Do not read this

Juan C. Bicarregui

Published version information

Citation: JC Bicarregui. 'Do not read this.' In FME 2002: Formal Methods—Getting IT Right (FME 2002), Copenhagen, Denmark, 22-24 Jul 2002, (2002): 106-125. Lecture Notes in Computer Science, vol 2391. Springer, Berlin, Heidelberg.

DOI: https://doi.org/10.1007/3-540-45614-7_7

Do not read this

Juan Bicarregui
CLRC Rutherford Appleton Laboratory, Oxfordshire, UK
http://www.bitd.clrc.ac.uk/Person/J.C.Bicarregui

Abstract. We discuss the interpretation of read and write frames in model-oriented specification taking the B's generalised substitutions as the vehicle for the presentation. In particular, we focus on the interpretation of read frames, the semantics of which have not been considered by previous authors. We gives several examples of the relevance of read frames and show that a substitution admits a read respecting implementation if and only if a certain bisimulation condition is satisfied. We use this to motivate a richer semantic model for substitutions which interprets read and write constraints directly in the denotation of a substitution. This semantics yields some non-interference results between substitutions which cannot be given at this level without the use of read and write frames.

1 Introduction

In [Dun02], Dunne raises the question of whether $x := x$ and *skip* are equivalent and argues that they should not be considered equivalent because they do not exhibit substitutivity of equals, in particular, when composed in parallel with another substitution writing x. For example, $skip \parallel x := x + 1$ is well formed, whereas $x := x \parallel x := x + 1$ is not. This and other considerations lead Dunne to develop a semantic model for generalised substitutions which strengthens the standard weakest precondition semantics with an explicit *write frame*[1]. This enables the distinction of the above substitutions at the semantic level and leads to the clarification of various properties of generalised substitutions.

The present author has previously advocated the explicit treatment of frames in the semantics of model oriented specifications and has proposed a number of models which extend the relational semantics of VDM[Jon86,Jon87] with explicit treatment of read and write frames [Bic93,Bic94,Bic95]. In this paper, we consider read frames in B and discuss the interpretation of the variables which can be read in a substitution. To continue the above example, a similar question to that raised by Dunne above is whether the substitution $x := 0$ is equivalent to $x := y\text{-}y$? In this case well formedness of the latter expression depends on which variables are in (read) scope and so substitutivity depends on the visibility rules employed in a structured specification.

In this paper we define the concept of read and write frames for B's generalised substitutions[Abr95] and show how they can be used to enrich the semantic

[1] Dunne calls it the *active frame*

model of substitutions to yield non-interference properties between substitutions and a richer definition of refinement.

1.1 Examples of the relevance of read frames

Read frames and non-interference Under what conditions is $S \parallel T$ is refined by $S; T$ Clearly, a sufficient condition is *if T does not read and variables written by S*, but in view of the above example, $x := y$-y, this is clearly not a necessary condition and a more precise definition would be if T *does not depend* on any variable written by S.

One would like to be able formalise the semantic justification of syntactic sufficient conditions for non-interference such as:

$$\frac{reads(S2) \cap writes(S1) = \{\}}{S1 \parallel S2 \sqsubseteq S1; S2}$$

Read frames and initialisation Consider the substitution which arbitrarily sets a boolean variable ($x := true [] x := false$). Two obvious refinements of this substitution are $x := true$ and $x := false$. But there are also others such as $x := x$, $x := \neg x$ or $x := y$, for some other boolean variable y. If this substitution appeared in an initialisation, syntactic constraints would be applied which prohibited this latter set of implementations. However, all of these implementations are admitted by the standard semantics of refinement.

In this case, what we wish to say, is that the substitution should be refined by a substitution that *does not read any variable* thus permitting only the first two of the implementations given above.

Read frames and encapsulation Consider the same substitution but in a context where x and y relate to separate aspects of the specification. In order to allow for separate development of the components related to x and y, the specifier may wish to indicate that the substitution should be refined without reference to y. That is, so as to allow the first 4 implementations which manipulate x according to its previous value, but not the fifth where the outcome is dependent on y.

Clearly, machine structuring could be used to give full hiding of the y variable in the machine manipulating x but this brings with it other constraints which may restrict the specifiers freedom. Furthermore, how is one to demonstrate formally that the syntactic conditions imposed by the structuring mechanisms actually ensure the desired semantic properties.

In this case, by specifying that the substitution must be refined by a substitution that *does not read y*, we can allow the first four refinements but disallow the last.

Read frames and underspecification In B it is commmonplace to use machine CONSTANTS to specify that looseness present in a specification should be treated as underspecification, that is, that it must be implemented by a deterministic function. For example, the machine with a constant $c: 0, 1$, a variable, $x: 0, 1$, and an operation, $op == x := c$ admits implementations $x := 0$ and $x := 1$. That is, like the initialisation example above, it *does not read any variable* in its implementation.

Note that if we try to specify this range of implementations without the use of constants, for example by $(x := 0 [] x := 1)$, we also admit implementations such as *skip*, $x := x$-1 etc,. as well as implementations which make use of state which is added as a result of refinement later in the development.

Again, specifying a read frame for the substitution could be a way to capture the required semantics. (Naturally, constants can be used more generally that this, for example to relate aspects of different substitutions in a machine, but this use exemplifies a reasonably common case.)

1.2 Read frames and refinement

For all of the above examples we would wish to formalise how such properties interact with refinement. Note that it is not the case that non-dependence on a variable is preserved by refinement since the refinement may resolve some non-determinism present in the specification by refence to a variable not referenced in the abstract description. For example when the above example is refined by $x := y$.

Ultimately, one would wish to give a definition of read respecting refinement which ensure no new read dependencies are introduced as a result of reduction of non-determinism. For example one would wish to give rules which restrict the usual definition of refinement, such as

$$\frac{vars(P) \subset reads(S)}{S \sqsubseteq (P \Longrightarrow S)}$$

Classical interpretations of specifications and refinements make it impossible to justify such rules.

1.3 Objective

In this paper we consider the semantic conditions required for such properties to hold. This leads us to propose a strengthening of the semantics to also include a second frame, a "read frame" which determines which variables can influence the outcome of a substitution or any refinement of it. This extended semantic model supports the formalisation, at specification time, of implementation

constrains hitherto only captured by syntactic constraints implicit in the structuring of specifications and so clarifies some issues related to non-interference, initialisation, encapsulation, and underspecification in B developments.

2 Language

We consider a language similar to that in [Dun02] but with a slight change to the construct for (write) frame extension and with the addition of similar construct read frames. Thus language of substitutions is given by:

name	subst	default read frame	default write frame
skip	$skip$	empty	$\{\}$
simple assignment	$x := E$	vars E	$\{x\}$
preconditioned substitution	$P \mid S$	vars P $\cup$ reads S	writes S
guarded substitution	$G \Longrightarrow S$	vars G $\cup$ reads S	writes S
sequential substitution	$S ; T$	reads S $\cup$ reads T	writes S $\cup$ writes T
bounded choice	$S [] T$	reads S $\cup$ reads T	writes S $\cup$ writes T
unbounded choice	$@z.S$	reads S $-\{z\}$	writes S $-\{z\}$
opening	$S^{\wedge}$	reads S	writes S
parallel composition	$S \parallel T$	reads S $\cup$ reads T	writes S $\cup$ writes T
set_reads	S_{R}	R	writes S
set_writes	${}_{W}S$	reads S	W

The default frames are those which are calculated from the text of the substitution. The set_reads and set_writes constructs can be used to override the default frames. Note that set_reads and set_writes set the respective frames to those given whatever they were perviously. Thus the frames can be expanded or contracted by these constructs.

2.1 Abstract syntax

In the abstract syntax, we make explicit the read and write frames for all substitutions and also incorporate the alphabet of variables in scope. Thus, formally, a substitution is a quadruple (F, R, W, S) where F is the set of all variables in scope, R is the subset of F which are readable, W is the subset of F which are writeable and S is the body of the generalised substitution. The first component declares and binds the variables appearing in the substitution. The second and third give information about access to the variables that must be respected by any implementation, and the last gives the body of the substitution[2]. We do not insist on any relationship between the *reads* and *writes* and the variables

[2] The exposition here does not deal with substitutions with parameters and results.

appearing in S. Nor do we require that *writes* $\subseteq$ *reads*. Note that variables that can be read and written appear in both *reads* and *writes* clauses.

3 Interpreting the read frame

We find it convenient to present the majority of this discussion in terms of a relational semantics composed of a termination predicate giving those states for which termination is guaranteed and a meaning relation which gives the possible state transitions. We consider first the meaning relation.

For a given substitution (F, R, W, S), we define four relations on the state space giving interpretations for the substitution respecting none, one or other or both frames. Let Σ be the state space spanned by F, then $M_\emptyset$ is the meaning relation not respecting either frame; M_W is the meaning relation which respects the write frame (only); M_R is the meaning relation which respects the read frame (only); and M_{RW} is the meaning relation which respects both read and write frames. For any $(\sigma, \sigma') \in \Sigma \times \Sigma$, we write $\sigma M_\emptyset \sigma'$ for $(\sigma, \sigma') \in M_\emptyset$ and $\sigma M_\emptyset$ for the relational image of σ under $M_\emptyset$ (and respectively for M_R, M_W M_{RW}).

The first two relations, $M_\emptyset$ and M_W, are simple to define. The third, M_R, is the subject of this section. We intend that M_R and M_W can each be defined independently of the other frame and that M_{RW} can be recovered as the intersection of the two: $M_{RW} = M_W \cap M_R$.

We define $M_\emptyset$ in the usual way as the "predicate" of a substitution:

$$M_\emptyset \triangleq \left\{(\sigma, \sigma') \in \Sigma \times \Sigma \middle| prd(S, \sigma, \sigma')\right\}$$

where $prd(S, \sigma, \sigma') \triangleq \neg [S] \neg (\sigma = \sigma')$.

It is a simple matter to extend this definition to incorporate the semantics of the write frame. We simply remove from the meaning relation any transition which changes a value of a non-written variable[3].

$$M_W \triangleq M_\emptyset \cap \Xi_{F-W}$$

The task now remaining is to define the constraint on the meaning relation which interprets the constraint imposed by the read frame.

For a given substitution, the interpretation M_R, which respects the read frame but ignores the write frame, must be equivalent to a read-write respecting interpretation of a similar substitution with the same F, R and S but with a universal write frame. Clearly, this substitution may have a write frame which is bigger than its read frame and so, to make this definition, we will need to

[3] We borrow a notation from Z and, generalising slightly, define for any set of identifiers $S \subseteq F$, a relation on states $\Xi_S \triangleq \{(\sigma, \sigma') : \Sigma \times \Sigma \mid \sigma\lfloor_S = \sigma'\lfloor_S\}$ where $\sigma\lfloor_S$ is the projection of σ onto S.

give an interpretation to variables which are in the write frame but not the read frame.

But how are we to make sense of such "write-only" variables? Although such variables are not normally used in substitutions, and arguably do not correspond to any useful programming concept, perhaps such unrealistic constructs, much like miracles in [Mor88], can play a *part* in defining useful substitutions and so help us to understand the underlying concepts.

The following subsections propose three alternative interpretations for write-only variables and discuss the merits of each. We begin by considering the strictest of the three interpretations.

Must-write semantics The first interpretation we consider is that write-only variables *must* be written by the substitution with a value which depends only on the reads.

Consider a simple assignment, $x := E$, under read frame R where $x \notin R$. Clearly, this substitution can be implemented by $x := e$ where e is an equivalent expression only mentioning variables in R, and in particular x:=c where c is a constant expression not referring to any variables. However, in this case **skip**, which is akin to $x := x$ should be thought of as reading x and thus is not a valid implementation.

To formalise this definition, we consider a state with universal write frame and read frame R. In this case there are two sets of variables, one set is read-write and the other is write-only. In this interpretation, we require that two starting states which differ only in the WO component must have exactly the same possible final states - not just the same RW components. That is, any result state possible from a given start state should also be possible from any other start state that differs only in a write-only component[4] (Figure 1).

$$\forall \sigma_1, \sigma_2 \cdot \sigma_1 \Xi_R \sigma_2 \Rightarrow \sigma_1[S] = \sigma_2[S]$$

Note that this condition forces equality on the whole after states and so, in particular, on those components which are neither read nor written. Therefore, when we reintroduce the write frame, which requires identity on the unwritten components, this leads to infeasibility for initial states which differ on components which are neither read nor written because these must be made equal without changing them. Thus this interpretation of write-only variables is too strong for the orthogonal treatment of reads and writes which we seek.

May-write semantics A second, slightly more liberal, alternative arises if we reintroduce the possibility of skip on the write-only variables. This corresponds more closely to what might be expected to arise from a static analysis of code

[4] Here we consider S as a relation on states and take the relational image.

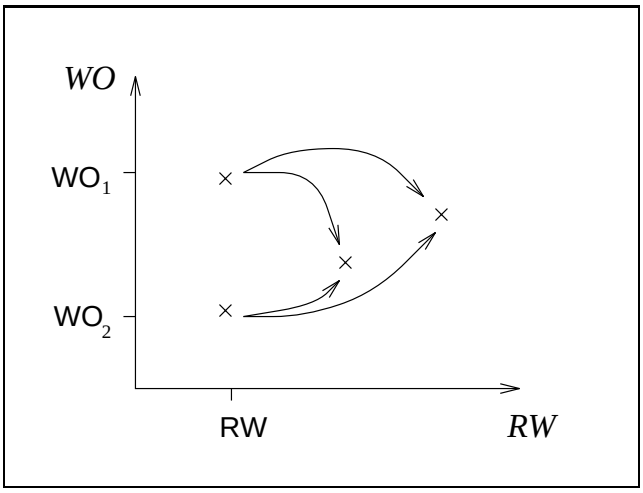

Fig. 1. "Must-write" semantics

ascertaining which variables are read and written. By allowing **skip** on unread variables, this interpretation would allow their final values to be the same as their initial values, but it would not allow their original values to affect the final values in any other case. Thus we call this a *may*-write semantics.

This semantics can be formalised by explicitly adding to the must-write semantics the possibility of no change of value for the unread variables. However, in this case, the formulation is rather unwieldy[5] and so for brevity we simply observe that it doesn't exhibit the property we want when recombined with the write frame constraint.

Non-interference semantics The third and most liberal alternative, not only allows **skip** on unread variables but also allows other statements that, whilst allowing the value of the unread variables to be changed, do not allow of any read variables to depend on any of the unread variables. Thus, the original values of the write-only variables can influence their own final values, and those of other write-only variables, but not the final values of any read variable. Examples of such statements include statements to increment a number, append to, or reverse a list, or even to swap the values in two variables which are both write-only.

This is clearly an information flow condition, and so is appropriate as a basis for defining non-interference between substitutions. For this reason we call it a 'non-interference' semantics.

[5] Roughly speaking, for each transition $\sigma \xrightarrow{\circ} \sigma'$ and for each subset of the write-only variables, WO, we must add a transition of the form $\sigma \xrightarrow{\circ} (\sigma\big|_{WO}, \sigma'\big|_{F-WO})$.

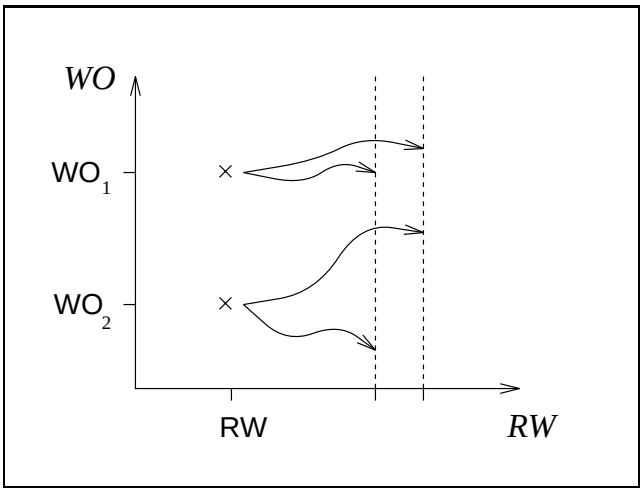

Fig. 2. "Non-interference" semantics

The condition required to yield this interpretation states that the initial values of write-only variables should be able to influence their own final values and those of any other write-only variables, but should not influence the final values of any read variables. For a substitution with universal write frame, this can be pictured as in Figure 2.

Here we see that it would be impossible to deduce the initial values of the write-only variables from the final values of only the read variables. This is formalised by weakening the predicate given above to require merely the existence of some state with the required read component[6].

$$\forall \sigma_1, \sigma_2 \cdot \sigma_1 M_R \sigma_2 \;\Rightarrow\; \sigma_1[S]\Big|_R = \sigma_2[S]\Big|_R$$

It is clear that when we reintroduce the write frame constraint, the freedom on the unwritten variables is constrained to the one value for which the final values are equal to the initial ones[7]. So this interpretation does indeed yield the orthogonality between the interpretations of the two frames that we seek.

$$M_{RW} = M_R \cap M_W$$

It is interesting to note that this constraint as akin to the definition of a (strong) bisimulation from process algebra (eg. [Mil89]). Recall that a relation $\mathcal{S}$ over agents is a (strong) bisimulation if

[6] Note that projection is lifted pointwise to sets of states in the relational image, and that this represents an implicit existential quantification in the set equality: $S_1\big|_R = S_2\big|_R \;\triangleq\; \forall e_1 \in S_1 \cdot \exists e_2 \in S_2 \cdot e_1\big|_R = e_2\big|_R \wedge$ *vice versa.*

[7] To see this note that for $x \in W \subseteq R$ we have ${\sigma'}_1\big|_{\{x\}} = \sigma_1\big|_{\{x\}} = \sigma_2\big|_{\{x\}} = {\sigma'}_2\big|_{\{x\}}$.

"For any agents P and Q such that $(P, Q) \in \mathcal{S}$, and for all actions $\alpha \in Act$ we have:
i) $P \xrightarrow{\alpha} P' \;\Rightarrow\; \exists Q' \cdot Q \xrightarrow{\alpha} Q'$ and $(P', Q') \in \mathcal{S}$, and
ii) $Q \xrightarrow{\alpha} Q' \;\Rightarrow\; \exists P' \cdot P \xrightarrow{\alpha} P'$ and $(P', Q') \in \mathcal{S}$"

Here P, Q, P', Q' correspond to $\sigma_1, \sigma_2, \sigma'_1, \sigma'_2$ and $(A, B) \in \mathcal{S}$ corresponds to $A\Big|_R = B\Big|_R$.

This yields the following theorem

Theorem

The relation M_R is the largest subrelation of $M_\emptyset$ such that Ξ_R is a strong bisimulation on M_R.

The fact that M_R is the *largest* subset of $M_\emptyset$ clearly corresponds to the definition of the weakest strong bisimulation, the existence of which is discussed for example in [Mil89].

This surprising result warrants some further investigation to understand its significance more fully. Consider that Ξ_R is the meaning of a substitution, call it $write_{F-R}$, which is the least refined substitution which has F-R as write frame. Clearly, the above theorem indicates a relationship between any substitution that respects a read frame R and $write_{F-R}$.

For any substitution S_R with read frame R, since S_R does not read the values of variables in $F-R$, and $write_{F-R}$ only affects those variables, whatever behaviour was possible for S_R from a given state, will also be possible if preceded by the execution of $write_{F-R}$. The resulting states in the two cases again being equal on $F-R$.

This is formalised by the condition[8]

$$S_R ; write_{F-R} \sqsubseteq write_{F-R} ; S_R$$

In this light, the bisimulation condition can be recast in terms of relational algebra. Writing M_α for the relation $\{(P, P') \mid P \xrightarrow{\alpha} P'\}$, and writing $\mathcal{S}$ and M_α infixed, the definition of bisimulation can be written as (Figure 3)

$$\forall \alpha \in Act \cdot \; S^{-1} ; M_\alpha \subseteq M_\alpha ; S^{-1} \quad \wedge \quad S ; M_\alpha \subseteq M_\alpha ; S$$

Note that if S is symmetric (as is the case for Ξ_R), then we have $S = S^{-1}$, so either conjunct can be dropped from the body of the quantification without effect.

[8] One should not be misled into believing that this containment is in fact an equality, for in the presence of an invariant the necessary intermediate state may not be valid.

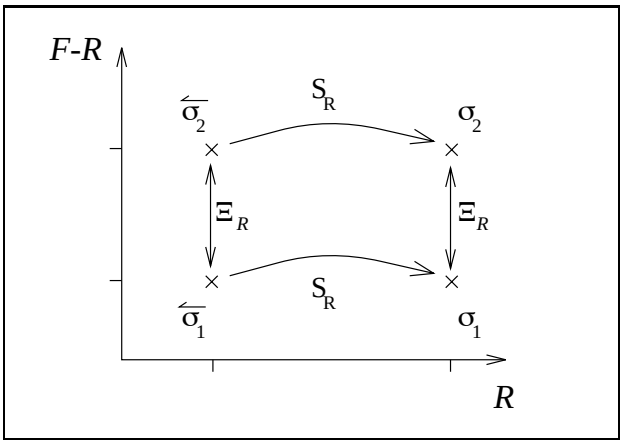

Fig. 3. Non-interfernce as a bisimulation

In our case, we have only one action defined by M. So, we finally achieve a concise characterisation of the condition for not reading outside R:

$$readsM(R, M) \quad \triangleq \quad \Xi_R ; M \subseteq M ; \Xi_R$$

Note that this definition is independent of W and so, unlike the write frame constraint, can be made without a closed world assumption.

3.1 Termination and read frames

Thus far, we have ignored the termination set in the semantic model. However, if the frames are to be interpreted as advanced information about the state accesses of implementing code, and if this code is to respect the read frame, then this will be reflected in the termination set of the substitution.

As might be expected, the criterion for read-respecting for a set is similar, but simpler, than that for a relation. It amounts to saying that a set T respects the frame R if it is a cylinder in the state space (Figure 4).

which can be formalised as a constraint on the relational image of T under Ξ_R:

$$readsT(R, T) \quad \triangleq \quad \Xi_R \, (\!|\, T \,|\!) \subseteq T$$

4 Interpreting Substitutions with read and write frames

We have given an interpretation of read frames and have related this to the established concept of bisimulation. We now recombine this interpretation with the

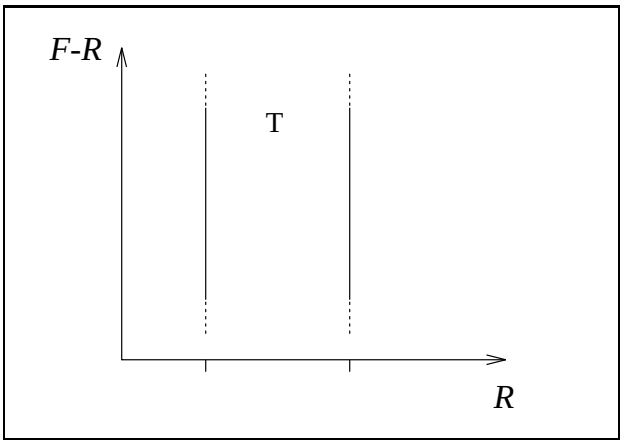

Fig. 4. The termination set must be a cylinder in the state space

interpretation of the other components of the substitution to define a semantic model for substitutions.

In order to prepare the ground for the richer semantic model that will be used in the treatment of refinement later, we take a slightly indirect route to defining the substitution semantics. Rather than defining T and M directly from the components of the substitution definition, we consider the (T, M) for all the valid refinements of the substitution and take the most general of these.

Again consider a fixed substitution $\mathsf{S} = (F, R, W, S)$. Below, we identify three separate compliance conditions for a pair $(T, M) : \mathcal{D}_o$.

subst which ensures that the termination set and meaning relation interpret the substitution:

$$subst_{(F,R,W,S)}(T, M) \quad \triangleq \quad T \supseteq [S]\mathbf{true} \quad \wedge \quad M \subseteq \neg\,[S]\neg\,(\sigma = \sigma')$$

writes which ensures that the write frame is adhered to

$$writes_{(F,R,W,S)}(T, M) \quad \triangleq \quad M \subseteq \Xi_{F-W}$$

and

reads which ensures the same for the read frame

$$reads_{(F,R,W,S)}(T, M) \quad \triangleq \quad \Xi_R ; M \subseteq M ; \Xi_R \quad \wedge \quad \Xi_R \,(\!|\, T \,|\!) \subseteq T$$

We define the semantics of a substitution to be the most general (T, M) pair which satisfies these three conditions. First we define the set of all the interpretations satisfying the three conditions

$$\mathcal{S} = \left\{ (T, M) : \mathcal{D}_o \;\middle|\; \begin{array}{l} subst_{(F,R,W,S)}(T, M) \\ \wedge\ writes_{(F,R,W,S)}(T, M) \\ \wedge\ reads_{(F,R,W,S)}(T, M) \end{array} \right\}$$

Then we take the unique, most general of these pairs as the interpretation of the substitution[9]

$$[\![(F, R, W, S)]\!]_0 \quad \triangleq$$
$$\iota(T, M) \in \mathcal{S} \cdot \forall (T_i, M_i) \in \mathcal{S} \cdot T \subseteq T_i \wedge M \supseteq M_i$$

Existence and uniqueness of such a (T, M) pair is demonstrated in the next section.

4.1 Non-interference

We can now give our first non-interference result which we state without proof as it is superseded by a stronger result later.

Theorem For two substitutions over the same alphabet: $\mathsf{S}_i = (F, R_i, W_i, S_i)$,

$$\frac{R_1 \supseteq W_1 \wedge R_2 \supseteq W_2 \qquad R_1 \cap W_2 = \{\} = R_2 \cap W_1}{[\![\mathsf{S}_1 \,||\, \mathsf{S}_2]\!]_0 = [\![\mathsf{S}_1 ; \mathsf{S}_2]\!]_0 = [\![\mathsf{S}_2 ; \mathsf{S}_1]\!]_0}$$

However, as stated earlier, it is not necessarily the case that non-interference is preserved by usual definition of refinement.

5 Refinement with Read Frames

As discussed above, the usual semantic model for substitutions is not rich enough to ensure the preservation of read-respecting behaviour by refinement. One approach to resolving this difficulty is to restrict the definition of refinement so that it is by the read frame. An alternative approach is to define a richer semantic model for specifications where substitutions are interpreted as the set of their valid refinements and this set is filtered by the read constraint. In effect, we encode the read frame into the semantics of the substitution as "advanced

[9] The suffix in $[\![\ ]\!]_0$ is used to distinguish this interpretation from that defined in the next section.

information" about the valid refinements. With this interpretation, a simple definition of refinement, as containment of implementations, does respect the read frame. This section gives such a semantics for substitutions and refinement.

5.1 Semantics for substitutions with refinement

Consider again a fixed substitution $\mathsf{S} = (F, R, W, S)$ and, taking the three conditions defined above, instead of defining the semantics of a substitution to be the most general (T, M) pair which satisfies these conditions, we simply take the set itself as the semantics[10].

$$[\![(F, R, W, S)]\!]_1 \quad \triangleq \quad \left\{ (T, M) : \mathcal{D}_o \;\middle|\; \begin{array}{l} \mathit{subst}_{(F,R,W,S)}(T, M) \\ \wedge\ \mathit{writes}_{(F,R,W,S)}(T, M) \\ \wedge\ \mathit{reads}_{(F,R,W,S)}(T, M) \end{array} \right\}$$

5.2 Semantics of Refinement

With this model, refinement is simply set containment

$$\mathsf{S} \sqsubseteq_1 \mathsf{T} \quad \triangleq \quad [\![\mathsf{S}]\!]_1 \supseteq [\![\mathsf{T}]\!]_1$$

5.3 Consistency

The semantics of Section 4 is thus a "retrieval" of this model defined by

$$\begin{array}{l} \mathit{retr} : \mathcal{D}_1 \to \mathcal{D}_0 \\ \mathit{retr}(\mathcal{S}) \quad \triangleq \quad (\bigcap_{(T_i, M_i) \in \mathcal{S}} T_i \;,\; \bigcup_{(T_i, M_i) \in \mathcal{S}} M_i) \end{array}$$

then, for any substitutions S, we have $[\![\mathsf{S}]\!]_0 = \mathit{retr}([\![\mathsf{S}]\!]_1)$ and refinement in $\mathcal{D}_1$ is a sufficient condition for refinement in $\mathcal{D}_0$

$$\frac{\mathsf{S}_a \sqsubseteq_1 \mathsf{S}_c}{\mathsf{S}_a \sqsubseteq_0 \mathsf{S}_c}$$

where $\sqsubseteq_0$ is satisfaction in the $\mathcal{D}_0$ semantics defined in the usual way.

Note that the reverse is not the case since the new definition of refinement prohibits putative refinements which do not respect the read frame.

To see that these definitions are well-defined we need to show existence and uniqueness of such a (T, M) pair and preservation of refinement between the models. It is also necessary to show monotonicity of each substitution constructor with respect to refinement[BvW98,BB98].

[10] The suffix in $[\![\ ]\!]_1$ is used to distinguish this interpretation from that defined in the previous section.

Lemma For $\mathcal{S}$ defined as above we have

$$\exists!\,(T, M) \in \mathcal{S} \cdot \forall (T_i, M_i) \in \mathcal{S} \cdot T \subseteq T_i \wedge M \supseteq M_i$$

Proof Existence within $\mathcal{D}_o$ and uniqueness within $\mathcal{S}$ are both trivial on constructing $[\![\mathsf{S}]\!]_0 = retr([\![\mathsf{S}]\!]_1)$. To show "closure", that the pair constructed above is in the set $\mathcal{S}$, we must show that (T, M) satisfies the three properties defining $\mathcal{S}$. Each property follows easily from the assumption that all $(T_i, M_i) \in \mathcal{S}$ by various distributive properties of **dom** and $\lhd$ over $\subseteq$ and $\cup$. It is interesting to note that the proof of each property relies only on the assumption of the respective property for the (T_i, M_i).

The proof that *retr* preserves refinement is straightforward. In this setting, with refinement defined as set inclusion, monotonicity is also trivial since constructing a set comprehension over smaller sets clearly leads to smaller constructed set.

Miracles Note that when no implementation is possible then $[\![\mathsf{S}]\!]_1$ is empty. This corresponds to miracle [Mor88] since

$$retr(\{\,\}) = (\bigcap_{\{\}} , \bigcup_{\{\}}) = (\overline{\mathbf{true}}, \overline{\mathbf{false}}) = [\![\mathbf{miracle}]\!]_0$$

Incompleteness Note that this semantics does not address issues arising from non-essential read and write accesses. For example, if y is not read, then the semantics will not allow $x \in \{0, 1\}$ to be refined by $x := \mathbf{if}\ y = 0\ \mathbf{then}\ 0\ \mathbf{else}\ 1$, whilst the refinement $x := \mathbf{if}\ y = 0\ \mathbf{then}\ 0\ \mathbf{else}\ 0$ is permitted since this is equivalent to $x := 0$. Thus the semantics does admit refinements which syntactically break the read and write frame provided that they are equivalent to substitutions that do respect them. On the other hand, proof rules which give syntactic sufficient conditions will prohibit refinements which break the frames even if such accesses have no effect on the resulting behaviour. This semantics will therefore not be complete with respect to such rules.

6 Examples

In order to illustrate properties of the above semantics, we consider some examples including those given in Section 1.1.

6.1 Filtering of refinements

We first consider an extremely simple example which demonstrates the result of filtering the refinements in the semantics of a substitution. Consider two variables

x and y such that $x, y \in \{0,1\}$ and the substitution $(\{x, y\}, \{x\}, \{y\}, y{:=}\,0 [] y{:=}\,1)$.

We calculate the most general (T, M) to be $(\{(0,0),(0,1),(1,0),(1,1)\}, \{(0,0) \mapsto (0,0), (0,0) \mapsto (0,1), (0,1) \mapsto (0,1), (0,1) \mapsto (0,0), (1,0) \mapsto (1,0), (1,0) \mapsto (1,1), (1,1) \mapsto (1,1), (1,1) \mapsto (1,0)\})$ as illustrated in the top circle of the following diagram. It has eight transitions and four points of non-determinacy. Note this relation satisfies the read frame constraint for $\{x\}$.

If the read frame were ignored, there would be eighty-nine different refinements of this substitution arising by reducing the non-determinacy by removing one, two, three, or four transitions whilst maintaining the totality of the substitution. (There are eight interpretations with seven transitions, forty-eight with six, twenty-four with five, and eight with four.) However, if we 'filter' these interpretations by the read frame constraint, we have to remove transitions in read respecting pairs, and so we remove all but eight of the subrelations. The ones remaining correspond to combinations of the interpretations of $y{:=}\,0$, $y{:=}\,1$, $y{:=}\,x$ and $y{:=}\,1\text{-}x$. So, for example, we have that $\mathsf{S} \sqsubseteq y{:=}\,1\text{-}x$ but not by $y{:=}\,1\text{-}y$ or by *skip*. This illustrates how an explicitly stated read frame can restrict the valid implementations even when the substitution itself respects the read frame.

We now revisit the examples given in Section 1.1

6.2 Read frames and non-interference

We are now in a position to prove the stronger form of the non-interference result from Section 4.1 which states that non-interference is preserved by refinement.

For two substitutions in the same context $\mathsf{S}_i \triangleq (F, R_i, W_i, S_i),\ i = 1, 2$, we have:

$$\frac{R_1 \cap W_2 = \{\,\} = R_2 \cap W_1 \qquad \mathsf{S}_i \sqsubseteq_1 \mathsf{T}_i}{\mathsf{T}_1;\mathsf{T}_2 = \mathsf{T}_2;\mathsf{T}_1 = \mathsf{T}_1 \parallel \mathsf{T}_2}$$

The proof, which is reasonably straightforward, relies on the four 'frame' properties of the semantics

$$\mathit{readsM}(R_1, M_1),\ \mathit{writes}(W_1, M_1),\ \mathit{readsM}(R_2, M_2),\ \mathit{writes}(W_2, M_2)$$

the non-interference conditions in a slightly different form

$$F - W_2 \supseteq R_1 \text{ and } F - W_1 \supseteq R_2$$

and on the 'global' conditions

$$R_1 \supseteq W_1 \text{ and } R_2 \supseteq W_2$$

The first example in Section 1.1 is simply one half of this result with the identity refinement.

It could reasonably be argued that non-interference should hold provided the write frame of each substitution does not intersect either frame of the other substitution, irrespective of whether it is contained in its own read frame. We later discuss (Section 7.1) a slight strengthening of the reads constraint which yields this stronger form of the non-interference result.

6.3 Read frames and initialisation

An initialisation is simply a substitution with empty read frame. For example, if we want to initialise to any state satisfying the invariant, we can simply give the substitution $(F, \{\}, F, \mathit{skip})$. The usual proofs for maintaining the invariant together with the requirement not to depend on the old value of any variable will ensure the invariant is established by the initialisation. Of course, skip itself does not now respect the read frame and must be implemented by code which does.

Even if the abstract initialisation mentions a variable on the right hand side of an assignment, we would require that by the time the initialisation is implemented the dependence on that variable is removed. For example, for invariant $x = y$, we can give the initialisation as $(F, \{\}, F, x := y)$ which would then need to be implemented by something like $x := c; y := c$.

6.4 Read frames and encapsulation

It is a simple matter to specify that y is not used in the implementation of a substitution manipulating x by excluding y from the read frame. This can be used to suggest a decomposition of a specification without bringing in structuring mechanisms which ensure full hiding. Furthermore, such techniques may be a useful basis to define a form of structuring for *abstract* machines which employs full hiding and so permits separate development. (See Section7.4.)

6.5 Read frames and underspecification

Using the read frame it is a simple matter to specify which variables can be used to resolve any looseness in an abstract substitution and so we can remove some uses of constants in specifications.

6.6 Read frames and refinement

In all of these examples, read-respecting refinement ensures that non-reading behaviour of an abstract substitution is preserved during refinement.

We now give some proof rules which give sufficient conditions for read-respecting refinement.

Strengthening the read and write frames We can contract the read and write frames in refinement[11]

$$\frac{R_1 \supseteq R_2 \supseteq W}{(F, R_1, W, S) \sqsubseteq (F, R_2, W, S)} \qquad \frac{W_1 \supseteq W_2}{(F, R, W_1, S) \sqsubseteq (F, R, W_2, S)}$$

The proof of the latter is straightforward on noting that $W_1 \supseteq W_2 \;\Rightarrow\; \Xi_{F-W_1} \subseteq \Xi_{F-W_2}$.

To prove the former, we must show

$$\frac{\begin{array}{c} R_1 \supseteq R_2 \\ \Xi_{R_2}; M \subseteq M; \Xi_{R_2} \end{array}}{\Xi_{R_1}; M \subseteq M; \Xi_{R_1}}$$

which can be shown provided $M \subseteq \Xi_{F-W}$ since $R_1 \supseteq R_2 \;\Rightarrow\; \Xi_{R_1} \subseteq \Xi_{R_2}$ and $M \subseteq \Xi_{F-W} \subseteq \Xi_{R_1-R_2}$. This final proviso explaining the presence of the last containment in the hypothesis of the rule.

As for non-interference result given above, one might expect the contract reads rule to be valid irrespective of the write frame. Again, with the current definition *reads*, this is not the case. In order to weaken the hypothesis to allow the introduction of write-only variables we must take a stronger definition of *reads* as discussed in the next section.

Strengthening the substitution We can strengthen the body of a substitution provided the new body does not introduce a new read dependency. Take for example guarded substitutions. Recall that in the standard semantics, we have $S \sqsubseteq G \Longrightarrow S$ for all substitutions S and guards G. The proof of this is straightforward from the definition of the weakest precondition of guarded substitution as $[G \Longrightarrow S]Q \;\;\triangleq\;\; G \;\Rightarrow\; [S]Q$.

To give a similar rule for read-respecting refinement, we must additionally show that $\Xi_R; M_{G \Longrightarrow S} \subseteq M_{G \Longrightarrow S}; \Xi_R$. On expanding $M_{G \Longrightarrow S}$ we are required to prove $\Xi_R; (G \wedge prd(S)) \subseteq (G \wedge prd(S)); \Xi_R$ which, on expanding the definition of sequence, follows provided $\Xi_R \,(\!|\, G \,|\!) \subseteq G$. This is the definition given of G read respects R. Thus we have

$$\frac{\Xi_R \,(\!|\, G \,|\!) \subseteq G}{(F, R, W, S) \sqsubseteq (F, R, W, G \Longrightarrow S)}$$

Similar rules can be defined for other constructs.

[11] The reader familiar with [Dun02] may at first be surprised by this as Dunne's approach allows expansion rather than contraction of write frames in refinement. The difference is not fundemental, it is simply due to the fact that different syntax is being used for the same concept. Dunne's conservative frame expansion conserves the values of the variables introduced to the frame by the frame expansion, whereas, here we conserve the variables outside the specified frame.

7 Further work and conclusions

7.1 Introducing write-only variables

It is interesting to note that the proof of non-interference and of soundness for contracting the read frame requires that the reads should contain the writes for each substitution. This is perhaps surprising as it is reasonable to expect that a substitution reading fewer variables than permitted should be a valid refinement whatever the write frame might be. Furthermore, these is the only places where the orthogonality of the treatment of the *reads*, *writes* and *subst* is broken.

It turns out that with a slight adjustment to the definition of the read constraint, we can remove the requirement that the reads contain the writes. We record here without detailed justification a stronger version of the read predicate which exhibits this cleaner property.

$$reads_{wo}(R, M) \triangleq \underline{\triangle} \forall S \supseteq R \cdot \Xi_S ; M \subseteq M ; \Xi_S$$

An informal understanding of this condition is that it does not allow information flow between the unread variables so read frame can be contracted within the writes without concern.

This interpretation yields a non-interference result for relations without recourse to the assumptions such as $R_i \supseteq W_i$. We give the result for arbitrary M_1 and M_2, for its use in practice, one of the M_i will be Ξ_{R_i}.

$$\frac{\begin{array}{c} reads_{wo}(R_1, M_1) \wedge writes(W_1, M_1) \\ reads_{wo}(R_2, M_2) \wedge writes(W_2, M_2) \\ W_1 \cap (R_2 \cup W_2) = \{\} = W_2 \cap (R_1 \cup W_1) \end{array}}{M_1 ; M_2 = M_2 ; M_1}$$

7.2 Weakest precondition semantics

It is not straightforward to present the concepts developed here in a standard weakest precondition framework[Dij76]. Rather, a more convenient form for a weakest precondition semantic model of these concepts is to give the sets of identifiers comprising the read and write frames directly as components of the semantics. Thus $[\![S]\!]$ is a triple $(R, W, [S])$ where $[S]$ is the usual weakest precondition semantics of S. This would be a relatively simple extension to the approach taken by Dunne in defining a semantics for substitutions with write frames as a pair $(W, [S])$.

7.3 Proof rules for read-respecting refinement

We have presented the key concepts required to interpret the fact that a substitution does not read outside a given frame of variables, and given a definition

of refinement which respects this property. We have given a few examples of proof rules which give sufficient conditions for read respecting refinement. There is obviously more work to be done to give a comprehensive suite of proof rules which are sound with respect to this semantics.

7.4 Specification structuring and read-respecting refinement

There is also considerable work to be done to build these concepts into a useful development methodology which exploits read and write frames fully in the structuring of specifications. This work could be progressed in two directions. Firstly, the new semantics could be used to justify semantically the visibility rules in B's existing machine structuring mechanisms, and secondly, perhaps new structuring mechanisms could be found which give a more orthogonal treatment of horizontal and vertical structuring of developments. One might hope that with this would come more intuitively obvious visibility rules between components.

We have presented semantic conditions interpreting the concept that the outcome of a substitution does not depend on the value of variables outside a "read frame". We have taken care to ensure that this interpretation of the read frame is orthogonal to the interpretation of the write frame. This semantics supports the formalisation, at specification time, of implementation constraints previously only captured by visibility rules implicit in certain forms the structuring of specifications.

We have given some example proof rules for refinement which ensure that the read-respecting behaviour of an abstract substitution is preserved by its implementation. A more comprehensive treatment of such proof rules for the language of substitutions is still to be undertaken. We believe that such a treatment could lead to a more expressive and compositional development method with more orthogonal specification structuring constructs.

References

[Abr95] J-R. Abrial. *The B-Book.* Cambridge University Press, 1995.

[BB98] R.J.R. Back and M.J. Butler. Fusion and simultaneous substitution in the refinement calculus. *Acta Informatica*, 35(11):921–940, 1998.

[Bic93] J.C. Bicarregui. Algorithm refinement with read and write frames. In J.C.P. Woodcock and P.G. Larsen, editors, *Proceedings of Formal Methods Europe '93*, volume 670 of *Lecture Notes in Computer Science*. Springer-Verlag, 1993.

[Bic94] J.C. Bicarregui. Operation semantics with read and write frames. In *Sixth Refinement Workshop*, Workshops in Computer Science. Springer-Verlag, 1994.

[Bic95] J.C. Bicarregui. *Intra-Modular Structuring in Model-oriented Specification: Expressing non-interference with read and write frames.* PhD thesis, Manchester University, Computer Science, June 1995. UMCS-95-10-1.

[BvW98] R.J.R. Back and J. von Wright. *Refinement Calculus: a Systematic Introduction.* Springer-Verlag, 1998.

[Dij76] E.W. Dijkstra. *A Discipline of Programming.* Prentice Hall, 1976.

[Dun02] S. Dunne. A theory of generalised substitutions. In D. Bert et al, editor, *Proceedings of ZB2002.* Springer Verlag, LNCS 2272, 2002.

[Jon86] C.B. Jones. *Systematic Software Development using VDM.* Prentice Hall, 1986.

[Jon87] C.B. Jones. VDM proof obligations and their justification. In M. Mac an Airchinnigh D. Bjørner, C.B. Jones and E.J. Neuhold, editors, *Proceedings of VDM '87*, volume 252 of *Lecture Notes in Computer Science.* Springer-Verlag, 1987.

[Mil89] R. Milner. *Communication and Concurrency.* Prentice Hall, 1989.

[Mor88] C. Morgan. Data refinement by miracles. *Inf. Proc. Letters*, 26(5), Jan. 1988.