, these observations can be combined in order to reconstruct at least one system-level observation of T that is present in S. Intuitively, co-operating refinement ensures the users represented by $\mathcal{W}$ are collectively unable to detect that T possesses any behaviour not present in S.

Co-operating refinement can be expressed in the UTP in terms of $\sqsubseteq$ and the $\mathbf{P}$ predicate transformer, as shown in Definition 4 [7].

Definition 4 (Co-operating refinement). *T is a co-operating refinement of S w.r.t. a set of views $\mathcal{W}$ if and only if, for every system-level observation Φ of T, the projections of Φ through each view in $\mathcal{W}$ are matched by the projections of a system-level observation of S through each view:*

$$S \sqsubseteq_{\mathcal{W}}^{co} T \triangleq \mathbf{P}\left(\bigwedge \mathcal{W}, S\right) \sqsubseteq \mathbf{P}\left(\bigwedge \mathcal{W}, T\right) \tag{25}$$

Corollary 2 ($\sqsubseteq_{\mathcal{W}}^{co}$ is weaker than $\sqsubseteq$). *It follows from Lemma 1 that, for all sets $\mathcal{W}$ of interface-disjoint views, $S \sqsubseteq T$ guarantees $S \sqsubseteq_{\mathcal{W}}^{co} T$ [7,11].*

While co-operating refinement is not as strong as classical refinement, it is strong enough to preserve the inherent functionality in a system's specification from the perspective of the system's groups of users. Furthermore, co-operating refinement provides the designers of a system with extra flexibility to execute some kinds of useful refinement steps that are not permitted by classical refinement, such as distributing a system across independent processors [7,11]. We postulate that this ability to add new behaviours to system designs (in a controlled

manner) can help to overcome the difficulties encountered when developing a system to satisfy a given confidentiality property by stepwise refinement.

When defining a co-operating refinement relation for a group of users, we assume that users in the group can communicate with each other (but not with users outside the group). It follows that, by exchanging their observations, a group of low-level users may therefore be able to deduce more information about high-level activities than they could deduce from their individual observations alone. Hence, when reasoning about confidentiality properties in the setting of co-operating refinement, we represent a group of low-level users $\mathcal{LS}$ as a single user with observational abilities equivalent to those of all the users in $\mathcal{LS}$ combined.

Suppose that Z is a system that interacts with a single high-level user (with view H_Z) and a group of communicating low-level users associated with a set LS_Z of (interface-disjoint) views. Using co-operating refinement in place of $\sqsubseteq$, we can obtain a refinement relation that preserves the confidentiality property $\pi_Z = \langle H_Z, \bigwedge LS_Z, R_Z, Q_Z \rangle$ for this system:

$$S \sqsubseteq_{\pi_Z}^{co/cp} T \triangleq S \sqsubseteq_{\{H_Z\}}^{co} T \wedge S \sqsubseteq_{LS_Z}^{co} T \wedge S \preceq_{\pi_Z} T \tag{26}$$

A potential advantage of $\sqsubseteq_{\pi}^{co/cp}$ over $\sqsubseteq_{\pi}^{cp}$ is that all the terms of $\sqsubseteq_{\pi}^{co/cp}$ are expressed with the **P** predicate transformer, which may simplify the task of verifying that $\sqsubseteq_{\pi}^{co/cp}$ holds between system designs. It is also unnecessary to redefine the R and Q components of π when dealing with multiple low-level users, since R and Q refer exclusively to high-level observations.

Corollary 3 ($\sqsubseteq^{co/cp}$ preserves confidentiality properties). *Whenever S satisfies the confidentiality property $\pi = \langle H, \bigwedge LS, R, Q \rangle$ and $S \sqsubseteq_{\pi}^{co/cp} T$ holds, then T also satisfies π.*

A family of confidentiality-preserving refinement relations can be constructed from $\sqsubseteq^{co/cp}$. If a system is required to satisfy a set of confidentiality properties Π, then a suitable refinement relation for that system can be constructed by taking the least upper bound of the $\sqsubseteq_{\pi}^{co/cp}$ relations for each $\pi \in \Pi$. Furthermore, if the users of a system are partitioned into a set of isolated groups — such that high-level users are not placed in the same group as low-level users — then a suitable refinement relation can be calculated by combining $\sqsubseteq^{co/cp}$ relations for each group and each confidentiality property that the system must satisfy.

6 Related Work

Frameworks. Several semantic frameworks (including the MAKS) for expressing a range of confidentiality properties in a uniform manner have been proposed in the security literature [5,6,9,10,12,13,14]. The objective of these frameworks is to consolidate the existing definitions of noninterference-like properties in the literature, in order to evaluate and compare these properties systematically and to enable new confidentiality properties to be defined rigorously.

In these frameworks, a system's semantics is taken to be a set of traces. This emphasis on a trace semantics contrasts with the UTP approach, in which systems are represented by predicates within a UTP theory. By abstracting away from a trace semantics, our framework avoids the need to translate a system specification (with a UTP semantics) to a set of traces. However, our framework does not differentiate between inputs and outputs of systems, which is a prerequisite for expressing many noninterference-like properties. In order to capture such properties, it would be necessary to extend our framework with notation for modelling inputs and outputs separately or, alternatively, to interpret undashed UTP variables as inputs and dashed variables as outputs.

In contrast to noninterference-like properties, we do not regard all $\mathcal{H}$ activities as (equally) confidential. We also expect that $\mathcal{L}$ can deduce some aspects of $\mathcal{H}$'s observations legitimately and seek only to constrain $\mathcal{L}$ from deducing the occurrence of a subset of $\mathcal{H}$ activities. Moreover, we argue that confidentiality properties weaker than noninterference fit more closely with software development in practice. For example, noninterference from $\mathcal{H}$ to $\mathcal{L}$ is too strong a requirement for the guessing game, where the functionality of the game dictates that $\mathcal{L}$'s observation of the outcome of its guess must be influenced by $\mathcal{H}$.

To date, the most complete framework for expressing noninterference-like confidentiality properties is Mantel's MAKS [5,6]. The MAKS defines a collection of "basic security predicates" (BSPs) to express a variety of transformations on the high-level components of system traces, capturing low-level users' uncertainty about high-level inputs and outputs. These BSPs can be combined to express many (but not all) of the noninterference-like confidentiality properties defined in the literature. Since our definition of confidentiality properties is based on the schema form of these BSPs, we may select predicates to model restrictions and closure requirements encoding the range of BSPs within a UTP theory with a trace semantics. This implies that existing noninterference-like properties that are expressible in the MAKS can be re-expressed using our framework within a UTP theory that distinguishes input events from output events.

Refinement. Most notions of confidentiality-preserving refinement in the literature are realised in two ways: by limiting the space of confidentiality properties to those that are closed under classical refinement, or by strengthening the refinement relation to ensure it preserves confidentiality properties in specifications. To ensure that confidentiality properties are not unreasonably strong, the first approach often necessitates a means for distinguishing between two kinds of non-determinism — namely, under-specification (which can be removed safely) and unpredictability (which restricts information flow from $\mathcal{H}$ to $\mathcal{L}$ and so should not be removed) — in specifications [15]. However, many specification languages (and UTP theories) lack facilities for modelling non-determinism intended to provide unpredictability separately from under-specification. Thus, it is unclear how the first approach can be applied to a UTP theory without extending the semantics of that theory to distinguish between these kinds of non-determinism explicitly. We have instead focused on the second approach for confidentiality-preserving

refinement by applying the confidentiality ordering, to avoid specialising our framework to particular UTP theories.

Seehusen and Stølen [13,14] have proposed a (MAKS-like) framework for integrating confidentiality properties into stepwise software development processes. In their framework, a system specification is formalised as a set of *obligations* (trace sets); a system satisfies a specification if it contains at least one trace from each obligation in the specification. Intuitively, the obligations ensure that a system's implementation provides a minimum level of unpredictability about the system's behaviour from a low-level perspective. Hence, obligations are closely related to our notion of cover stories. Seehusen and Stølen use obligations to facilitate a novel approach to confidentiality-preserving refinement, which parallels our own: while refinement may remove some traces from a system design, the refined system must still fulfil each obligation in the specification.

A recent and significant development is Morgan's *shadow semantics* [16,17], which builds on the refinement calculus for sequential programs [18]. In this semantics, $\mathcal{L}$ (the adversary) is assumed to be able to monitor how demonic non-determinism is resolved at each program step. This novel device allows the "shadow set" of possible values of a high-level variable that are consistent with $\mathcal{L}$'s observations to itself be modelled at the semantic level. A program is secure if the shadow set never reveals the value of the high-level variable to $\mathcal{L}$ (even when $\mathcal{L}$ monitors the control flow); and refinement is "ignorance-preserving" if it does not decrease the shadow set associated with any confidential variable. This notion of refinement (like our own) ensures that $\mathcal{L}$ can never deduce any more information regarding confidential data from an implementation of a system as it could from the corresponding specification.

Our approach towards confidentiality-preserving refinement also has links to work by Alur et al. [19]. In this work, the refinement of a labelled transition system is expected to preserve $\mathcal{L}$'s inability to deduce whether system runs satisfy a specified set of properties on secret variables. Alur et al. also describe how this refinement relation maps to standard simulation-based proof techniques, which (we conjecture) may be useful for discharging the proof obligations associated with our own confidentiality-preserving refinement relations.

Compositionality. An important topic in the study of information flow security is the compositionality of system designs, as a means of developing secure systems in a modular fashion. Given a collection of sub-systems $S_1, \ldots, S_N$ which individually satisfy a confidentiality property π, it is desirable that the system obtained by composing $S_1, \ldots, S_N$ together will also satisfy π. However, this is not generally the case for operators such as parallel composition, because the composite system may feature confidential activities but lack the cover stories that are permitted by its components. Mantel has identified a collection of formal conditions for which the BSPs of the MAKS are preserved under certain compositional operators [6,20]. These conditions may provide a starting point for identifying compositionality conditions for our UTP formulation of confidentiality properties.

A further topic for investigation is the compositionality of confidentiality-preserving refinement relations. This topic has not been addressed in the MAKS, but has been investigated elsewhere [21].

Limitations. Like the BSPs of the MAKS, our formulation of confidentiality properties is qualitative w.r.t. information flow. This means that a confidentiality property is violated if even a single bit of information relating to confidential data is disclosed to a low-level user. Small leaks of data from high-level users to low-level users are often acceptable (and sometimes unavoidable) in real-world systems, provided that a low-level user cannot deduce any significant details about confidential data from such a leak. In these circumstances, it may be difficult to work with qualitative confidentiality properties. This difficulty can be overcome by defining confidentiality properties in terms of how much information can flow between users. Recent research has modelled quantitative information flow within the framework of Shannon-style information theory [22].

Another limitation of our treatment of confidentiality properties — which is shared by the MAKS — is that we do not address the probability distribution of the high-level user's activities. This shortcoming could lead to serious security breaches, because if a low-level user has knowledge of this distribution, then it may be able to deduce confidential data with near certainty but without violating the confidentiality property. Some research has investigated *probabilistic* confidentiality properties which account for the likelihood of alternative high-level activities [23,24]. While these properties are attractive in theory, their application in practice is not without difficulty: the probability distribution of the high-level user's interactions with a system may be unknown and, even with this knowledge, it may be intractable to determine whether a non-trivial system model satisfies a probabilistic confidentiality property [25].

A further practical issue with existing frameworks for modelling confidentiality properties (including our own) is that, while they are suitable for reasoning about information flow at the abstract level of system specifications, they do not account for the (potentially confidential) information that a low-level user may deduce by monitoring physical and temporal characteristics of its interface at the implementation level, such as the time interval between requests and responses. These channels may potentially be exploited to obtain confidential information about a system's behaviour, so it is desirable to model these channels formally and to extend the analysis of information flow to them. We regard this as a challenging task that is beyond the scope of this paper.

7 Conclusions

The original contributions of this work are two-fold. First, we have laid the foundations for reasoning about information flow in the UTP and provided a UTP encoding of "possibilistic" confidentiality properties. Second, we have identified a spectrum of confidentiality-preserving refinement relations based on the confidentiality ordering and co-operating refinement.

Our UTP formulation of confidentiality properties in the predicative style is concise and generic in the UTP theory under consideration, yet also provides expressive power equivalent to the MAKS framework for specifying confidentiality properties. Moreover, our approach can be deployed across the various families of specification languages and programming paradigms that have a UTP semantics, rather than just those based on trace semantics.

A drawback of our formulation of confidentiality properties in the UTP is that verifying UTP specifications against confidentiality properties can be cumbersome, due to the need to translate from system-level to interface-level predicates. Indeed, errors can be easily introduced into these calculations if they are carried out manually, and especially when the mapping from system-level to interface-level observations is not straightforward. This difficulty could be alleviated by identifying a collection of laws that can be used to simplify these calculations.

In future work, we intend to integrate our formulation of confidentiality properties with the UTP semantics of *Circus* [26] — a specification language which combines Z and CSP to model the state and behavioural aspects of concurrent systems — in order to realise a unified framework for the specification and development of secure software. Indeed, we envisage that a customer's specification of confidentiality properties can play an integral role in a formal stepwise development process — from constructing an abstract system design that is verified to satisfy the confidentiality specification, through a series of confidentiality-preserving refinement steps — to yield a final implementation that is guaranteed to satisfy the customer's security requirements.

Acknowledgements

Michael Banks is supported by a UK Engineering and Physical Sciences Research Council DTA studentship. Thanks to Ana Cavalcanti for suggestions on improving the structure of this paper; to the anonymous referees for identifying omissions in an earlier draft; and to Chris Poskitt for proofreading.

References

1. Denning, D.E.: Cryptography and Data Security. Addison-Wesley Longman Publishing Company, Inc., Boston, MA, USA (1982)
2. Lampson, B.W.: A note on the confinement problem. Communications of the ACM **16**(10) (October 1973) 613–615
3. Goguen, J.A., Meseguer, J.: Security policies and security models. In: Proceedings of the 1982 IEEE Symposium on Security and Privacy, IEEE Computer Society (April 1982) 11–20
4. Hoare, C.A.R., He, J.: Unifying Theories of Programming. Prentice Hall International Series in Computer Science. Prentice Hall Inc. (1998)
5. Mantel, H.: Possibilistic definitions of security — an assembly kit. In: 13th IEEE Computer Security Foundations Workshop (CSFW '00). (2000) 185–199
6. Mantel, H.: A Uniform Framework for the Formal Specification and Verification of Information Flow Security. PhD thesis, Universität Saarbrücken (July 2003)

7. Banks, M.J., Jacob, J.L.: On modelling user observations in the UTP. In Qin, S., ed.: 3rd International Symposium on Unifying Theories of Programming (UTP 2010). Lecture Notes in Computer Science, Springer Berlin / Heidelberg (2010)
8. 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
9. McLean, J.: A general theory of composition for trace sets closed under selective interleaving functions. In: Proceedings of the 1994 IEEE Symposium on Security and Privacy. (1994) 79–93
10. Jacob, J.L.: Security specifications. In: Proceedings of the 1988 IEEE Symposium on Security and Privacy. (1988) 14–23
11. Jacob, J.L.: Refinement of shared systems. In McDermid, J.A., ed.: The Theory and Practice of Refinement: Approaches to the Development of Large-Scale Software Systems. Butterworths (1989) 27–36
12. Focardi, R., Gorrieri, R.: A taxonomy of security properties for process algebras. Journal of Computer Security **3**(1) (1995) 5–34
13. Seehusen, F., Stølen, K.: Maintaining information flow security under refinement and transformation. In: Formal Aspects in Security and Trust. Volume 4691 of Lecture Notes in Computer Science., Springer Berlin / Heidelberg (2007) 143–157
14. Seehusen, F., Stølen, K.: Information flow security, abstraction and composition. IET Information Security **3**(1) (2009) 9–33
15. Roscoe, A.W.: CSP and determinism in security modelling. In: Proceedings of the 1995 IEEE Symposium on Security and Privacy, IEEE Computer Society (1995) 114–127
16. Morgan, C.: The shadow knows: Refinement and security in sequential programs. Science of Computer Programming **74**(8) (June 2009) 629–653
17. Morgan, C.: How to brew-up a refinement ordering. Electronic Notes in Theoretical Computer Science **259** (December 2009) 123–141
18. Morgan, C.: Programming from Specifications. Second edn. Prentice Hall International Series in Computer Science. Prentice Hall Inc., Hertfordshire, UK (1994)
19. Alur, R., Černý, P., Zdancewic, S.: Preserving secrecy under refinement. In: Automata, Languages and Programming. Volume 4052 of Lecture Notes in Computer Science., Springer Berlin / Heidelberg (2006) 107–118
20. Mantel, H.: On the composition of secure systems. In: Proceedings of the 2002 IEEE Symposium on Security and Privacy. (2002) 88–101
21. Santen, T., Heisel, M., Pfitzmann, A.: Confidentiality-preserving refinement is compositional – sometimes. In Gollmann, D., Karjoth, G., Waidner, M., eds.: Computer Security – ESORICS 2002. Volume 2502 of Lecture Notes in Computer Science., Springer Berlin / Heidelberg (2002) 194–211
22. Smith, G.: On the foundations of quantitative information flow. In Alfaro, L., ed.: Foundations of Software Science and Computational Structures. Volume 5504 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (2009) 288–302
23. Santen, T.: A formal framework for confidentiality-preserving refinement. In: Computer Security – ESORICS 2006. Volume 4189 of Lecture Notes in Computer Science., Springer Berlin / Heidelberg (2006) 225–242
24. Santen, T.: Preservation of probabilistic information flow under refinement. Information and Computation **206**(2-4) (February 2008) 213–249
25. 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
26. Oliveira, M., Cavalcanti, A., Woodcock, J.: A UTP semantics for Circus. Formal Aspects of Computing **21**(1) (February 2009) 3–32

A Proofs

Lemma 1

Proof. A proof of this lemma is provided elsewhere [7].

Lemma 2

Proof.

$$
\begin{array}{ll}
& S \preceq_\pi T \wedge \mathbf{C}(\pi, S) \\
\Leftrightarrow & \quad \langle \text{ definition of } \preceq \text{ and } \mathbf{C} \ \rangle \\
& S \lesssim_\pi T \wedge S \trianglelefteq_\pi T \wedge \left[\begin{array}{l} \mathbf{P}\,(L \wedge H, S) \wedge R \\ \Rightarrow \left(\exists\, \widetilde{u_H}, \widetilde{u'_H} \bullet \mathbf{P}\left(L \wedge \widetilde{H}, S \right) \wedge Q \right) \end{array} \right] \\
\Rightarrow & \quad \langle \text{ definition of } \lesssim \text{ and transitivity of implication } \rangle \\
& S \trianglelefteq_\pi T \wedge \left[\mathbf{P}\,(L \wedge H, T) \wedge R \Rightarrow \left(\exists\, \widetilde{u_H}, \widetilde{u'_H} \bullet \mathbf{P}\left(L \wedge \widetilde{H}, S \right) \wedge Q \right) \right] \\
\Leftrightarrow & \quad \langle \text{ definition of } \trianglelefteq \text{ and predicate calculus } \rangle \\
& \left[\begin{array}{c} \mathbf{P}\,(L \wedge H, T) \wedge R \Rightarrow \left(\exists\, \widetilde{u_H}, \widetilde{u'_H} \bullet \mathbf{P}\left(L \wedge \widetilde{H}, S \right) \wedge Q \right) \\ \wedge \left(\mathbf{P}\,(L \wedge H, T) \wedge R \wedge \mathbf{P}\left(L \wedge \widetilde{H}, S \right) \wedge Q \right) \Rightarrow \mathbf{P}\left(L \wedge \widetilde{H}, T \right) \end{array} \right] \\
\Rightarrow & \quad \langle \text{ predicate calculus } \rangle \\
& \left[\mathbf{P}\,(L \wedge H, T) \wedge R \Rightarrow \left(\exists\, \widetilde{u_H}, \widetilde{u'_H} \bullet \mathbf{P}\left(L \wedge \widetilde{H}, T \right) \wedge Q \right) \right] \\
\Leftrightarrow & \quad \langle \text{ definition of } \mathbf{C} \ \rangle \\
& \mathbf{C}(\pi, T) \qquad \square
\end{array}
$$

Lemma 3

Proof. First, the proof that $\lesssim_\pi$ is monotonic w.r.t. $\sqsubseteq$ follows from Lemma 1. Second, we show that $\trianglelefteq_\pi$ is anti-monotonic w.r.t. $\sqsubseteq$.

$$
\begin{array}{ll}
& S \sqsubseteq T \\
\Rightarrow & \quad \langle \text{ by Lemma 1 } \rangle \\
& \mathbf{P}\left(L \wedge \widetilde{H}, S \right) \sqsubseteq \mathbf{P}\left(L \wedge \widetilde{H}, T \right) \\
\Rightarrow & \quad \langle \text{ predicate calculus } \rangle \\
& \mathbf{P}\left(L \wedge \widetilde{H}, S \right) \sqsubseteq \left(\mathbf{P}\left(L \wedge \widetilde{H}, T \right) \wedge R \wedge Q \wedge \mathbf{P}\,(L \wedge H, S) \right) \\
\Leftrightarrow & \quad \langle \text{ definition of } \trianglelefteq_\pi \ \rangle \\
& T \trianglelefteq_\pi S
\end{array}
$$

These two results imply that the relation formed by combining $\lesssim_\pi$ and $\trianglelefteq_\pi$ — namely, $\preceq_\pi$ — is neither monotonic nor anti-monotonic w.r.t. $\sqsubseteq$. $\square$