The Shadow Knows:

Refinement of ignorance in sequential programs

Carroll Morgan

Dept. Comp. Sci. & Eng., Uni. NSW, Sydney 2052 Australia
carrollm@cse.unsw.edu.au

Abstract. Separating sequential-program state into "visible-" and "hidden" parts facilitates reasoning about knowledge, security and privacy: applications include zero-knowledge protocols, and security contexts with hidden "high-security" state and visible "low-security" state. A rigorous definition of how specifications relate to implementations, as part of that reasoning, must ensure that implementations reveal no more than their specifications: they must, in effect, preserve ignorance.

We propose just such a definition –a relation of *ignorance-preserving refinement*– between specifications and implementations of sequential programs. Its purpose is to enable a development-by-refinement methodology for applications like those above.

Since preserving ignorance is an extra obligation, the proposed refinement relation restricts (rather than extends) the usual. We suggest general principles for restriction, and we give specific examples of them.

To argue that we do not restrict too much –for "no refinements allowed at all" is trivially ignorance-preserving– we derive The Dining Cryptographers protocol via a program algebra based on the restricted refinement relation. It is also a motivating case study, as it has never before (we believe) been treated refinement-algebraically.

In passing, we discuss –and solve– the *Refinement Paradox*.

1 Introduction

Refinement as a relation between sequential programs is based traditionally on a state-to-state operational model, with a corresponding logic of Hoare-triples $\{\Phi\}$ S $\{\Psi\}$ [1] or equivalently weakest preconditions $wp.\mathrm{S}.\Psi$ [2], and it generates an algebra of (in-)equations between program fragments [3, 4]. A specification S1 is said to be *refined by* an implementation S2, written S1 $\sqsubseteq$ S2, just when S2 preserves all logically-expressible properties of S1.

Ignorance is (for us) what an observer doesn't know about the parts of the program state he can't see. If we partition the state into a "visible" part v and a "hidden" part h, and we consider a known program operating over v, h, then we can ask "from the final value of v, what can the observer deduce about the final value of h?" If the program is $v{:=}\,0$, what he knows afterwards about h is just

what he knew beforehand; if it is $v{:=}\,h \bmod 2$, then he has learned h's parity; and if it is $v{:=}\,h$ then he has learned h's value exactly.

Traditional refinement does not preserve ignorance. If we assume v, h both to have type T, then "choose v from T" is refinable into "set v to h" — it is simply a reduction of demonic nondeterminism. But that refinement, which we write $v{:\in}\,T \sqsubseteq v{:=}\,h$, is called the "Refinement Paradox" (Sec. 6) precisely because it does not preserve ignorance: program $v{:\in}\,T$ tells us nothing about h, whereas $v{:=}\,h$ tells us everything [5]. *Thus we cannot use traditional refinement "as is" for ignorance-preservation.* We must alter it.

Our first contribution is to propose the following principles that should apply to a *refinement algebra* altered to respect ignorance-preservation:

- Pr1 *All traditional "visible-only" refinements are retained* — It would be impractical to search an entire program for hidden variables in order to validate local *visible-only* reasoning in which the hiddens are not even mentioned.
- Pr2 *All traditional "structural" refinements are retained* — Associativity of sequential composition, distribution of code into branches of a conditional *etc.* are refinements (actually equalities) that do not depend on the actual code fragments affected: they are *structurally* valid, acting *en bloc*. It would be impractical to have to check through the fragments' interiors (including *e.g.* procedure calls) to validate such familiar rearrangements.
- Pr3 *Some traditional "explicit-hidden" refinements are excluded* — Those that preserve ignorance will be retained; the others (*e.g.* the Paradox) will be excluded. For this principle we need a model and a logic.

Our second, and main contribution (Secs. 3–5) is to extend the model and logic of sequential programming (only slightly) to realise the above principles: existing visible-only and structural refinements will all remain sound (Pr1,Pr2); and explicit-hidden (putative) refinements can be checked individually (Pr3) for exclusion (*e.g.* Sec. 6) or retention (*e.g.* Sec. 7).

Ignorance-preserving refinement should be of great utility for developing zero-knowledge- or security-sensitive protocols (at least); and our final contribution (Secs. 7,8) is therefore a case study, the Dining Cryptographer's protocol [6], which will bolster our confidence that Pr3 has not excluded too much.

Sections 2 and 9 are informal, discussions of motivations and of comparisons and conclusions respectively.

2 Realising the refinement-algebra of ignorance

2.1 Guiding intuitions

The great advantage of having our goals expressed as algebraic principles is that we can conduct early (and intellectually inexpensive) *gedanken* experiments that will inform the subsequent construction of our model and logic. For example...

Does program $v{:=}\,h; v{:=}\,0$ *reveal* h*?* Yes, it does, because –first– sequential composition ";" remains associative (from Pr2); and $v{:=}\,0; v{:\in}\,T = v{:\in}\,T$ (Pr1);

then $v{:\in}\, T \sqsubseteq \mathbf{skip}$ (Pr1), with sequential composition retaining $\sqsubseteq$-monotonicity (Pr2); and finally **skip** is still the identity (Pr2). Thus we can reason

$$(v{:=}\, h; v{:=}\, 0); v{:\in}\, T \quad = \quad v{:=}\, h; v{:\in}\, T; \quad \sqsubseteq \quad v{:=}\, h; \mathbf{skip} \quad = \quad v{:=}\, h \ ;$$

since the implementation (*rhs*) fails to conceal h, so must the specification (*lhs*) have failed also. Hence our model must have "perfect recall" [7], because escape of h into v is not "erased" by the v-overwriting $v{:=}\, 0$ — and that is what allows h to be "copied" by the final $v{:\in}\, T$, our algebraic means of detecting the leak.

Arguments like the above –as well as advice[1]– suggest the Logic of Knowledge [8] as a suitable basis (App. A). Here we give the intuitions that basis supplies.

The observed program includes a notion of step-by-step atomicity, and the observer knows at any time what atoms have actually been executed, what effect they potentially had, and what the *visible* variables' values were after each one. With "actually" and "potentially" we are making a distinction between *composite* nondeterminism, written *e.g.* $h{:=}\, 0 \sqcap h{:=}\, 1$ and acting *between* atoms (or larger structures), and *atomic* nondeterminism, written *e.g.* $h{:\in}\, \{0, 1\}$ and acting *within* an atom:

- in the composite case, afterwards we know which of atoms $h{:=}\, 0$ or $h{:=}\, 1$ was executed (actually), and thus we know the value of h too; yet
- in the atomic case, afterwards we know only that the (potential) effects were to set h to 0 or to 1, and thus we know at least (but only) that $h{\in}\{0, 1\}$.

Thus atomicity makes $h{:=}\, 0 \sqcap h{:=}\, 1$ and $h{:\in}\, \{0, 1\}$ different. (Regularity of syntax however allows $v{:=}\, 0 \sqcap v{:=}\, 1$ and $v{:\in}\, \{0, 1\}$ as well; but since in that case we can see v anyway, afterwards, there is no difference between those latter two.)

Fig. 1 illustrates this viewpoint with some small examples.

2.2 An appropriate logic, informally

Our logical language is first-order predicate formulae Φ, interpreted conventionally over the variables of the program, but augmented with a "knows" modal operator so that $\mathrm{K}\Phi$ holds in this state just when Φ itself holds in *all* (other) states compatible with the visible part of this state, the program text and what we have seen (as above) about the execution path and earlier visible values.

The dual modality "possibly" is written $\mathrm{P}\Phi$ and defined $\neg\mathrm{K}(\neg\Phi)$; and it is the modality we will use mainly, as it expresses ignorance directly. (Because $\mathrm{K}\Phi$ seems more easily grasped initially, we explain both.)

Fig. 2 illustrates the logic with our earlier examples in Fig. 1.

2.3 Refinement, and the paradox

Traditional refinement $\sqsubseteq$ between programs allows the reduction of demonic nondeterminism, as in $v{:\in}\, \{0, 1\} \sqsubseteq v{:=}\, 0$.[2] It is a partial order over the program

[1] Moses, Engelhardt and van der Meyden have long advocated combining refinement with the Logic of Knowledge; they operate mainly in a concurrent setting.

[2] It also allows elimination of divergence, which we do not treat here.

In each case we imagine that we are at the end of the program given, that the initial values were v_0, h_0, and that *we* are the observer (so we write "we know" *etc.*)

	Program	Informal commentary
1.1	both $v{:\in}\{0,1\}$ and $v{:=}\,0 \sqcap v{:=}\,1$	We can see the value of v, either 0 or 1.We know h is still h_0, though we cannot see it.
1.2	(one atomic statement) $h{:\in}\{0,1\}$	We know that h is either 0 or 1, but we don't know which; we see that v is v_0.
1.3	(two atomic statements) $h{:=}\,0 \sqcap h{:=}\,1$	We know the value of h, because we know which of atomic $h{:=}\,0$ or $h{:=}\,1$ was executed.
1.4	$h{:\in}\{0,1\}$; $v{:=}\,0 \sqcap v{:=}\,1$	We don't know whether h is 0 or it is 1: even the $\sqcap$-demon cannot see the hidden variable.
1.5	$h{:\in}\{0,1\}$; $v{:\in}\{h, 1{-}h\}$	Though the choice of v refers to h it reveals no information, since the statement is atomic.
1.6	$h{:\in}\{0,1\}$; $v{:=}\,h \sqcap v{:=}\,1{-}h$	Here h is revealed, because we know which of the two atomic assignments to v was executed.
1.7	$h{:\in}\{0,1,2,3\}$; $v{:=}\,h$	We see v; we deduce h since we can see $v{:=}\,h$ in the program text.
1.8	$h{:\in}\{0,1,2,3\}$; $v{:=}\,h \bmod 2$	We see v; from that either we deduce h is 0 or 2, or that h is 1 or 3.
1.9	$h{:\in}\{0,1,2,3\}$; $v{:=}\,h \bmod 2$; $v{:=}\,0$	We see v is 0; but our deductions about h are as for 1.8, because we saw v's earlier value.

In 1.4 the "$\sqcap$-demon" refers anthropomorphically to unpredictable run-time effects as a demon striving to reduce the utility of the program: the worst alternative is taken whenever choice is offered. If for example these are scheduling choices in a concurrent setting, this adversarial scheduler might be said to be "unable to see" certain variables.

We have assumed throughout that v, h are of type $\{0,1\}$ so that, for example, in 1.5 the choice $h{:\in}\{0,1\}$ reveals nothing.

Fig. 1. Examples of ignorance, informally interpreted

lattice [3] and, as such, satisfies $\mathrm{S1} \sqcap \mathrm{S2} \sqsubseteq \mathrm{S1}$ in general; and it is induced by the chosen program logic, so that $\mathrm{S1} \sqsubseteq \mathrm{S2}$ just when all expressible properties of S1 are preserved in S2.

Our expressible properties will be traditional Hoare-style triples (equivalently Dijkstra-style weakest preconditions), *but over formulae whose truth is preserved by increase of ignorance*: those in which all modalities K occur negatively, and all modalities P occur positively. We say that such occurrences of modalities are *ignorant*; and a formula is ignorant just when all its modalities are.

Thus we say that $\mathrm{S1} \sqsubseteq \mathrm{S2}$ just when for all *ignorant* formulae Φ, Ψ we have

$$\{\Phi\}\ \mathrm{S1}\ \{\psi\} \quad \text{implies} \quad \{\Phi\}\ \mathrm{S2}\ \{\psi\}\ , \tag{1}$$

although this is informal here because we have not yet given our interpretation of programs (Sec. 3), or formulae (Sec. 4) or their connection (Sec. 5).

The initial values are v_0, h_0. "Valid conclusion" means *true in all final states* and "Invalid conclusion" means *false in some final state*.

	Program	Valid conclusion	Invalid conclusion
2.1	both $v{:\in}\,\{0,1\}$ and $v{:=}\,0 \sqcap v{:=}\,1$	$v \in \{0,1\}$	$v = 0$
2.2	$h{:\in}\,\{0,1\}$	$\mathrm{P}(h{=}0)$	$\mathrm{K}(h{=}0)$
2.3	$h{:=}\,0 \sqcap h{:=}\,1$	$h \in \{0,1\}$	$\mathrm{P}(h{=}0)$
2.4	$h{:\in}\,\{0,1\}$; $v{:=}\,0 \sqcap v{:=}\,1$	$\mathrm{P}(v{=}h)$	$\mathrm{K}(v{\neq}h)$
2.5	$h{:\in}\,\{0,1\}$; $v{:\in}\,\{h, 1{-}h\}$	$\mathrm{P}(h{=}0)$ In fact Program 2.5 equals Program 2.4.	$\mathrm{P}(v{=}0)$
2.6	$h{:\in}\,\{0,1\}$; $v{:=}\,h \sqcap v{:=}\,1{-}h$	$v \in \{0,1\}$ But Program 2.6 differs from Program 2.5.	$\mathrm{P}(h{=}0)$
2.7	$h{:\in}\,\{0,1,2,3\}$; $v{:=}\,h$	$\mathrm{K}(v{=}h)$	$\mathrm{P}(v{\neq}h)$
2.8	$h{:\in}\,\{0,1,2,3\}$; $v{:=}\,h \bmod 2$	$v{=}0$ $\Rightarrow$ $\mathrm{P}(h{\in}\{2,4\})$	$\mathrm{P}(h{=}1)$ $\wedge$ $\mathrm{P}(h{=}2)$
2.9	$h{:\in}\,\{0,1,2,3\}$; $v{:=}\,h \bmod 2$; $v{:=}\,0$	$\mathrm{P}(h{\in}\{1,2\})$ The $v{:=}\,0$ is an unsuccessful "cover up".	$v{=}0$ $\Rightarrow$ $\mathrm{P}(h{\in}\{2,4\})$

- In 2.3 the invalidity is because $\sqcap$ might resolve to the right: then $h{=}0$ is impossible.
- In 2.6 the invalidity is because $:\in$ might choose 1 and the subsequent $\sqcap$ choose $v{:=}\,h$, in which case v would be 1 and $h{=}0$ impossible.
- In 2.8 the validity is weak: we know h cannot be 4; yet still its membership of $\{2,4\}$ is possible. The invalidity is because the assignment $v{:=}\,h \bmod 2$ leaves us in no doubt about h's parity; we cannot simultaneously consider both 1 and 2 to be possible.
- In 2.9 the invalidity is v might have been 1 earlier.

Fig. 2. Examples of ignorance logic, informally interpreted

We saw that The Refinement Paradox [5] is an issue because traditional refinement allows the "secure" $v{:\in}\,T$ to be refined to the "insecure" $v{:=}\,h$ as an instance of reduction of demonic nondeterminism. But with (1) we have solved that problem: we can show that the property $\{\mathrm{P}(h{=}C)\}\ v{:\in}\,T\ \{\mathrm{P}(h{=}C)\}$ is valid, but that property $\{\mathrm{P}(h{=}C)\}\ v{:=}\,h\ \{\mathrm{P}(h{=}C)\}$ is not valid (Sec. 6).

An operational argument for the refinement's failure is given also.

3 *The Shadow Knows:* an operational model

We now give our ignorance-sensitive interpretation of sequential programs; in Sec. 4 we interpret modal formulae; and in Sec. 5 we connect the two via "weakest preconditions" [2], an approach equivalent to Hoare-triples.

Assume a state space with two variables v (visible) and h (hidden). To model knowledge (and hence ignorance) explicitly, we add a third variable H –called the *shadow* of h– and the shadow "knows" all values that h has *potentially* at

any point. Thus for example $h{:\in}\,\{0,1\}$ leads us to either of two states, one with $h{=}0$ and the other with $h{=}1$; but in both of them the shadow H is $\{0,1\}$.

The operational model is thus given by converting "ignorance-sensitive" (that is v,h-) programs to "ordinary" (that is v,h,H-) programs via the scheme of Fig. 3.[3] In the ordinary programs, traditional sequential semantics applies.

For an ignorance-sensitive program S we write $[\![\mathrm{S}]\!]$ for its conversion into the shadowed form. In this simplified presentation we exclude declarations, supposing only single variables v,h (ranging over a set D), and then H is simply a set of the potential values for h (thus ranging over the powerset $\mathbb{P}.\mathsf{D}$).

On the right the traditional semantics applies: in particular, use of $:\in$ indicates an ordinary nondeterministic choice, from the set given, without any "atomic" implications. Variable e is fresh, just used for the exposition.

Identity	$[\![\mathbf{skip}]\!]$	$\widehat{=}$	$\mathbf{skip}$
Assign to visible	$[\![v{:=}\,E]\!]$	$\widehat{=}$	$e{:=}\,E;\ H{:=}\,\{h{:}\,H \mid e{=}E\};\ v{:=}\,e$
Choose visible	$[\![v{:\in}\,E]\!]$	$\widehat{=}$	$e{:\in}\,E;\ H{:=}\,\{h{:}\,H \mid e{\in}E\};\ v{:=}\,e$
Assign to hidden	$[\![h{:=}\,E]\!]$	$\widehat{=}$	$h{:=}\,E;\ H{:=}\,\{h{:}\,H \cdot E\}$
Choose hidden	$[\![h{:\in}\,E]\!]$	$\widehat{=}$	$h{:\in}\,E;\ H{:=}\,(\cup h{:}\,H \cdot E)$
Demonic choice	$[\![\mathrm{S1} \sqcap \mathrm{S2}]\!]$	$\widehat{=}$	$[\![\mathrm{S1}]\!] \sqcap [\![\mathrm{S2}]\!]$
Composition	$[\![\mathrm{S1};\mathrm{S2}]\!]$	$\widehat{=}$	$[\![\mathrm{S1}]\!];\ [\![\mathrm{S2}]\!]$
Conditional	$[\![\mathbf{if}\ E\ \mathbf{then}\ \mathrm{S1}\ \mathbf{else}\ \mathrm{S2}\ \mathbf{fi}]\!]$		
	$\widehat{=}$	$\mathbf{if}\ E\ \mathbf{then}\ H{:=}\,\{h{:}\,H \mid E\};[\![\mathrm{S1}]\!]\ \mathbf{else}\ H{:=}\,\{h{:}\,H \mid \neg E\};[\![\mathrm{S2}]\!]\ \mathbf{fi}$	

In Fig. 4 we apply the above to give the shadow semantics for our earlier examples.

Fig. 3. Operational semantics

4 Interpretation of the logic

As we foreshadowed (Sec. 2.2), our logical language is first-order augmented with a modal operator so that $\mathrm{K}\Phi$ is read "know Φ" [8, 3.7.2]. Here we set out its interpretation.

We give the language function- (including constant-) and relation symbols as needed, among which we distinguish the (program-variable) symbols *visibles* in V and *hiddens* in H; as well there are the usual (logical) variables in L over which we allow $\forall,\exists$ quantification. The visibles, hiddens and variables are collectively the *scalars* $\mathrm{X} \mathrel{\widehat{=}} \mathrm{V} \cup \mathrm{H} \cup \mathrm{L}$.

A *structure* comprises a non-empty domain D of values, together with functions and relations over it that interpret the function- and relation symbols mentioned above; within the structure we name the partial functions v,h that interpret visibles,hiddens respectively; we write their types $\mathrm{V}\twoheadrightarrow\mathsf{D}$ and $\mathrm{H}\twoheadrightarrow\mathsf{D}$.

[3] Our definitions –in particular the introduction of H– are induced by abstraction, from a lower level given in Sec. A.3.

	(v, h)-program S	(v, h, H)-program ⟦S⟧
4.1a	$v{:\in}\{0,1\}$	$e{:\in}\{0,1\};\ H{:=}\{h{:}\,H \mid e{\in}\{0,1\}\};\ v{:=}e$,
	and the *rhs* simplifies to $v{:\in}\{0,1\}$	
4.1b	$v{:=}0 \sqcap v{:=}1$	$e{:=}0;\ H{:=}\{h{:}\,H \mid e{=}0\};\ v{:=}e$ $\sqcap\ e{:=}1;\ H{:=}\{h{:}\,H \mid e{=}1\};\ v{:=}e$
	simplifies to $v{:\in}\{0,1\}$	
4.2	$h{:\in}\{0,1\}$	$h{:\in}\{0,1\};\ H{:=}(\cup h{:}\,H \cdot \{0,1\})$
	simplifies to $h{:\in}\{0,1\};\ H{:=}\{0,1\}$	
4.3	$h{:=}0 \sqcap h{:=}1$	$h{:=}0;\ H{:=}\{h{:}\,H \cdot 0\}$ $\sqcap\ h{:=}1;\ H{:=}\{h{:}\,H \cdot 1\}$
	simplifies to $h{:\in}\{0,1\}; H{:=}\{h\}$	
4.4	$h{:\in}\{0,1\};$ $v{:=}0 \sqcap v{:=}1$	$h{:\in}\{0,1\};\ H{:=}\{0,1\};$ $v{:\in}\{0,1\}$
	simplifies to $h{:\in}\{0,1\};\ H{:=}\{0,1\};\ v{:\in}\{0,1\}$	
4.5	$h{:\in}\{0,1\};$ $v{:\in}\{h, 1{-}h\}$	$h{:\in}\{0,1\};\ H{:=}\{0,1\};$ $v{:\in}\{h, 1{-}h\};\ H{:=}\{h{:}\,H \mid v{\in}\{h, 1{-}h\}\}$
	which is the same as 4.4	
4.6	$h{:\in}\{0,1\};$ $v{:=}h \sqcap v{:=}1{-}h$	$h{:\in}\{0,1\};\ H{:=}\{0,1\};$ $v, H{:=}h, \{h\} \sqcap v, H{:=}1{-}h, \{h\}$
	simplifies to $h{:\in}\{0,1\};\ v{:\in}\{0,1\};\ H{:=}\{h\}$	
4.7	$h{:\in}\{0,1,2,3\};$ $v{:=}h$	$h{:\in}\{0,1,2,3\};\ H{:=}\{0,1,2,3\};$ $v, H{:=}h, \{h\}$
	simplifies to $h{:\in}\{0,1,2,3\};\ v{:=}h;\ H{:=}\{h\}$	
4.8	$h{:\in}\{0,1,2,3\};$ $v{:=}h \bmod 2$	$h{:\in}\{0,1,2,3\};\ H{:=}\{0,1,2,3\};$ $v{:=}h \bmod 2;\ H{:=}\{h{:}\,H \mid v = h \bmod 2\}$
	simplifies to $(H{:=}\{0,2\} \sqcap H{:=}\{1,3\});\ h{:\in}H;\ v{:=}h \bmod 2$	
4.9	$h{:\in}\{0,1,2,3\};$ $v{:=}h \bmod 2;$ $v{:=}0$	$h{:\in}\{0,1,2,3\};\ H{:=}\{0,1,2,3\};$ $v{:=}h \bmod 2;\ H{:=}\{h{:}\,H \mid v = h \bmod 2\};$ $v{:=}0$
	simplifies to $(H{:=}\{0,2\} \sqcap H{:=}\{1,3\});\ h{:\in}H;\ v{:=}0$	

Fig. 4. Operational-semantics examples

A *valuation* is a partial function from scalars to D, thus typed X↛D; one valuation w_1 can override another w so that for scalar x we have $(w \lhd w_1).x$ is $w_1.x$ if w_1 is defined at x and is w.x otherwise. The valuation $\langle x{\mapsto}d\rangle$ is defined only at x, where it takes value d.

A *state* (v, h, H) comprises a visible- v, hidden- h and *shadow*- part H; the last, in $\mathbb{P}.(H{\nrightarrow}D)$, is a set of valuations over hiddens only. We require that $h \in H$.[4]

[4] Our state corresponds to Fagin's *Kripke structure and state* together [8]; but our use of Kripke structures is extremely limited (App. A). Not only do we make the Common-Domain Assumption, but we do not allow the structure to vary between worlds except for the interpretation h of hiddens.

To allow for declarations of additional variables, we must make H a set of valuations rather than (as in Sec. 3) simply a set of values. We hope it is clear how the simpler view is a special case of this section's more formal presentation.

We define truth of Φ at $(\mathsf{v},\mathsf{h},\mathsf{H})$ under valuation w by induction, writing $(\mathsf{v},\mathsf{h},\mathsf{H}),\mathsf{w} \models \Phi$. Let t be the term-valuation built inductively from the valuation $\mathsf{v} \triangleleft \mathsf{h} \triangleleft \mathsf{w}$. Then we have the following [8, pp. 79,81]:

- $(\mathsf{v},\mathsf{h},\mathsf{H}),\mathsf{w} \models \mathrm{R}.\mathrm{T}_1.\cdots.\mathrm{T}_k$ for relation symbol R and terms $\mathrm{T}_1 \cdots \mathrm{T}_k$ iff the tuple $(\mathsf{t}.\mathrm{T}_1, \cdots, \mathsf{t}.\mathrm{T}_k)$ is an element of the interpretation of R.
- $(\mathsf{v},\mathsf{h},\mathsf{H}),\mathsf{w} \models \mathrm{T}_1 = \mathrm{T}_2$ iff $\mathsf{t}.\mathrm{T}_1 = \mathsf{t}.\mathrm{T}_2$.
- $(\mathsf{v},\mathsf{h},\mathsf{H}),\mathsf{w} \models \neg\Phi$ iff $(\mathsf{v},\mathsf{h},\mathsf{H}),\mathsf{w} \not\models \Phi$.
- $(\mathsf{v},\mathsf{h},\mathsf{H}),\mathsf{w} \models \Phi_1 \wedge \Phi_2$ iff $(\mathsf{v},\mathsf{h},\mathsf{H}),\mathsf{w} \models \Phi_1$ and $(\mathsf{v},\mathsf{h},\mathsf{H}),\mathsf{w} \models \Phi_2$.
- $(\mathsf{v},\mathsf{h},\mathsf{H}),\mathsf{w} \models (\forall \mathrm{L} \cdot \Phi)$ iff $(\mathsf{v},\mathsf{h},\mathsf{H}),\mathsf{w} \triangleleft \langle \mathrm{L} \mapsto \mathsf{d} \rangle \models \Phi$ for all d in D.
- $(\mathsf{v},\mathsf{h},\mathsf{H}),\mathsf{w} \models \mathrm{K}\Phi$ iff $(\mathsf{v},\mathsf{h}_1,\mathsf{H}),\mathsf{w} \models \Phi$ for all h_1 in H.

We write just $(\mathsf{v},\mathsf{h},\mathsf{H}) \models \Phi$ when w is empty, and $\models \Phi$ when $(\mathsf{v},\mathsf{h},\mathsf{H}) \models \Phi$ for all $\mathsf{v},\mathsf{h},\mathsf{H}$ with $\mathsf{h} \in \mathsf{H}$, and we take advantage of the usual "syntactic sugar" for other operators (including P as $\neg\mathrm{K}\neg$). Thus for example we have $\models \Phi \Rightarrow \mathrm{P}\Phi$.

5 Weakest-precondition modal semantics

For practicality, we introduce a weakest-precondition semantics to support direct reasoning at the v,h-level of syntax, *i.e.* without translation to v,h,H-programs. It corresponds to the operational semantics of Fig. 3, given the interpretation in Sec. 4 of the modal formulae.

The predicate-transformer semantics is given in two layers, in Fig. 5 and Fig. 6, because the modal- and classical aspects seem to separate naturally.

Substitute	$[e\backslash E]$	Replaces e by E, with alpha-conversion as necessary if distributing through $\forall, \exists$.

Distribution through P however is affected by that modality's implicitly quantifying over hidden variables: if e is a hidden variable, then $[e\backslash E]\mathrm{P}\Phi$ is just $\mathrm{P}\Phi$; and if E contains hidden variables, the substitution does not distribute into $\mathrm{P}\Phi$ at all (which therefore requires simplification by other means).

Shrink shadow	$[\Downarrow E]$	Distributes through all classical operators, with renaming; has no effect on classical atomic formulae.

We have $[\Downarrow E]\mathrm{P}\Phi \mathrel{\widehat{=}} \mathrm{P}(E \wedge \Phi)$; hidden variables in E are *not* renamed.

Set hidden	$[h \leftarrow E]$	Distributes through all operators, including P, with renaming as necessary for $\forall, \exists$ (not P). Replaces h by E.
Set shadow	$[h \Leftarrow E]$	Distributes through all classical operators, with renaming; has no effect on classical atomic formulae.

For modal formulae we have $[h \Leftarrow E]\mathrm{P}\Phi \mathrel{\widehat{=}} \mathrm{P}(\exists h\colon E \cdot \Phi)$; note that h's in E (if any) are not captured by the $(\exists h \cdots)$.

Fig. 5. Technical predicate transformers

Visible and hidden variables have separate declarations VIS v and HID h respectively. Declarations within a local scope do not affect visibility: a global hidden variable cannot be seen by the observer; a local visible variable can.

Identity	$wp.\mathbf{skip}.\Psi$	$\widehat{=}$	Ψ
Assign to visible	$wp.(v{:=}E).\Psi$	$\widehat{=}$	$[e\backslash E]\,[\Downarrow e{=}E]\,[v\backslash e]\,\Psi$
Choose visible	$wp.(v{:\in}E).\Psi$	$\widehat{=}$	$(\forall e{:}\,E\cdot[\Downarrow e{\in}E]\,[v\backslash e]\,\Psi)$
Assign to hidden	$wp.(h{:=}E).\Psi$	$\widehat{=}$	$[h{\leftarrow}E]\,\Psi$
Choose hidden	$wp.(h{:\in}E).\Psi$	$\widehat{=}$	$(\forall e{:}\,E\cdot[h\backslash e]\,[h{\Leftarrow}E]\,\Psi)$
Demonic choice	$wp.(\mathrm{S1}\sqcap\mathrm{S2}).\Psi$	$\widehat{=}$	$wp.\mathrm{S1}.\Psi\ \wedge\ wp.\mathrm{S2}.\Psi$
Composition	$wp.(\mathrm{S1};\mathrm{S2}).\Psi$	$\widehat{=}$	$wp.\mathrm{S1}.(wp.\mathrm{S2}.\Psi)$
Conditional	$wp.(\mathbf{if}\ E\ \mathbf{then}\ \mathrm{S1}\ \mathbf{else}\ \mathrm{S2}\ \mathbf{fi}).\Psi$		
	$\widehat{=}\quad E\Rightarrow[\Downarrow E]wp.\mathrm{S1}.\Psi\ \wedge\ \neg E\Rightarrow[\Downarrow\neg E]wp.\mathrm{S2}.\Psi$		
Declare visible	$wp.(\textsc{vis}\ v).\Psi$	$\widehat{=}$	$(\forall e\cdot[v\backslash e]\,\Psi)$
Declare hidden	$wp.(\textsc{hid}\ h).\Psi$	$\widehat{=}$	$(\forall e\cdot[h{\leftarrow}e]\,\Psi)$

Note that both these substitutions propagate within modalities in Ψ.

Logical variable e is fresh.

The *assign to visible* rule has two components conceptually. The first is of course an assignment of E to v, although this is split into two sections $[e\backslash E]\cdots[v\backslash e]$ so that v's initial- and final values are distinguished in between, necessary should v occur in E. The second is the "collapse" of ignorance caused by E's value being revealed: this is inserted as a conjunct, by $[\Downarrow e = E]$, into the body of P-modalities.

Fig. 6. Weakest-precondition modal semantics

Occurrences of v, h in the rules may be vectors of visible- or vectors of hidden variables, in which case substitutions such as $[h\backslash h']$ apply throughout the vector. We assume *wlog* that modalities are not nested, since we can remove nestings via $\models \mathrm{P}\Psi \equiv (\exists c\cdot[h\backslash c]\Psi\wedge\mathrm{P}(h{=}c))$.

The congruence of the transformer- and operational semantics is justified by the following observation:

> If we translate the v, h program fragments into v, h, H fragments via the operational semantics (Fig. 3), and translate correspondingly the modal formulae into ordinary first-order formulae, in both cases we have introduced the shadow H explicitly: in effect our language and logic are both regarded as syntactic sugar for a more basic form. For example (recall Example 2.8),
>
> $$v{:=}\,h \bmod 2 \quad \text{becomes} \quad v{:=}\,h \bmod 2;\ H{:=}\{h{:}\,H \mid v = h \bmod 2\}$$
> $$\text{and} \quad v{=}0 \Rightarrow \mathrm{P}(h{\in}\{2,4\}) \quad \text{becomes} \quad v{=}0 \Rightarrow (\exists h{:}\,H \mid h{\in}\{2,4\}).$$
>
> Then the normal *wp*-semantics [2] is used over the explicit v, h, H program fragments, and the resulting preconditions are translated back from the pure first-order $(\exists h{:}\,H\cdots)$-form into the modal P-form.

The *wp*-logic of Figs. 5,6 has the following significant features, which bear directly on the principles we set out in Sec. 1:

1. All visible-only program refinements (hence equalities) are preserved (Pr1).

2. All refinements relying only on Demonic choice, Composition, Identity ("structural") are preserved (Pr2).
3. The transformers defined in Fig. 6 distribute conjunction, as standard transformers do [2]. Thus complicated postconditions can be treated piecewise.
4. Non-modal postconditions can be treated using traditional semantics [1, 2], even if the program contains hidden variables.
5. Because of (3,4) the use of the modal semantics can be restricted to only the modal conjuncts of a postcondition.

Crucially, from (4) we see that we have not *added* refinements. The practical value of (5) is illustrated by the Dining Cryptographers specification (Fig. 10).

6 Avoiding the Refinement Paradox

In this section we see an example of excluding a refinement (Pr3). Fig. 7 uses *wp*-logic to support our claim, in Sec. 2.3 earlier, concerning avoiding the Refinement Paradox on Hoare-triple grounds: if $\not\models wp.\mathrm{S1}.\Psi \Rightarrow wp.\mathrm{S2}.\Psi$, then $\{wp.\mathrm{S1}.\Psi\}$ S1 $\{\Psi\}$ holds (perforce) — but $\{wp.\mathrm{S1}.\Psi\}$ S2 $\{\Psi\}$ does not.

$$
\begin{array}{ll}
 & wp.(v{:\in}\,T).(\mathrm{P}(h{=}C)) \\
\equiv & \text{“Choose visible”} \\
 & (\forall e{:}\,T \cdot [\Downarrow e{\in}T]\,[v\backslash e]\,\mathrm{P}(h{=}C)) \\
\equiv & \text{“}v\text{ not free”} \\
 & (\forall e{:}\,T \cdot [\Downarrow e{\in}T]\,\mathrm{P}(h{=}C)) \\
\equiv & \text{“Shrink shadow”} \\
 & (\forall e{:}\,T \cdot \mathrm{P}(e{\in}T \wedge h{=}C))) \\
\equiv & \text{“}h\text{ not free in }e{\in}T\text{”} \\
 & (\forall e{:}\,T \cdot e{\in}T \wedge \mathrm{P}(h{=}C)) \\
\equiv & \mathrm{P}(h{=}C)\ .\ \text{“}e\text{ not free in P}(h{=}C)\text{”}
\end{array}
\qquad
\begin{array}{ll}
 & wp.(v{:=}\,h).(\mathrm{P}(h{=}C)) \\
\equiv & \text{“Assign to visible”} \\
 & [e\backslash h]\,[\Downarrow e{=}h]\,[v\backslash e]\,\mathrm{P}(h{=}C) \\
\equiv & \text{“}v\text{ not free; Shrink shadow”} \\
 & [e\backslash h]\,\mathrm{P}(e{=}h \wedge h{=}C) \\
\equiv & [e\backslash h]\,\mathrm{P}(e{=}C \wedge h{=}C) \qquad \text{“}h{=}C\text{”} \\
\equiv & \text{“}h\text{ not free in }e{=}C\text{”} \\
 & [e\backslash h]\,(e{=}C \wedge \mathrm{P}(h{=}C)) \\
\equiv & h{=}C \wedge \mathrm{P}(h{=}C) \qquad \text{“}e\text{ not free”} \\
\equiv & h{=}C\ . \qquad \text{“}\models \Phi \Rightarrow \mathrm{P}\Phi\text{”}
\end{array}
$$

We exploit that $\models \mathrm{P}(\Phi \wedge \Psi) \equiv \Phi \wedge \mathrm{P}\Psi$ when Φ contains no hidden variables.

The right-hand side shows that $h{=}C$ is the weakest Φ such that $\{\Phi\}\ v{:=}\,h\ \{\mathrm{P}(h{=}C)\}$, yet $(\mathsf{v},\mathsf{h},\mathsf{H}) \models \mathrm{P}(h{=}C) \Rightarrow (h{=}C)$ for all C only when $\mathsf{H} = \{\mathsf{h}\}$. Thus, when the expression T contains no h, the fragment $v{:\in}\,T$ can be replaced by $v{:=}\,h$ only if we know h already.

Fig. 7. Avoiding the Refinement Paradox, seen logically

The corresponding operational view is that we have S1 $\sqsubseteq$ S2 just when for some initial (v_0, h_0, H_0) every possible outcome (v_2, h_2, H_2) of S2 has $v_1{=}v_2 \wedge h_1{=}h_2 \wedge H_1{\subseteq}H_2$ for some outcome (v_1, h_1, H_1) of S1.[5] We illustrate this via Fig. 8, where we have *e.g.* (8.3) $\sqsubseteq$ (8.2), (8.6) $\sqsubseteq$ (8.5) and $((8.7); v{:=}\,0) \sqsubseteq$ (8.9).

[5] This is the *Smyth* powerdomain-order over an underlying refinement on single triples that allows the H-component –*i.e.* ignorance– to increase [9].

The initial state is $(v_0, h_0, \{h_0\})$.

	Program	Final states in the shadowed model	
8.1	both $v{:\in}\{0,1\}$ and $v{:=}0 \sqcap v{:=}1$	$(0,h_0,\{h_0\})$, $(1,h_0,\{h_0\})$	
8.2	$h{:\in}\{0,1\}$	$(v_0,0,\{0,1\})$, $(v_0,1,\{0,1\})$	
8.3	$h{:=}0 \sqcap h{:=}1$	$(v_0,0,\{0\})$, $(v_0,1,\{1\})$	
8.4	$h{:\in}\{0,1\}$; $v{:=}0 \sqcap v{:=}1$	$(0,0,\{0,1\})$, $(0,1,\{0,1\})$, $(1,0,\{0,1\})$, $(1,1,\{0,1\})$	
8.5	$h{:\in}\{0,1\}$; $v{:\in}\{h, 1{-}h\}$	$(0,0,\{0,1\})$, $(1,0,\{0,1\})$, $(0,1,\{0,1\})$, $(1,1,\{0,1\})$	Thus this and 8.4 are equal.
8.6	$h{:\in}\{0,1\}$; $v{:=}h \sqcap v{:=}1{-}h$	$(0,0,\{0\})$, $(1,0,\{0\})$, $(0,1,\{1\})$, $(1,1,\{1\})$	But this one differs.
8.7	$h{:\in}\{0,1,2,3\}$; $v{:=}h$	$(0,0,\{0\})$, $(1,1,\{1\})$, $(2,2,\{2\})$, $(3,3,\{3\})$	
8.8	$h{:\in}\{0,1,2,3\}$; $v{:=}h \bmod 2$	$(0,0,\{0,2\})$, $(1,1,\{1,3\})$, $(0,2,\{0,2\})$, $(1,3,\{1,3\})$	
8.9	$h{:\in}\{0,1,2,3\}$; $v{:=}h \bmod 2$; $v{:=}0$	$(0,0,\{0,2\})$, $(0,1,\{1,3\})$, $(0,2,\{0,2\})$, $(0,3,\{1,3\})$	The final $v{:=}0$ does not affect H.

In (8.9) partial information about h remains, represented by two possibilities for H of $\{0,2\}$ and $\{1,3\}$, even though $v{=}0$ in all outcomes.

Fig. 8. Examples (Figs. 1,2) revisited: a relational interpretation

Apropos the Paradox we see that $v{:\in}T \not\sqsubseteq v{:=}h$ because the former's final states are $\{e{:}T \cdot (e, h_0, H_0)\}$ whereas the latter's are just $\{\ (h_0, h_0, \{h_0\})\ \}$ and, even supposing $h_0{\in}T$, still in general $H_0 \not\subseteq \{h_0\}$.

7 The Encryption Lemma

In this section we see an example of retaining a refinement (Pr3); and we prepare for our treatment of the *DC* (Dining Cryptographers') protocol.

When a hidden secret is encrypted with a hidden key and published as a visible message, the intention is that observers ignorant of the key cannot use the message to deduce the secret, even if they know the encryption method. A special case of this occurs in the *DC* protocol, where a secret is encrypted (via exclusive-or) with a key (a hidden Boolean) and becomes a message (is published).

We examine this simple situation in the ignorance logic; the resulting formalisation will provide one of the key steps in the *DC* derivation of Sec. 8.

Lemma 1. Let $s{:}\,S$ be a secret, $k{:}\,K$ a key, and $\oslash$ an encryption method so that $s \oslash k$ is the encryption of s. In a context HID s we have the refinement

$$\textbf{skip} \quad \sqsubseteq \quad [\![\ \text{VIS}\ m; \text{HID}\ k \cdot \quad k{:\in}K; m{:=}s \oslash k\ ]\!]\ ,$$

which expresses that publishing the encryption as a message m reveals nothing about the secret s, provided the *Key-Complete Condition* (4) of Fig. 9 (KCC) is satisfied and the key k is not revealed.[6]

Proof. The calculation is given in Fig. 9. Informally we note that the KCC tells us that for every message $s \oslash k$ revealed in m, every guess s' of s we make is supported by some key $k'{:}K$ that could have produced the same m. □

8 Deriving the Dining Cryptographers' Protocol

The Dining Cryptographers Protocol (DC) is an example of ignorance preservation [6]. In the original formulation, three cryptographers have finished their meal, and ask the waiter for the bill: he says it has already been paid; they know that the payer is either one of them or is the *NSA*. They devise a protocol to decide which — without however revealing the payer in the former case.

Each two cryptographers flip a Boolean coin, hidden from the third cryptographer; and each publishes (says aloud) the exclusive-or $\oplus$ of the two coins he sees and a Boolean indicating whether he paid. The exclusive-or of the three announcements, each known to all observers, is true iff some cryptographer paid; but it reveals nothing about which one did.

We model this with global Boolean variables p (some cryptographer paid) and p_i (Cryptographer i paid): a typical specification would include $p = p_0 \oplus p_1 \oplus p_2$ as a *post*-condition. If we take the waiter as observer, then from his point of view the postcondition should also include $p \Rightarrow \langle\!\langle p_0{:}\mathbb{B}\rangle\!\rangle \wedge \langle\!\langle p_1{:}\mathbb{B}\rangle\!\rangle \wedge \langle\!\langle p_2{:}\mathbb{B}\rangle\!\rangle$, where in general for hidden (vector) h we introduce this abbreviation:

– *Complete ignorance* $\quad \langle\!\langle h{:}E \mid \Phi\rangle\!\rangle \;\widehat{=}\; (\forall e{:}E \cdot [h\backslash e]\Phi \Rightarrow \mathrm{P}(h{=}e))$ [7]

(with omitted Φ defaulting to TRUE). It expresses ignorance of h's value beyond its membership in $\{h{:}E \mid \Phi\}$; thus in this case, even if p holds, still the waiter is to know nothing about whether p_0, p_1 or p_2 hold individually.

As a specification *pre*-condition we would find $\langle\!\langle p_0, p_1, p_2{:}\mathbb{B} \mid \sum_i p_i \le 1\rangle\!\rangle$ requiring (with an abuse of notation) that the waiter consider any combination possible provided at most one p_i holds. Putting pre- and post- together, the suitability of a specification S could be expressed

$$\{\langle\!\langle p_0, p_1, p_2{:}\mathbb{B} \mid \sum_i p_i \le 1\rangle\!\rangle\} \;\; \mathrm{S} \;\; \{p = p_0 \oplus p_1 \oplus p_2 \wedge p \Rightarrow \begin{pmatrix} \langle\!\langle p_0{:}\mathbb{B}\rangle\!\rangle \\ \wedge\, \langle\!\langle p_1{:}\mathbb{B}\rangle\!\rangle \\ \wedge\, \langle\!\langle p_2{:}\mathbb{B}\rangle\!\rangle \end{pmatrix}\} \;, \quad (2)$$

and Fig. 10 shows it indeed is satisfied when S is the assignment $p{:=}p_0 \oplus p_1 \oplus p_2$. (Compare Halpern and O'Neill's specification [10]: ours is less expressive because we deal with only one agent at a time.)

[6] The Key-Complete Condition is very strong, requiring as many potential keys as messages; yet it applies to the DC protocol, where both are just one bit.

[7] Naturally we have *e.g.* $\models \langle\!\langle h{:}E\rangle\!\rangle \Rightarrow \mathrm{P}(h{\in}E)$, but in fact the latter is strictly weaker: for example, program $h{:\in}\{0,1\}$ establishes $\mathrm{P}(h{\in}\{1,2\})$ but not $\langle\!\langle h{:}\{1,2\}\rangle\!\rangle$.

postcondition $s{=}B \ \wedge \ \mathrm{P}(s{=}C)$

through $wp.(m{:=}\, s \oslash k)$ gives "Assign visible"

$[e\backslash s \oslash k]\ [\Downarrow e = s \oslash k]\ [m\backslash e]\ (s{=}B \ \wedge \ \mathrm{P}(s{=}C))$

$\equiv$ $s{=}B \ \wedge \ [e\backslash s \oslash k]\ \mathrm{P}(e = s \oslash k \ \wedge \ s{=}C)$ "m not free; Shrink shadow"

$\equiv$ $s{=}B \ \wedge \ [e\backslash s \oslash k]\ \mathrm{P}(e = C \oslash k \ \wedge \ s{=}C)$ "$s{=}C$"

through $wp.(k{:\in}\, K)$ gives "Choose hidden"

$(\forall e{:}\, K \cdot [k\backslash e]\ [k{\Leftarrow}K]\ (s{=}B \ \wedge \ [e\backslash s \oslash k]\ \mathrm{P}(e = C \oslash k \ \wedge \ s{=}C)))$

$\equiv$ $s{=}B \ \wedge \ (\forall e{:}\, K \cdot [k{\Leftarrow}K]\ [k\backslash e]\ [e\backslash s \oslash k]\ \mathrm{P}(e = C \oslash k \ \wedge \ s{=}C))$ "distribute"

$\equiv$ $s{=}B \ \wedge \ (\forall e{:}\, K \cdot [k{\Leftarrow}K]\ [e\backslash s \oslash e]\ \mathrm{P}(e = C \oslash k \ \wedge \ s{=}C))$ "combine subs."

$\equiv$ $s{=}B \ \wedge \ (\forall e{:}\, K \cdot [e\backslash s \oslash e]\ [k{\Leftarrow}K]\ \mathrm{P}(e = C \oslash k \ \wedge \ s{=}C))$ "subs. disjoint"

$\equiv$ $s{=}B \ \wedge \ (\forall e{:}\, K \cdot [e\backslash s \oslash e]\ \mathrm{P}(\exists k'{:}\, K \cdot e = C \oslash k' \ \wedge \ s{=}C))$ "Set shadow"

$\equiv$ $s{=}B \ \wedge \ (\forall e{:}\, K \cdot [e\backslash s \oslash e]\ (\exists k'{:}\, K \cdot e = C \oslash k') \ \wedge \ \mathrm{P}(s{=}C))$ "distribute"

$\equiv$ $s{=}B \ \wedge \ \mathrm{P}(s{=}C) \ \wedge \ (\forall e{:}\, K \cdot (\exists k'{:}\, K \cdot s \oslash e = C \oslash k'))$ "distribute, sub."

We use a subsidiary lemma (App. B) that **skip** $\sqsubseteq$ S if for all A, B, C we have

$$\{v{=}A \wedge h{=}B \wedge \mathrm{P}(h{=}C)\} \quad \mathrm{S} \quad \{v{=}A \wedge h{=}B \wedge \mathrm{P}(h{=}C)\}\ , \tag{3}$$

where v, h are the (vectors of) all variables in context.
In Lem. 1 the context is HID s (and no v), giving the initial calculation boxed above. Because neither m nor k is free in its final line, concluding the calculation by applying the remaining commands VIS m, HID k has no effect. Finally, assuming the precondition, $\forall$-quantifying over B, C, s in their type S, then renaming e, C to k, s', leaves only

$$KCC\text{—} \qquad (\forall s, s'{:}\, S; k{:}\, K \cdot (\exists k'{:}\, K \cdot s \oslash k = s' \oslash k'))\ , \tag{4}$$

which we call the *Key-Complete Condition* for encryption $\oslash$ with key-set K.

Fig. 9. Deriving the Key-Complete Condition for the Encryption Lemma (Sec. 7)

postcondition $p \Rightarrow \langle\!\langle p_1{:}\, \mathbb{B} \rangle\!\rangle$

through $wp.(p{:=}\, p_0 \oplus p_1 \oplus p_2)$ gives "Assign visible"

$[e\backslash p_0 \oplus p_1 \oplus p_2]\ [\Downarrow e = p_0 \oplus p_1 \oplus p_2]\ [p\backslash e]\ (p \Rightarrow \langle\!\langle p_1{:}\, \mathbb{B} \rangle\!\rangle)$

$\equiv$ $[e\backslash p_0 \oplus p_1 \oplus p_2]\ [\Downarrow e = p_0 \oplus p_1 \oplus p_2]\ (e \Rightarrow \langle\!\langle p_1{:}\, \mathbb{B} \rangle\!\rangle)$ "substitute"

$\equiv$ $[e\backslash p_0 \oplus p_1 \oplus p_2]\ [\Downarrow e = p_0 \oplus p_1 \oplus p_2]\ (e \Rightarrow (\forall b{:}\, \mathbb{B} \cdot \mathrm{P}(p_1{=}b)))$ "expand $\langle\!\langle\cdot\rangle\!\rangle$"

$\equiv$ $[e\backslash p_0 \oplus p_1 \oplus p_2]\ (e \Rightarrow (\forall b{:}\, \mathbb{B} \cdot \mathrm{P}(e = p_0 \oplus p_1 \oplus p_2 \ \wedge \ p_1{=}b)))$ "Shrink shadow"

$\equiv$ $[e\backslash p_0 \oplus p_1 \oplus p_2]\ (e \Rightarrow (\forall b{:}\, \mathbb{B} \cdot \mathrm{P}((p_0 \oplus p_1 \oplus p_2) \ \wedge \ p_1{=}b)))$ "e in antecedent"

$\Leftarrow$ $\mathrm{P}((p_0{=}p_2) \wedge p_1) \ \wedge \ \mathrm{P}((p_0 \oplus p_2) \wedge \neg p_1)$ "drop antecedent; expand $\forall$"

$\Leftarrow$ $\mathrm{P}(\neg p_0 \wedge \neg p_2 \wedge p_1) \ \wedge \ \mathrm{P}(\neg p_0 \wedge p_2 \wedge \neg p_1)$ "P is $\Rightarrow$-monotonic"

$\Leftarrow$ $(\forall e_0, e_1, e_2{:}\, \mathbb{B} \cdot \sum_i e_i \leq 1 \Rightarrow \mathrm{P}(p_0, p_1, p_2{=}e_0, e_1, e_2))$ "instantiate $\forall$ twice"

$\equiv$ $\langle\!\langle p_0, p_1, p_2{:}\, \mathbb{B} \mid \sum_i p_i \leq 1 \rangle\!\rangle$. "contract $\langle\!\langle\cdot\rangle\!\rangle$"

Because *wp* is conjunctive (Sec. 5 point 3.) we can deal with postcondition conjuncts separately; the standard part follows from ordinary *wp* (Sec. 5 point 4.); and by symmetry we can concentrate *wlog* on the p_1 case for the remainder.

Fig. 10. Adequacy of the cryptographers' specification (Sec. 8)

Rather than prove (2) for an implementation directly –which could be complex– we can *use program algebra to manipulate a specification for which (2) has already been established*. An implementation reached via ignorance-preserving refinement steps requires no further proof of ignorance-preservation [11].

A derivation of the *DC* protocol, from the specification "validated" in Fig. 10, is given in Fig. 11. To illustrate the possibility of different viewpoints, we observe as Cryptographer 0 –rather than as the waiter– which makes p_0 visible rather than hidden and thus alters our global context to VIS p, p_0; HID p_1, p_2: we can see the final result, and we can "see" whether *we* paid; but we cannot see directly whether Cryptographers 1 or 2 paid.

9 Contributions, comparisons and conclusions

Consider this putative refinement in which the variables range over arbitrary integers which, "Boolean-wise" however, are each set initially in $\{0,1\}$:

$$p{:=}\, p_0 \oplus p_1 \oplus p_2 \quad \sqsubseteq^{?} \quad [\![\ \text{VIS } s_0, s_1, s_2, c_1, c_2{:}\{0,1\}; \ \text{HID } c_0{:}\{0,1\} \cdot \ s_0{:=}\, c_1 + p_0 - c_2;\ s_1{:=}\, c_2 + p_1 - c_0;\ s_2{:=}\, c_0 + p_2 - c_1;\ p{:=}\,(s_0 + s_1 + s_2) \bmod 2 \ ]\!] \,. \tag{5}$$

If $c_2, p_1, c_0 = 1, 1, 0$, then visible s_1 will be 2 and hidden p_1=1 is revealed. *It shouldn't be.*

The coins c_i cancel just as in Fig. 11, but this time additively. Although Refinement (5) is valid traditionally, the boxed text shows that it does not satisfy *ignorance-preserving* refinement, as we have defined it, in any context where p_1 was declared to be hidden — and so its traditional proof must use some rule excluded by Pr3. Thus, in a sense,

Our contribution has been to disallow this refinement, and others like it.[8]

More generally our contribution is to have altered the rules for refinement of sequential programs, just enough, so that ignorance of hidden variables is preserved. We can still derive correct protocols (Fig. 11), but can no longer mistakenly propose incorrect ones (5).

Compared to the work of Halpern and O'Neill, who apply the Logic of Knowledge to secrecy [7] and anonymity [10], ours is a very restricted special case: we allow just one agent; our $(\mathsf{v}, \mathsf{h}, \mathsf{H})$ model allows only h to vary in the Kripke model [8]; and our programs are not concurrent. They treat *DC*, as do Engelhardt, Moses & van der Meyden [12], and van der Meyden & Su [13].

What we add back –having specialised away so much– is reasoning in the *wp*-based assertional/sequential style, thus exploiting the specialisation to preserve traditional reasoning patterns where they can apply.

[8] One role of Formal Methods is to *prevent* people from writing incorrect programs.

The global context VIS p, p_0; HID p_1, p_2 expresses Cryptographer 0's viewpoint; we number the coins so that c_i is opposite p_i, and thus c_0 is the only coin he can't see.

$p{:=}\, p_0 \oplus p_1 \oplus p_2$

$=$ **skip**; "**skip** is identity"
$p{:=}\, p_0 \oplus p_1 \oplus p_2$

$\sqsubseteq$ "Encryption Lemma"
$[\![$ **vis** s_1, c_2**; hid** c_0 $\cdot$
$c_0{:}\in \mathbb{B}$;
$s_1{:=}\, c_2 \oplus p_1 \oplus c_0$ $]\!]$;
$p{:=}\, p_0 \oplus p_1 \oplus p_2$

$=$ "move into block; typed declaration"
$[\![$ VIS s_1, c_2; HID **$c_0{:}\,\mathbb{B}$** $\cdot$
$s_1{:=}\, c_2 \oplus p_1 \oplus c_0$
$p{:=}\, p_0 \oplus p_1 \oplus p_2$ $]\!]$

$=$ "new inner block equals **skip**"
$[\![$ VIS s_1, c_2; HID $c_0{:}\,\mathbb{B}$ $\cdot$
$s_1{:=}\, c_2 \oplus p_1 \oplus c_0$
$p{:=}\, p_0 \oplus p_1 \oplus p_2$
$[\![$ **vis** s_0, s_2, c_1 $\cdot$
$s_0{:=}\, c_1 \oplus p_0 \oplus c_2$;
$s_2{:=}\, p \oplus s_0 \oplus s_1$ $]\!]$ $]\!]$

$=$ "reorder"
$[\![$ VIS **s_0**, s_1, **s_2**, **c_1**, c_2; HID $c_0{:}\,\mathbb{B}$ $\cdot$
$s_0{:=}\, c_1 \oplus p_0 \oplus c_2$;
$s_1{:=}\, c_2 \oplus p_1 \oplus c_0$
$p{:=}\, p_0 \oplus p_1 \oplus p_2$
$s_2{:=}\, p \oplus s_0 \oplus s_1$ $]\!]$

$=$ "Boolean algebra"
$[\![$ VIS s_0, s_1, s_2, c_1, c_2; HID $c_0{:}\,\mathbb{B}$ $\cdot$
$s_0{:=}\, c_1 \oplus p_0 \oplus c_2$;
$s_1{:=}\, c_2 \oplus p_1 \oplus c_0$
$p{:=}\, p_0 \oplus p_1 \oplus p_2$
$s_2{:=}\, c_0 \oplus p_2 \oplus c_1$ $]\!]$

$=$ "reorder"
$[\![$ VIS s_0, s_1, s_2, c_1, c_2; HID $c_0{:}\,\mathbb{B}$ $\cdot$
$s_0{:=}\, c_1 \oplus p_0 \oplus c_2$;
$s_1{:=}\, c_2 \oplus p_1 \oplus c_0$
$s_2{:=}\, c_0 \oplus p_2 \oplus c_1$
$p{:=}\, p_0 \oplus p_1 \oplus p_2$ $]\!]$

The derivation begins at upper-left with the specification, ending with the implementing protocol at right. **Bold text** highlights changes.

The "typed declaration" HID $c_0{:}\,\mathbb{B}$ $\cdot$ abbreviates HID c_0 $\cdot$ $c_0{:}\in \mathbb{B}$.

$=$ "Boolean algebra"
$[\![$ VIS s_0, s_1, s_2, c_1, c_2; HID $c_0{:}\,\mathbb{B}$ $\cdot$
$s_0{:=}\, c_1 \oplus p_0 \oplus c_2$;
$s_1{:=}\, c_2 \oplus p_1 \oplus c_0$
$s_2{:=}\, c_0 \oplus p_2 \oplus c_1$
$p{:=}\, s_0 \oplus s_1 \oplus s_2$ $]\!]$.

Local variables c_i (coins) and s_i (Cryptographer i *said*) for $i{:}\,0, 1, 2$ are introduced during the derivation, which depends principally on Key-Completeness and the Boolean algebra of $\oplus$.

In our use of Lem. 1 (Encryption) the message (m) is s_1, the secret (s) is p_1, the encryption ($\oslash$) is $\cdot(\oplus\, c_2\, \oplus)\cdot$ –which satisfies the Key-Complete Condition (4) for both values of the visible c_2– and the key (k) is c_0.

Other refinements, such as moving statements into blocks where there is no capture, and swapping of statements that do not share variables, are examples of Pr1 and Pr2.

Fig. 11. Deriving the Dining Cryptographers' Protocol

Comparison with security comes from regarding hidden variables as "high-security" and visible variables as "low-security", and concentrating on program semantics rather than *e.g.* extra syntactic annotations: thus we take the *extensional* view [14] of *non-interference* [15] where security properties are deduced directly from the semantics of a program [16, III-A]. Recent examples of this include elegant work by Leino *et al.* [17] and Sabelfeld *et al.* [18].

Again we have specialised severely — we do not consider lattices, nor loops (and thus possible divergence), nor concurrency, nor probability. However our "agenda" of Refinement, the Logic of Knowledge, and Program Algebra, has induced four interesting differences from the usual approaches to security:

1. *We do not prove "absolute" security of a program.* Rather we show that it is no less secure than another; this is induced by our refinement agenda. After all, the *DC specification* is not secure in the first place: it reveals whether the cryptographers (collectively) paid or not. To attempt to prove the *DC implementation* (absolutely) secure is therefore pointless.

   However, if we did wish to establish absolute security we would simply prove refinement of an absolutely secure specification (*e.g.* **skip**, or $v{:\in}\,T$).
2. *We concentrate on* final- *rather than initial hidden values.* This is induced by the Kripke structure of the Logic of Knowledge approach (App. A), which models what other states are possible "now" (rather than "then").

   The usual approach relates instead to hidden *initial* values, so that $h{:=}\,0$ would be secure and $v{:=}\,h; h{:\in}\,T$ insecure; for us just the opposite holds. Nevertheless, we could achieve the same effect by operating on a local hidden copy, thrown away at the end of the block. Thus $[\![\ \textsc{hid}\ h'{:}\,\{h\} \cdot h'{:=}\,0\ ]\!]$ is secure (for both interpretations), and $[\![\ \textsc{hid}\ h'{:}\,\{h\} \cdot v{:=}\,h'; h'{:\in}\,T\ ]\!]$ is insecure.

   A direct comparison with *non-interference* considers the relational semantics R of a program, operating over $v, h{:}\,T$; the refinement $v{:\in}\,T \sqsubseteq v{:\in}\,R.v.h$ then expresses absolute security for the *rhs* with respect to h's initial value. Operational reasoning (App. C) then shows that

$$\textit{Absolute security} — \qquad (\forall v, h, h'{:}\,T \cdot R.v.h = R.v.h')$$

   is necessary and sufficient, which is non-interference for R exactly [17, 18].
3. *We insist on perfect recall.* This is induced by our algebraic principles (recall the *gedanken* experiment of Sec. 2), and thus we consider $v{:=}\,h$ to have revealed h's value at that point, no matter what follows.[9] The usual semantic approach allows instead a subsequent $v{:=}\,0$ to "erase" the information leak.

   Perfect recall is also a side-effect of (thread) concurrency [7],[16, IV-B], but has different causes. We are concerned with ignorance-preservation during program *development*; the concurrency-induces-perfect-recall problem occurs during program *execution*.

   The "label creep" [16, II-E] caused by perfect recall, where the build-up of un-erasable leaks makes the program eventually useless, is mitigated

[9] A similar experiment shows the principles also imply that we can see the program counter.

because our knowledge of the *current* hidden values can decrease (via *e.g.* $h{:\in}\, T$), even though knowledge of *initial-* (or even previous) values cannot.

4. *We do not require "low-view determinism"* [16, IV-B]. This is induced by our explicit policy of retaining abstraction, and of determining exactly when we can "refine it away" and when we cannot. The approach of Roscoe and others instead requires low-level behaviour to be deterministic [19].

We conclude that *ignorance refinement* is able to handle examples of simple design, at least — even though their significance may be far from simple. Because *wp*-logic for ignorance retains most structural features of traditional *wp*, we expect that loops and their invariants, divergence, and concurrency via *e.g. action systems* [20] could be feasible extensions.

Adding probability via modal "expectation transformers" [21] is a longer-term goal, but will require a satisfactory treatment of conditional probabilities (the probabilistic version of *Shrink shadow*) in that context.

Acknowledgements

Thanks to Yoram Moses and Kai Engelhardt for the problem, the right approach and the historical background, and to Annabelle McIver, Jeff Sanders, Mike Spivey, Susan Stepney and members of IFIP WG2.1 for suggestions. (Lambert Meertens proposed numbering the cryptographers' coins symmetrically.)

The reviewers were very helpful also.

Michael Clarkson and Chenyi Zhang supplied useful references.

References

1. Hoare, C.: An axiomatic basis for computer programming. Communications of the ACM **12**(10) (1969) 576–80, 583
2. Dijkstra, E.: A Discipline of Programming. Prentice Hall International, Englewood Cliffs, N.J. (1976)
3. Back, R.J., von Wright, J.: Refinement Calculus: A Systematic Introduction. Springer Verlag (1998)
4. Morgan, C.: Programming from Specifications. second edn. Prentice-Hall (1994) `web.comlab.ox.ac.uk/oucl/publications/books/PfS/`.
5. Jacob, J.: Security specifications. In: IEEE Symposium on Security and Privacy. (1988) 14–23
6. Chaum, D.: The Dining Cryptographers problem: Unconditional sender and recipient untraceability. J. Cryptol. **1**(1) (1988) 65–75
7. Halpern, J., O'Neill, K.: Secrecy in multiagent systems. In: Proc. 15th IEEE Computer Security Foundations Workshop. (2002) 32–46
8. Fagin, R., Halpern, J., Moses, Y., Vardi, M.: Reasoning about Knowledge. MIT Press (1995)
9. Smyth, M.: Power domains. Jnl. Comp. Sys. Sciences **16** (1978) 23–36
10. Halpern, J., O'Neill, K.: Anonymity and information hiding in multiagent systems. In: Proc. 16th IEEE Computer Security Foundations Workshop. (2003)

11. Mantel, H.: Preserving information flow properties under refinement. In: Proc. IEEE Symp. Security and Privacy. (2001) 78–91
12. Engelhardt, K., Moses, Y., van der Meyden, R.: Unpublished report (2005)
13. van der Meyden, R., Su, K.: Symbolic model checking the knowledge of the Dining Cryptographers. In: Proc. 17th IEEE Worksh. Computer Security Foundations. (2004) 280–91
14. Cohen, E.: Information transmission in sequential programs. ACM SIGOPS Operatings Systems Review **11**(5) (1977) 133–9
15. Goguen, J., Meseguer, J.: Unwinding and inference control. In: Proc. IEEE Symp. on Security and Privacy. (1984) 75–86
16. Sabelfeld, A., Myers, A.: Language-based information-flow security. IEEE Jnl. Selected Areas Comm. **21**(1) (2003)
17. Leino, K., Joshi, R.: A semantic approach to secure information flow. Science of Computer Programming **37**(1–3) (2000) 113–38
18. Sabelfeld, A., Sands, D.: A PER model of secure information flow. Higher-Order and Symbolic Computation **14** (2110) 59–91
19. Roscoe, A.W., Woodcock, J., Wulf, L.: Non-interference through determinism. Journal of Computer Security **4**(1) (1996) 27–54
20. Back, R.J., Kurki-Suonio, R.: Decentralisation of process nets with centralised control. In: 2nd ACM SIGACT-SIGOPS Symp. Principles of Distributed Computing. (1983) 131–42
21. McIver, A., Morgan, C.: Abstraction, Refinement and Proof for Probabilistic Systems. Tech. Mono. Comp. Sci. Springer Verlag, New York (2005)
22. Hintikka, J.: Knowledge and Belief: an Introduction to the Logic of the Two Notions. Cornell University Press (1962) Available in a new edition, Hendricks and Symonds, Kings College Publications, 2005.
23. Halpern, J.Y., Moses, Y.: Knowledge and common knowledge in a distributed environment. Journal of the ACM **37**(3) (1990) 549–587 Earlier appeared in Proc. *PoDC*, 1984.

A The Logic of Knowledge

A.1 Introduction, and inspiration

The seminal work on formal logic for knowledge is Hintikka's [22], who used Kripke's possible-worlds semantics for the model: he revived the discussion on a subject which had been a topic of interest for philosophers for millennia. It was first related to multi-agent computing by Halpern and Moses [23], and much work by many other researchers followed. Fagin *et al.* summarise the field in their definitive introduction [8].

Engelhardt, Moses, and van der Meyden have earlier treated the *DC* via a refinement-calculus of knowledge and ignorance [12], and their work (and advice from them) is the direct inspiration for what is here. It supports our presentation in the following way.

The standard model for knowledge-based reasoning is based on possible "runs" of a system and participating agents' ignorance of how the runs have interleaved: although each agent knows the (totality of) the possible runs, a sort of "static" knowledge, he does not have direct "dynamic" knowledge of which

run has been taken on any particular occasion. Thus he knows a fact in a given global state (of an actual run) iff that fact holds in all possible global states (allowed by other runs) that have the same local state as his.

We severely specialise this view in three ways. The first is that we consider only sequential programs, with explicit demonic choice. As usual, such choice can represent both *abstraction*, that is freedom of an implementor to choose among alternatives (possible refinements), and *ignorance*, that is not knowing which environmental factors might influence run-time decisions.

Secondly, we consider only one agent: informally, we think of this as an observer of the system, whose local state is our system's visible part and who is is trying to learn about (what is for him) the non-local, hidden part.

Finally, we emphasise ignorance rather than (its dual) knowledge, and *loss of ignorance* is sufficient to exclude an otherwise acceptable refinement.

A.2 The model as a Kripke structure

We are given a sequential program text, including a notion of atomicity: unless stated otherwise, each *syntactically* atomic element of the program changes the program counter when it is executed. Demonic choice is either a (non-atomic) choice between two program fragments, thus $\mathrm{S1} \sqcap \mathrm{S2}$, or an (atomic) selection of a variable's new value from some set, thus $x{:\in}\, X$. For simplicity we suppose we have just two (untyped) variables, the visible v and the hidden h.

The global state of the system comprises both v, h variables' current and all previous values, sequences $\overline{v}, \overline{h}$, together with a history-sequence $\overline{p}$ of the program counter; the observer can see $\overline{v}, \overline{p}$ but not $\overline{h}$. For example, after $\mathrm{S1}; (\mathrm{S2} \sqcap \mathrm{S3}); \mathrm{S4}$ he can use $\overline{p}$ to "remember" which of S2 or S3 was executed earlier.

The possible runs of a system S are all sequences of global states that could be produced by the successive execution of atomic steps from some initial v_0, h_0, including all outcomes resulting from demonic choice (both $\sqcap$ and $:\in$).

If the current state is $(\overline{v}, \overline{h}, \overline{p})$, then the set of *possible* states associated with it is the set of triples $(\overline{v}, \overline{h}_1, \overline{p})$ that S can produce from v_0, h_0. We write $(\overline{v}, \overline{h}, \overline{p}) \sim (\overline{v}, \overline{h}_1, \overline{p})$ for this (equivalence) relation, which depends on S, v_0, h_0.

Thus from $\overline{p}$ the observer knows the execution trace; from $\overline{v}$ he knows the successive v values; but of hiddens he knows only h_0 directly. Fig. 1 illustrated this viewpoint with its small examples.

A.3 The connection with the shadowed operational model

The correspondence between the Kripke model of Sec. A.2 and the shadow model of Sec. 3 is via the abstraction

$$\mathsf{v} = \mathsf{last}.\overline{v} \;\wedge\; \mathsf{h} = \mathsf{last}.\overline{h} \;\wedge\; \mathsf{H} = \{\overline{h}' \mid (\overline{v}, \overline{h}', \overline{p}) \sim (\overline{v}, \overline{h}, \overline{p}) \cdot \mathsf{last}.\overline{h}'\} \;, \quad [10]$$

and it determines the operational semantics we gave in Fig. 3.

[10] Read the last as "vary $\overline{h}'$ such that $(\overline{v}, \overline{h}', \overline{p}) \sim (\overline{v}, \overline{h}, \overline{p})$ and take $\mathsf{last}.\overline{h}'$".

The abstraction works because programs cannot refer to the full run-sequences directly; what they *can* refer to –the current values of v, h– is just what is captured in the abstraction. The shadow H is used by the modal-logic semantics: it determines the *accessibility* relation with respect to which modalities are interpreted.[11]

B Subsidiary lemma for skip (Fig. 9)

Sufficient conditions for **skip** $\sqsubseteq$ S can be obtained operationally. Recall that (operational) refinement includes increase of H, thus being an option available to any refinement of **skip** (which nevertheless must still leave v and h unchanged). Hence the refinement fails only if $[\![\mathrm{S}]\!]$ can take some initial $\mathsf{v}, \mathsf{h}, \mathsf{H}$ to some final $\mathsf{v}', \mathsf{h}', \mathsf{H}'$, that is (writing $[\![\mathrm{S}]\!]$ as a relation)

$$(\mathsf{v}, \mathsf{h}, \mathsf{H}) \quad [\![\mathrm{S}]\!] \quad (\mathsf{v}', \mathsf{h}', \mathsf{H}') \ , \tag{6}$$

and then we find either $\mathsf{v}{\neq}\mathsf{v}'$ or $\mathsf{h}{\neq}\mathsf{h}'$ or $\mathsf{H}{\not\subseteq}\mathsf{H}'$. The first two possibilities are excluded directly by the A and B terms in the postcondition of Fig. 9's (3). For the third, we merely pick some C with $C{\in}\mathsf{H}$ but $C{\notin}\mathsf{H}'$; then the initial state satisfies $v{=}A \wedge h{=}B \wedge \mathrm{P}(h{=}C)$ for appropriate A, B but the final state does not. Thus the condition (3) excludes this case also.

C Comparison with non-interference (Sec. 9)

Consider a program with relational semantics R operating over $v, h{:}\,T$, and assume we have $v{:\in}\,T \sqsubseteq v{:\in}\,R.v.h$. Fig. 3 shows that from initial $\mathsf{v}, \mathsf{h}, \mathsf{H}$ the possible outcomes on the left are $t, \mathsf{h}, \mathsf{H}$ for all $t{:}\,T$, and that on the right they are $t', \mathsf{h}, \{h : \mathsf{H} \mid t'{\in}R.\mathsf{v}.h\}$ for all $t'{:}\,R.\mathsf{v}.\mathsf{h}$. For refinement from that initial state we must therefore have $t'{\in}R.\mathsf{v}.\mathsf{h} \Rightarrow t'{\in}T$, which is just type-correctness; but also

$$t' \in R.\mathsf{v}.\mathsf{h} \quad \Rightarrow \quad \mathsf{H} \subseteq \{h : \mathsf{H} \mid t'{\in}R.\mathsf{v}.h\} \tag{7}$$

must hold, both conditions coming from the operational definition of refinement (for which we recall Footnote 5 in Sec. 6). Since (7) constrains all initial states $\mathsf{v}, \mathsf{h}, \mathsf{H}$, and all t', we close it universally over those variables to give a formula which via predicate calculus and elementary set-theory reduces to the non-interference condition $(\forall \mathsf{v}, \mathsf{h}, \mathsf{h}'{:}\,T \cdot R.\mathsf{v}.\mathsf{h} = R.\mathsf{v}.\mathsf{h}')$ as usually given for demonic programs [17, 18].

[11] In fact the H-component makes h redundant –*i.e.* we can make do with just (v, H)– but this extra "compression" would complicate the presentation subsequently. The redundancy is captured by the healthiness condition

$$wp.\mathrm{S}.(\mathrm{K}\Psi) \quad = \quad \mathrm{K}(wp.\mathrm{S}.\Psi) \ .$$