SOFTWARE VERIFICATION RESEARCH CENTRE

SCHOOL OF INFORMATION TECHNOLOGY

THE UNIVERSITY OF QUEENSLAND

Queensland 4072
Australia

TECHNICAL REPORT

No. 00-01

Refinement and State Machine Abstraction

Karl Lermer and Paul Strooper [a]

February 2000

Phone: +61 7 3365 1003
Fax: +61 7 3365 1533
http://svrc.it.uq.edu.au

Note: Most SVRC technical reports are available via anonymous ftp, from `svrc.it.uq.edu.au` in the directory `/pub/techreports`. Abstracts and compressed postscript files are available via `http://svrc.it.uq.edu.au`

Refinement and State Machine Abstraction

Karl Lermer and Paul Strooper [b]

Abstract

Precise module interface specifications are essential in modular software development. The role of state in these specifications has been the issue of some debate and is central to the notion of data refinement. In previous work, Hoffman and Strooper introduce a state-abstraction lattice that defines a partial order on specifications for deterministic and complete languages. They use this lattice to define a notion of state abstractness and show that this intuitive notion corresponds to the use of the terms "abstract" and "concrete" as used in data-refinement proofs. In this paper, we extend this work for a class of specifications and languages that we call demonic and semi-deterministic. We also introduce a notion of backward refinement and prove that backward refinement together with the common forward refinement of VDM and Z form a sound and complete refinement technique with respect to a partial order on languages defined by demonic specifications. We illustrate the ideas using simple languages and specifications.

Keywords: formal methods, state machine, specification, refinement, modular software development, abstraction.

1. Introduction

The key idea in modular software development is to decompose a system into a small number of modules, and to continue doing so until all modules are of a reasonable complexity. This approach relies on being able to subdivide the system into relatively independent modules, and being able to precisely specify the interfaces between these modules.

We would like our specifications to be "black box." The internals of the module are of no concern to us; only the externally observable behaviour, in the form of inputs and outputs, is relevant. However, the notion of state is important in such specifications, because past inputs typically influence future outputs, which can only be attributed to a difference in state.

In this paper, we build on work by Hoffman and Strooper [9], who define a notion of state abstractness for specifications and introduce a state-abstraction lattice to characterise refinement proofs with abstraction functions. Their work applies to specifications of languages that are *complete* — every sequence of calls has a behaviour defined for it — and *deterministic* — there is at most one behaviour defined for every sequence of calls. Hoffman and Strooper show that state abstractness is in general independent of the choice between property-based, model-based, and operational specifications. Although some people object to using the notion of "state" for a property-based specification, equivalence

classes of traces can serve as a reasonable notion of state for such specifications [9]. In this paper, we explore the relation between the abstraction lattice and data-refinement proofs.

Data-refinement proofs [6,2,12,17] are important in modular software development. They are used to verify that a specification (or implementation) S^C with a concrete state representation is correct with respect to a specification S^A with an abstract state representation. The state-abstraction lattice justifies the terms *abstract* and *concrete* used in this setting: if there exists a data-refinement proof using an abstraction function that proves that S^C is correct with respect to S^A, then $R_{S^C} \preceq R_{S^A}$ in the state-abstraction lattice (where, for a specification S, R_S is an equivalence relation defined on S). The contrapositive of this result tells us that if $R_{S^C} \not\preceq R_{S^A}$ in the state-abstraction lattice, then we cannot prove that S^C is correct with respect to S^A using a standard data-refinement proof with an abstraction function.

In the remainder of this paper, we will generalise these results for languages and specifications that are not necessarily complete and deterministic. In particular, we introduce the notions of demonic languages and specifications, and semi-deterministic specifications, and we show how the results extend for these specifications and languages. In doing so, we define an ordering on demonic languages that provides a refinement semantics on demonic specifications. We prove that the common forward refinement notion of VDM and Z [12,18,17] together with an adequate backward refinement notion form a sound and complete proof technique with respect to this refinement semantics.

In Section 2, we present our terminology for languages and specifications. Section 3 formally defines the restrictions that Hoffman and Strooper place on the languages and specifications in their work. We also introduce the notions of demonic languages and specifications, and semi-deterministic specifications. In Section 4, we state the VDM and Z notion of forward refinement. We also introduce a partial order on demonic languages, and show that forward refinement is a sound proof technique with respect to this ordering. The completeness of forward refinement is proven for demonic, semi-deterministic specifications. Section 5 generalises the state-abstraction lattice for demonic and semi-deterministic specifications and languages. The mathematical structure that we use to capture state abstractness is no longer a lattice in this case, but simply a partially ordered set. In Section 6 we introduce a backward refinement technique and prove that forward and backward refinement together are sound and complete with respect to the refinement semantics given in Section 4. In Section 7, we review related work. In particular, we explain how our notion of refinement relates to the VDM and Z notion of data refinement [12,17], and how it relates to forward and backward simulation of state machines [8,10,11,15].

The appendices contain the definitions of a number of languages and specifications that we use as examples throughout the paper. Each appendix first defines a language informally, and then presents one or more Object-Z specifications [4] for that language. We have used Object-Z merely because it provides a convenient structuring notation for the types of modules and specifications that we consider in this paper. We do not use any of the object-oriented features of Object-Z. The languages and specifications are clearly contrived — they simply serve to illustrate the concepts introduced in the paper.

2. Languages and specifications

Following Parnas [16], we define a *module* as a programming work assignment, and a *module interface* as the set of assumptions that programmers using the module are permitted to make about its behaviour. An *interface specification* (hereafter just *specification*) is a statement of these assumptions. We view a module as a black box, accessible only through a fixed set of *operations* — the exported procedures and functions. The *syntax* of the specification states the names of the access routines, and their inputs and outputs. We use Op to denote the set of all operation names, In to denote the set of all inputs for Op, and Out to denotes the set of all outputs. We use the special symbol $\perp$ to indicate an operation with no input or no output.

The *semantics* of the specification describes the observable behaviour of the operations. We are interested in comparing the behaviour of different specifications. Because there are many ways to represent the state in a specification, we need a definition of behaviour that is independent of the state representation. We first consider *histories*: finite, possibly empty sequences of the form

$$h = \langle c_1, v_1\rangle\langle c_2, v_2\rangle \ldots \langle c_n, v_n\rangle$$

For $i \in \{1, ..., n\}$, $c_i = \langle \iota_i, op_i\rangle$ is a call to an operation $op_i \in Op$ with input $\iota_i \in In$, and $v_i \in Out$ is an output. We use the symbol ε to denote the empty history.

2.1. Languages

The set of all histories, $\mathcal{H}$, is determined by Op, In, and Out. A language $\mathcal{L}$ is defined as a subset of $\mathcal{H}$. In this paper, we only consider non-empty languages that are *prefix-closed*: for any history $h \in \mathcal{L}$ and any call-value pair $\langle c, v\rangle$, if $h\langle c, v\rangle \in \mathcal{L}$ then $h \in \mathcal{L}$.

$$\mathcal{L}an_{\mathcal{H}} = \{\mathcal{L} \subseteq \mathcal{H} \mid \mathcal{L} \neq \varnothing \wedge \mathcal{L} \text{ is prefix-closed}\}$$

Note that this is quite a natural restriction and that it implies that $\varepsilon \in \mathcal{L}$ for all languages in $\mathcal{L}an_{\mathcal{H}}$.

We introduce the following operators on histories. For any history

$$h = \langle c_1, v_1\rangle\langle c_2, v_2\rangle \ldots \langle c_n, v_n\rangle$$

we denote the corresponding *trace* or *input sequence* by

$$\mathcal{I}(h) = c_1 c_2 \ldots c_n$$

We define $\mathcal{I}(\varepsilon) = \varepsilon$. For a set of histories $H \subseteq \mathcal{H}$, we define the set of all traces of H by

$$Tr(H) = \{\mathcal{I}(h) \mid h \in H\}$$

For a language $\mathcal{L}$ and a trace $t \in Tr(\mathcal{H})$, we collect all possible histories (in $\mathcal{L}$) with trace t in the set

$$\nabla_{\mathcal{L}}(t) = \{h \mid h \in \mathcal{L} \wedge \mathcal{I}(h) = t\}$$

Note that for each $h \in \mathcal{L}$, $h \in \nabla_{\mathcal{L}}(\mathcal{I}(h))$. For languages $\mathcal{L}$, $\mathcal{L}' \subseteq \mathcal{H}$,

$$\nabla_{\mathcal{L}\cup\mathcal{L}'}(\cdot) = \nabla_{\mathcal{L}}(\cdot) \cup \nabla_{\mathcal{L}'}(\cdot)\ ,\ \nabla_{\mathcal{L}\cap\mathcal{L}'}(\cdot) = \nabla_{\mathcal{L}}(\cdot) \cap \nabla_{\mathcal{L}'}(\cdot)$$

We will also use the following operators on finite sequences $\sigma = s_1 s_2 ... s_n$: $\sigma[i] = s_i$, $head(\sigma) = s_1$, $front(\sigma) = s_1, ..., s_{n-1}$, $last(\sigma) = s_n$, and $\sigma \rhd \{1, ..., m\}$ the restriction of σ to $\{1, ..., m\}$. Finally, we use $\#S$ to denote the size or length of a set or sequence S.

2.2. Specifications

A specification S defines a language $\mathcal{L}$ — the subset of $\mathcal{H}$ expressing the behaviour defined by the specification. In general the form of the specification may vary, but in this paper we focus on *model-based* specifications, where the behaviour is specified in terms of a state space St.

Definition 1 *A model-based specification S is a six-tuple*

$$(Op, St, In, Out, \textit{INIT}, \_^S)$$

with operation set Op, state set St, input set In, output set Out, a nonempty set of initial states $\textit{INIT} \subseteq St$, and an interpretation function

$$\_^S \ : \ Op \to \mathbb{P}\,((In \times St) \times (St \times Out))$$

Note that Op, St, In, Out, $\textit{INIT}$ are permitted to be infinite sets. Any operation $op \in Op$ is interpreted via $\_^S$ as a set of pairs

$$(\langle \iota, s \rangle, \langle s', \omega \rangle)$$

where each pair represents a state transition with input $\iota \in In$, internal states $s, s' \in St$ (s denotes the state before and s' the state after the operation is performed), and output $\omega \in Out$. For a specification S, the *precondition* of operation $op \in Op$ with input $\iota \in In$ will be denoted by

$$pre_S(\langle \iota, op \rangle) = \{ s \in St \mid \exists\, s' \in St, \omega \in Out \ : \ (\langle \iota, s \rangle, \langle s', \omega \rangle) \in op^S \}$$

and the *postcondition* of op with input ι by

$$post_S(\langle \iota, op \rangle) = \{ s' \in St \mid \exists\, s \in St, \omega \in Out \ : \ (\langle \iota, s \rangle, \langle s', \omega \rangle) \in op^S \}$$

Similarly, we define the *postcondition* of a trace $t \in Tr(\mathcal{H})$ by

$$ptrace_S(t) = \begin{cases} \textit{INIT} & \text{if } t \text{ is the empty trace} \\ \{ s' \in St \mid & \exists\, s \in St, \omega \in Out \ : \\ & (\langle \iota, s \rangle, \langle s', \omega \rangle) \in op^S \wedge s \in ptrace_S(t_1) \} \\ & \text{if } t = t_1 \langle \iota, op \rangle \end{cases}$$

Note that in our setting, pre- and postconditions denote sets of states, not predicates.

Given a specification S and a history $h \in \mathcal{H}$, we denote the set of *final states* of h by

$$final_S(h) = \begin{cases} \textit{INIT} & \text{if } h = \varepsilon \\ \{ s' \in St \mid & \exists\, s \in St \ : \ (\langle \iota, s \rangle, \langle s', \omega \rangle) \in op^S \wedge s \in final_S(h_1) \} \\ & \text{if } h = h_1 \langle \langle \iota, op \rangle, \omega \rangle \end{cases}$$

Note that $final_S(h) \subseteq ptrace_S(\mathcal{I}(h))$ for all histories $h \in \mathcal{H}$ and

$$ptrace_S(t) = \cup \{ final_S(h) \mid h \in \mathcal{L}_S \wedge \mathcal{I}(h) = t \}$$

for all traces $t \in Tr(\mathcal{H})$.

We can now define the language accepted by a specification S, consisting of the empty history and all histories that are produced by starting from an initial state in $\textit{INIT}$ and recursively applying the operations from Op.

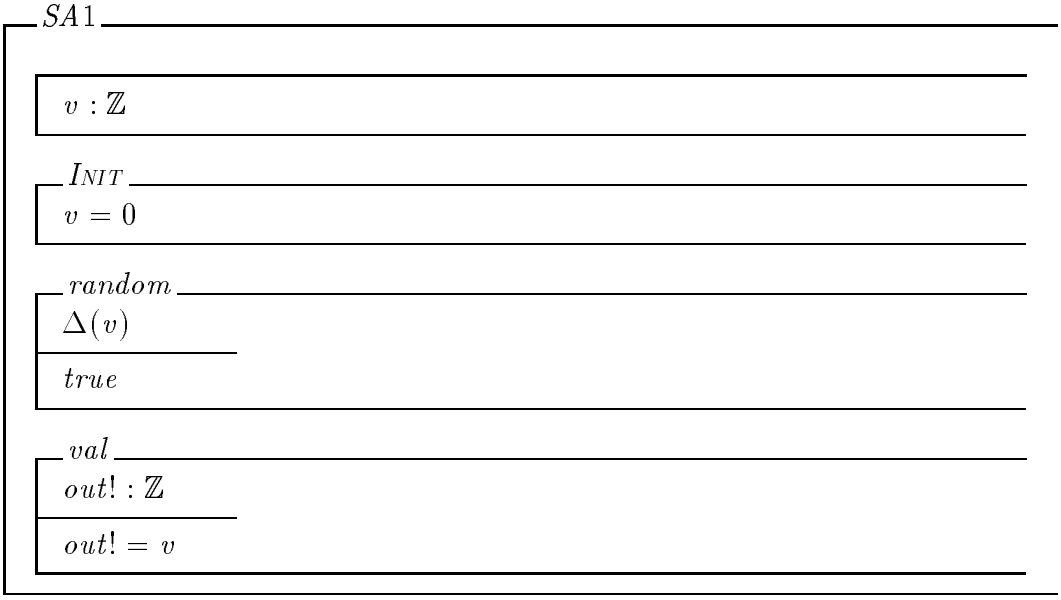


Figure 1. Object-Z specification $SA1$ for $\mathcal{L}_A$

Definition 2 *For a specification S, the language accepted by S is*

$$\begin{aligned}\mathcal{L}_S \;=\; & \{h \in \mathcal{H} \mid h = \varepsilon \vee \exists\, h_1 \in \mathcal{L}_S, op \in Op, \iota \in In, \omega \in Out \;: \\ & h = h_1\langle\langle \iota, op\rangle, \omega\rangle \wedge (\exists\, s \in \mathit{final}_S(h_1), s' \in \mathit{St} \;:\; (\langle \iota, s\rangle, \langle s', \omega\rangle) \in op^S)\}\end{aligned}$$

It follows from this definition that $\mathcal{L}_S$ is in $\mathcal{L}an_{\mathcal{H}}$.

2.3. Example

Consider the random number generating module defined by $\mathcal{L}_A$ in Appendix A. It contains two operations: *random* generates a random integer value and has no output (indicated by the special value $\perp$), and *val* returns the value generated by the last call to *random* as an output. If no call to *random* has been made, *val* returns 0. For example, the history

$$\langle val, 0\rangle\langle random, \perp\rangle\langle val, \perp 1\rangle\langle random, \perp\rangle\langle val, 1\rangle$$

belongs to the language $\mathcal{L}_A$, whereas the history

$$\langle random, \perp\rangle\langle val, \perp 1\rangle\langle val, 1\rangle$$

does not. Here we have used op as a shorthand for $\langle \perp, op\rangle$, a call to an operation with no input.

An Object-Z specification $SA1$ for $\mathcal{L}_A$ is shown in Figure 1. An Object-Z class is represented as a named box, $SA1$ in this case. A class contains an unnamed state schema, an initialisation schema (*INIT*), and zero or more operations (two in this case). For $SA1$,

the state consists of the integer variable v. The initial value of v is constrained to 0 in the initial state schema. The delta-list $\Delta(v)$ in the schema for *random* indicates that the value of v may change; the predicate *true* indicates that the value v' after the call is not constrained. Finally, the schema for *val* specifies that the output variable *out*! is equal to v. Since *val* does not have a delta-list, the value of v does not change (i.e., $v' = v$).

3. Demonic languages and specifications

The state-abstraction lattice defined in [9] applies to languages that are complete and deterministic. A language $\mathcal{L}$ is complete if for every trace in $Tr(\mathcal{H})$ there is at least one history in $\mathcal{L}$, and it is deterministic if for every trace in $Tr(\mathcal{H})$ there is at most one history in $\mathcal{L}$.

Definition 3 *A language* $\mathcal{L} \in \mathcal{L}an_{\mathcal{H}}$ *is complete if*

$$Tr(\mathcal{L}) \;=\; Tr(\mathcal{H})$$

A language $\mathcal{L}$ *is deterministic if*

$$\forall\, t \in Tr(\mathcal{L}) \;:\; \#\nabla_{\mathcal{L}}(t) \;=\; 1$$

For a language that is both complete and deterministic, there is exactly one history in the language for each trace. Specifications are also assumed to be *state-deterministic*, in that there is a unique final state for each history.

Definition 4 *A specification* S *is state-deterministic if*

$$\forall\, h \in \mathcal{L}_S \;:\; \#final_S(h) \;=\; 1$$

As a special case, note that for a state-deterministic specification S

$$\#final_S(\varepsilon) \;=\; \#\textsc{Init} \;=\; 1$$

The language $\mathcal{L}_A$ discussed in the previous section is complete, but not deterministic. The specification $SA1$ is not state-deterministic. The specification $SB1$, obtained from $SA1$ by changing the specification of *random* to

random

$\Delta(v)$
$v' = v + 1$

so that it increments the value of v each time it is called, is state-deterministic. It defines the language $\mathcal{L}_B$ that is both complete and deterministic.

If instead we change $SA1$ to $SC1$ by adding the operation

two

$v = 2$

that is only enabled when v has the value 2, then this defines a language $\mathcal{L}_C$ that is neither complete, nor deterministic. Note that *two* does not have a delta-list and therefore does not change the value of v; it checks that the value of v before the operation is 2, and if it is not, then the operation is not enabled. This means that $\mathcal{L}_C$ is not complete, because $\langle two, \bot \rangle \notin \mathcal{L}_C$. $\mathcal{L}_C$ is also not deterministic because $\mathcal{L}_A$ is not deterministic and $\mathcal{L}_A \subseteq \mathcal{L}_C$.

In the remainder of this paper, the notion of a demonic language will play a major role. We will demonstrate that this language class provides a natural semantics for data refinements in VDM and Z. Intuitively, a language $\mathcal{L}$ is demonic if for every trace t in $Tr(\mathcal{L})$, every history of $\mathcal{L}$ corresponding to a sub-trace of t must be extendible by calls from t.

Definition 5 *A language $\mathcal{L}$ is demonic if*

$$\forall\ t \in Tr(\mathcal{L}) \setminus \{\varepsilon\}\ :\ \nabla_{\mathcal{L}}(front(t)) = \{\, h \rhd \{1, ..., \#t \perp 1\} \mid h \in \nabla_{\mathcal{L}}(t) \,\}$$

The set of demonic languages will be denoted by

$$\mathcal{L}an^{d}_{\mathcal{H}} = \{\mathcal{L} \subseteq \mathcal{H} \mid \mathcal{L} \neq \varnothing \wedge \mathcal{L} \textit{ is demonic}\}$$

Note that the inclusion

$$\forall\ t \in Tr(\mathcal{L}) \setminus \{\varepsilon\}\ :\ \nabla_{\mathcal{L}}(front(t)) \supseteq \{\, h \rhd \{1, ..., \#t \perp 1\} \mid h \in \nabla_{\mathcal{L}}(t) \,\} \tag{1}$$

is an equivalent way of expressing that a non-empty language $\mathcal{L}$ is prefix-closed.

The languages $\mathcal{L}_A$ and $\mathcal{L}_B$ are both demonic, but the language $\mathcal{L}_C$ is not. This is because $SC1$ includes the operation *two* that is only enabled when the value of the state variable v is 2. For example, for

$$t = \langle random \rangle \langle val \rangle \langle two \rangle$$

we have

$$\nabla_{\mathcal{L}_C}(front(t)) = \{\, \langle random, \bot \rangle \langle val, i \rangle \mid i \in \mathbb{Z} \,\}$$

whereas

$$\{\, h \rhd \{1, 2\} \mid h \in \nabla_{\mathcal{L}_C}(t) \,\} = \{\, \langle random, \bot \rangle \langle val, 2 \rangle \,\}$$

Every deterministic language is demonic. Unfortunately, the set of demonic languages $\mathcal{L}an^{d}_{\mathcal{H}}$ does not behave as nicely as the set of prefix-closed languages $\mathcal{L}an_{\mathcal{H}}$, which forms a complete lattice under the inclusion ordering $\subseteq$ and the usual set operations. In general, demonic languages are not closed under intersection and union. However, if two demonic languages have the same set of traces, then their union is demonic.

Proposition 1 *If for a family of languages $\mathcal{L}_i \in \mathcal{L}an^{d}_{\mathcal{H}}$, $i \in I$ with $Tr(\mathcal{L}_i) = Tr(\mathcal{L}_j)$, $i, j \in I$, then $\cup_{i \in I} \mathcal{L}_i \in \mathcal{L}an^{d}_{\mathcal{H}}$.*

Proof. We take a trace $t \in Tr(\cup_{i\in I}\mathcal{L}_i) \setminus \{\varepsilon\}$. Then, $t \in Tr(\mathcal{L}_i) \setminus \{\varepsilon\}$, for every $i \in I$. The languages $\mathcal{L}_i$, $i \in I$ are demonic and therefore

$$\nabla_{\mathcal{L}_i}(front(t)) = \{\, h \rhd \{1, ..., \#t \perp 1\} \mid h \in \nabla_{\mathcal{L}_i}(t) \,\}$$

for $i \in I$. Hence,

$$\begin{array}{rcl} \nabla_{\cup_{i\in I}\mathcal{L}_i}(front(t)) & = & \cup_{i\in I}\nabla_{\mathcal{L}_i}(front(t)) \\ & = & \cup_{i\in I}\{\, h \rhd \{1, ..., \#t \perp 1\} \mid h \in \nabla_{\mathcal{L}_i}(t) \,\} \\ & = & \{\, h \rhd \{1, ..., \#t \perp 1\} \mid h \in \nabla_{\cup_{i\in I}\mathcal{L}_i}(t) \,\} \end{array}$$

□

There is a notion corresponding to demonic languages for specifications. A specification is demonic if any two states that can be reached after a certain number of calls can be extended by the same set of calls.

Definition 6 *A specification S is demonic if*

$$\forall\, t \in Tr(\mathcal{L}_S) \setminus \{\varepsilon\} \;:\; ptraces_S(front(t)) \subseteq pres_S(last(t))$$

Observe that every specification S that is *total* in the following sense,

$$\forall\, op \in Op, \iota \in In \;:\; pres_S(\langle \iota, op\rangle) \neq \varnothing \;\Rightarrow\; pres_S(\langle \iota, op\rangle) = St$$

is demonic. Every specification S that is not total has a natural total and hence demonic extension obtained by adding a new state *abort* and a new output symbol ab. We then extend every operation $op \in Op$ in the following way: if $pres_S(\langle \iota, op\rangle) \neq \varnothing$, we add a new transition $(\langle \iota, s\rangle, \langle abort, ab\rangle)$ to the interpretation op^S for every pair $\langle \iota, s\rangle$ such that $s \notin pres_S(\langle \iota, op\rangle)$. This includes transitions of the form $(\langle \iota, abort\rangle, \langle abort, ab\rangle)$ for every operation. Similarly, every prefix-closed language has a natural demonic extension.

Proposition 2 *Every demonic specification S defines a demonic language $\mathcal{L}_S$.*

Proof. Any specification S defines a prefix-closed language $\mathcal{L}_S$ and therefore we have inclusion (1). To prove the inclusion in the other direction, we assume a demonic specification S. Let $t \in Tr(\mathcal{L}_S) \setminus \{\varepsilon\}$ and $h \in \nabla_{\mathcal{L}_S}(front(t))$. Because S is demonic we have $ptraces_S(front(t)) \subseteq pres_S(last(t))$ and so $finals_S(h) \subseteq pres_S(last(t))$. Hence we may extend h by the call $last(t)$, and there exists an output $\omega \in Out$ such that $h\langle last(t), \omega\rangle \in \nabla_{\mathcal{L}_S}(t)$, which is what we need to prove the inclusion in the other direction. □

The converse is not true in general: there are non-demonic specifications that specify demonic languages. For example, the specification $SD1$ obtained from $SC1$ by removing *val* is not demonic, because even though $\langle random\rangle\langle two\rangle \in Tr(\mathcal{L}_D)$, we have $ptraces_{SD1}(\langle random\rangle) = \mathbb{Z}$ and $pres_{SD1}(\langle two\rangle) = \{2\}$. However, the language $\mathcal{L}_D$ specified by $SD1$ is clearly demonic, because there are no operations with any output. Nevertheless, we show in Proposition 11 that for every demonic language $\mathcal{L}$ there exists a demonic specification S such that $\mathcal{L} = \mathcal{L}_S$ (in fact, there are many such specifications).

We sometimes use an additional condition, requiring that each history $h \in \mathcal{L}_S$ corresponds to exactly one equivalence class of internal states. In this case, it is legitimate to think of "exactly one" state.

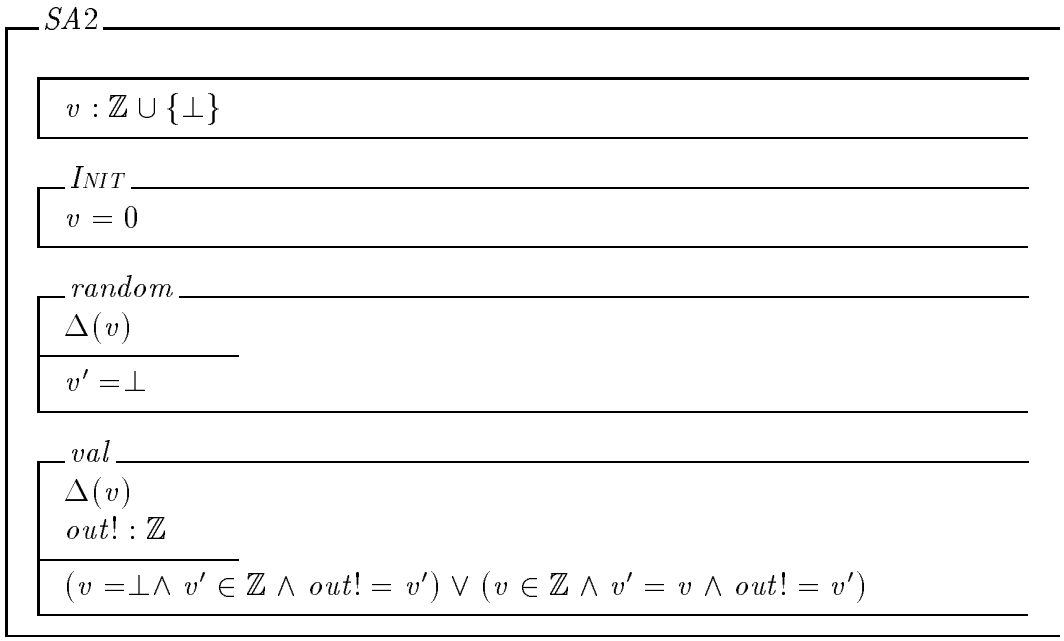


Figure 2. Object-Z specification $SA2$ for $\mathcal{L}_A$

Definition 7 *A specification S is semi-deterministic if*

$$\forall\, op \in Op,\, \iota \in In,\, \omega \in Out,\, s \in St,\, h \in \mathcal{L}_S\ :$$
$$(s \in final_S(h) \wedge h\langle\langle \iota, op\rangle, \omega\rangle \in \mathcal{L}_S) \Rightarrow (\exists\, s' \in St\ :\ (\langle \iota, s\rangle, \langle s', \omega\rangle) \in op^S)$$

In other words, if $h, hh' \in \mathcal{L}_S$ and σ is a state sequence that belongs to history h, then there exists a sequence of states σ' such that $\sigma\sigma'$ belongs to history hh'. So although a history might end up in different states, they must be indistinguishable with respect to future behaviour.

The specification $SA1$ for language $\mathcal{L}_A$ is not semi-deterministic. For example, for $h = \langle random, \bot\rangle$ we have $3 \in final_{SA1}(h)$ and $h\langle val, 5\rangle \in \mathcal{L}_A$, but there exists no state s' such that $(\langle \bot, 3\rangle, \langle s', 5\rangle) \in val^{SA1}$. However, the specification $SA2$, shown in Figure 2, is state-deterministic and hence semi-deterministic. In this specification, we have added $\bot$ as a special value for the state variable v to indicate that *random* has been called without being followed by a call to *val*. Thus, we delay the choice of the random value until a first call to *val* is made after a call to *random*. The disjunction in *val* deals with the two cases where *val* has not been called since the last call to *random* (first disjunct), and where *val* has been called and the value of v should remain the same (second disjunct).

Note that the notion of a semi-deterministic specification is more general than that of a state-deterministic specification. Consider the specification $SE1$ shown in Figure 3. It is similar to $SB1$, but it contains one additional state variable *stuck* and one additional operation *zero*. Initially, the value of v is 0 and *stuck* is *false*. As long as *stuck* is *false*,

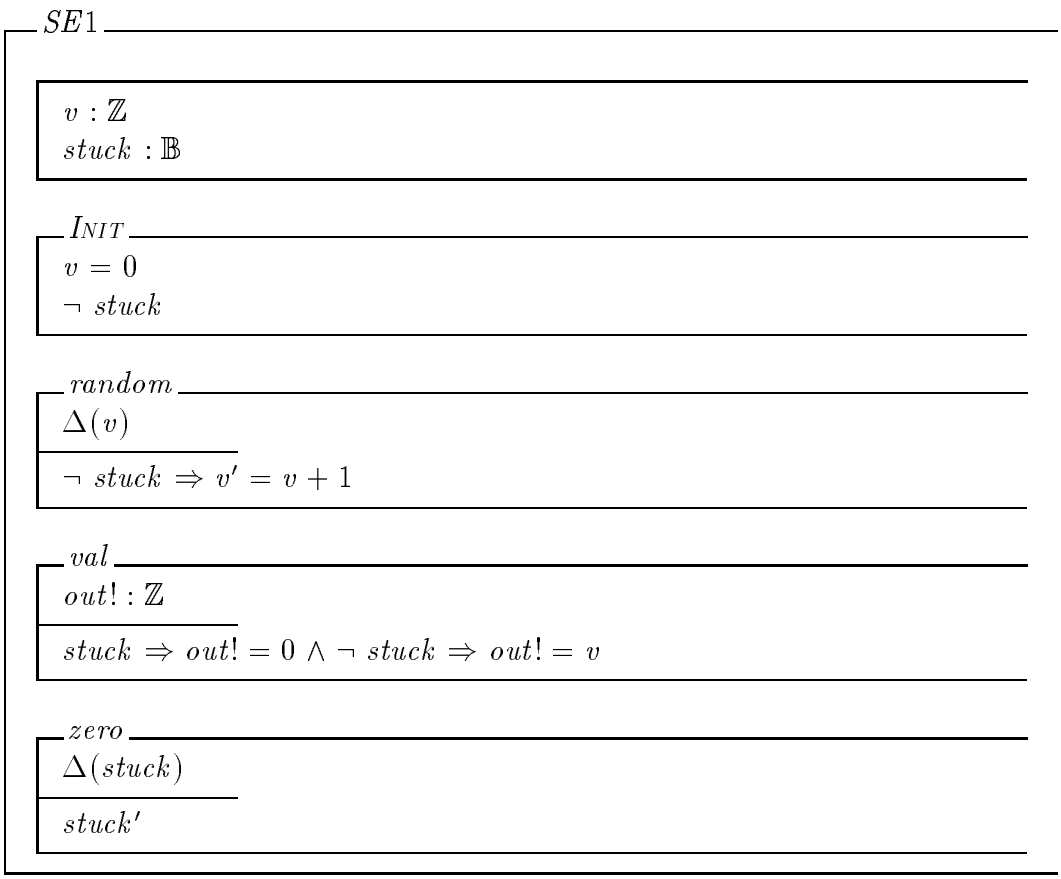


Figure 3. Object-Z specification $SE1$ for $\mathcal{L}_E$

the value of v is incremented each time *random* is called. Note that no value for v' is specified in *random* when *stuck* is *true*, which means that $SE1$ is not state-deterministic. The operation *val* returns 0 if *stuck* is *true*, and the value of v otherwise. Finally, the operation *zero* sets the value of *stuck* to *true*, thereby ensuring that *val* will always return 0 after that.

As explained above, $SE1$ is not state-deterministic. It is semi-deterministic, because no matter what the value of v' is after a call to *random* when *stuck* is *true*, the future behaviour of $SE1$ does not depend in any way on this value of v'. Clearly it is easy to change $SE1$ so that it is state-deterministic and still specifies the same behaviour, by specifying a specific value for v' in *random* when *stuck* is *true*. However, such a specification would unnecessarily restrict the value of v'. Although this is a contrived example, it shows a class of specifications that are semi-deterministic, but not state deterministic: whenever the future behaviour of the specification depends on only part of the state of the specification (for example, in $SE1$, the future behaviour does not depend on the value of v if *stuck* is *true*).

Proposition 3 *Let S be a semi-deterministic specification. S is demonic iff $\mathcal{L}_S$ is a demonic language.*

Proof. One direction of the implication follows from Proposition 1. For the other direction, assume that $\mathcal{L}_S$ is demonic and a trace $t \in Tr(\mathcal{L}_S) \setminus \{\varepsilon\}$ with $last(t) = \langle \iota, op \rangle$. We have to prove

$$ptrace_S(front(t)) \subseteq pre_S(\langle \iota, op \rangle)$$

So let us assume a state $s \in ptrace_S(front(t))$ and a history $h \in \mathcal{L}_S$ with $\mathcal{I}(h) = front(t)$ and $s \in final_S(h)$. Then it is sufficient to prove $s \in pre_S(\langle \iota, op \rangle)$ $\mathcal{L}_S$ is demonic and so we know that there exists an output $\omega \in Out$ such that $h\langle \langle \iota, op \rangle, \omega \rangle \in \mathcal{L}_S$. Hence, by the definition of a semi-deterministic specification, $s \in pre_S(\langle \iota, op \rangle)$.

□

4. Refinement

Data-refinement proofs [6,2,12,17] are used to verify that a specification (or implementation) S^C with a concrete state representation is correct with respect to a specification S^A with an abstract state representation.

There are various well-explored refinement techniques. The refinement of specifications or state machines is often defined as subset relation on observable behaviours [8,1,13,15]. In other words, refinement means that the observable behaviour of S^C must be a subset of the observable behaviour of S^A.

In the following we are going to define an ordering relation on the languages that are generated by specifications and we will use this ordering as the semantics for refinement proofs. Thus we are gaining a refinement semantics that is different to the refinement notions cited above. Briefly, a specification S^C refines a specification S^A if every input that was possible for S^A is valid for S^C and if the corresponding outputs are in a subset relation. With this semantics the notion of forward refinement of VDM and Z [12,18,17] will prove to be a sound refinement method.

Definition 8 *Given two specifications* $S^A = (Op, St^A, In, Out, \textsc{Init}^A, \_^{S^A})$ *and* $S^C = (Op, St^C, In, Out, \textsc{Init}^C, \_^{S^C})$*, a relation*

$$abs : St^C \leftrightarrow St^A$$

and operation $op \in Op$*, we say that* op^{S^A} *forward data-refines to* op^{S^C} ($op^{S^A} \sqsubseteq_{abs} op^{S^C}$) *iff the following obligations are fulfilled [12,17].*

$(DR1)$ $\forall \iota \in In\,,\ s \in St^A\,,\ t \in St^C$:
$((t,s) \in abs \wedge s \in pre_{S^A}(\langle \iota, op \rangle)) \Rightarrow\ t \in pre_{S^C}(\langle \iota, op \rangle)$

$(DR2)$ $\forall \iota \in In\,,\ \omega \in Out\,,\ s \in St^A\,,\ t, t' \in St^C$:
$(s \in pre_{S^A}(\langle \iota, op \rangle) \wedge (t,s) \in abs \wedge (\langle \iota, t \rangle, \langle t', \omega \rangle) \in op^{S^C}) \Rightarrow$
$(\exists\, s' \in St^A\ : (\langle \iota, s \rangle, \langle s', \omega \rangle) \in op^{S^A} \wedge (t', s') \in abs)$

$(DR1)$ asserts that all possible inputs for op^{S^A} are also possible inputs for op^{S^C}. In $(DR2)$ we do not claim that all transitions of op^{S^A} can be simulated. Instead, we require that every possible input of op^{S^A} must be accepted by op^{S^C} with outputs that were possible for op^{S^A}. The relation $\sqsubseteq$ defines a preorder on operations in the above context, i.e., on operations with input in In and output in Out.

With this notion of operation refinement we can state the corresponding technique of specification refinement [17].

Definition 9 *We say that a specification* $S^A = (Op, St^A, In, Out, \textsc{Init}^A, \_^{S^A})$ *can be forward refined to specification* $S^C = (Op, St^C, In, Out, \textsc{Init}^C, \_^{S^C})$ *and write* $S^A \sqsubseteq S^C$ *if there exists an abstraction relation abs as above such that*

(SR1) $\forall\, op \in Op\ :\ op^{S^A} \sqsubseteq_{abs} op^{S^C}$

(SR2) $\forall\, t \in \textsc{Init}^C\ \exists\, s \in \textsc{Init}^A\ :\ (t,s) \in abs$

We write $S^A \sqsubseteq_{abs} S^C$ *if we want to explicitly indicate the abstraction relation abs.*

Obligation $(SR1)$ requires that every abstract operation can be refined to a concrete one and obligation $(SR2)$ states that, via abs, every concrete initial state corresponds to at least one abstract state. Note that we overload the semantics of the symbol $\sqsubseteq$. It will be obvious from the context whether we mean operation or specification refinement.

For the example specifications in Appendices A and B, note that $SA1 \sqsubseteq SB1$ with the abstraction relation

$$abs = \{(i,i)\ :\ i \in \mathbb{Z}\}$$

However, $SB1$ does not refine to $SA1$ using the same abstraction relation, because there are many after states for *random* in $SA1$ and there is only a single one in $SB1$. In fact, as we will see below, there is no abstraction relation so that $SB1$ forward refines to $SA1$.

Proposition 4 *Forward refinement* $\sqsubseteq$ *defines a preorder on specifications* S *as defined above.*

Proof. This follows directly from the fact that forward refinement for operations is a preorder. Reflexivity follows from using the identity as the abstraction relation. For specifications S^A, S^B and S^C with

$$S^A \sqsubseteq_{abs_1} S^B \sqsubseteq_{abs_2} S^C$$

we have $S^A \sqsubseteq_{abs} S^C$ with the relation-composition $abs = abs_2 \circ abs_1$.

□

To provide a semantics for refinement proofs on specifications and to ultimately generalise the results from [9], we define a partial ordering on languages.

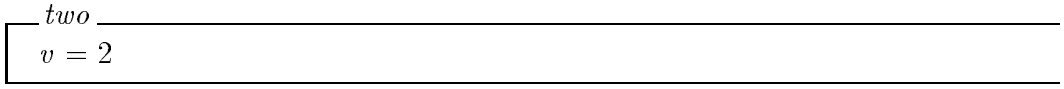
Definition 10 *Let $\mathcal{L}$ and $\mathcal{L}'$ be languages in $\mathcal{H}$,*

$$\mathcal{L}' \Subset \mathcal{L} \quad \textit{iff} \quad Tr(\mathcal{L}) \subseteq Tr(\mathcal{L}') \wedge \forall\, t \in Tr(\mathcal{L}) : \nabla_{\mathcal{L}'}(t) \subseteq \nabla_{\mathcal{L}}(t)$$

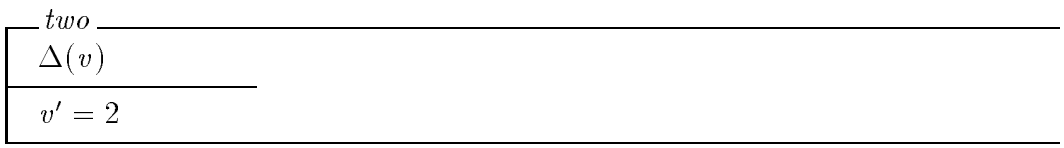
The above ordering on languages corresponds to the intuition behind obligations $(DR1)$ and $(DR2)$. For languages $\mathcal{L}$ and $\mathcal{L}'$, $\mathcal{L}' \Subset \mathcal{L}$ if all traces of $Tr(\mathcal{L})$ occur in $Tr(\mathcal{L}')$ and if every history in $\mathcal{L}'$ corresponding to a trace in $\mathcal{L}$ is also a history in $\mathcal{L}$.

For $\mathcal{L}_A$ and $\mathcal{L}_B$ from the appendices, we have $\mathcal{L}_B \Subset \mathcal{L}_A$ because the set of histories for $\mathcal{L}_B$ is a subset of the histories for $\mathcal{L}_A$ and all traces that occur in $\mathcal{L}_A$ also occur in $\mathcal{L}_B$. However, we do not have $\mathcal{L}_A \Subset \mathcal{L}_B$ because $\langle random\rangle\langle val\rangle \in Tr(\mathcal{L}_B)$, but $\langle random, \bot\rangle\langle val, 2\rangle$ is in $\nabla_{\mathcal{L}_A}(\langle random\rangle\langle val\rangle)$ and not in $\nabla_{\mathcal{L}_B}(\langle random\rangle\langle val\rangle)$.

Note that the ordering $\Subset$ is different from the subset ordering on languages. For example, to find two languages that are ordered by $\Subset$, but that are not in a subset relation, we define the language $\mathcal{L}_F$ obtained by changing the specification of *two* in $SC1$ from



to



in $SF1$. Note that *two* in $SF1$ does not have a precondition (i.e., the operation can always be applied) and always changes the value of v to 2; as a result, $\mathcal{L}_F$ is demonic. Now $\mathcal{L}_F \Subset \mathcal{L}_C$, even though $\mathcal{L}_F$ contains more histories than $\mathcal{L}_C$.

In the subsequent discussion we are going to identify the poset (partially ordered set)

$$(\mathcal{L}an^d_{\mathcal{H}}, \Subset)$$

as a domain for the characterisation of refinement proofs with forward refinement and a notion of backward refinement. We will also see that the partial ordering $\Subset$ on demonic languages characterises forward refinement proofs on demonic, semi-deterministic specifications. Note that $(\mathcal{L}an_{\mathcal{H}}, \Subset)$ and $(\mathcal{L}an^d_{\mathcal{H}}, \Subset)$ when extended with a bottom element are complete lattices similar to the complete lattice $(\mathcal{L}an_{\mathcal{H}}, \subseteq)$.

We pointed out that the intersection of demonic languages is not necessarily demonic. However, for demonic languages $\mathcal{L}' \Subset \mathcal{L}$, the intersection $\mathcal{L} \cap \mathcal{L}'$ is demonic.

Proposition 5 *For languages* $\mathcal{L}, \mathcal{L}', \mathcal{L}'' \subseteq \mathcal{H}$ *we have:*

i) $\mathcal{L}' \Subset \mathcal{L} \Rightarrow Tr(\mathcal{L} \cap \mathcal{L}') = Tr(\mathcal{L}) \cap Tr(\mathcal{L}') = Tr(\mathcal{L})$, $\mathcal{L}' \Subset \mathcal{L}' \cap \mathcal{L}$

ii) $\mathcal{L}' \Subset \mathcal{L} \Leftrightarrow (\, \mathcal{L}' \cap \mathcal{L} \Subset \mathcal{L} \wedge \mathcal{L}' \cup \mathcal{L} \Subset \mathcal{L} \,)$

iii) $\mathcal{L}'' \Subset \mathcal{L}' \Subset \mathcal{L} \Rightarrow \mathcal{L}'' \cap \mathcal{L} \subseteq \mathcal{L}'$

iv) ($\mathcal{L}'$ *demonic* $\wedge\, \mathcal{L}' \Subset \mathcal{L}$) $\Rightarrow$ $\mathcal{L}' \cap \mathcal{L}$ *demonic*

Proof. For *i*): For any two languages $\mathcal{L}$ and $\mathcal{L}'$

$$Tr(\mathcal{L} \cap \mathcal{L}') \subseteq Tr(\mathcal{L}) \cap Tr(\mathcal{L}') \subseteq Tr(\mathcal{L})$$

Assume $\mathcal{L}' \Subset \mathcal{L}$. Then, $Tr(\mathcal{L}) \subseteq Tr(\mathcal{L}) \cap Tr(\mathcal{L}')$. For $t \in Tr(\mathcal{L}) \cap Tr(\mathcal{L}')$ we have $\nabla_{\mathcal{L}'}(t) \subseteq \nabla_{\mathcal{L}}(t)$, hence $t \in Tr(\mathcal{L} \cap \mathcal{L}')$ and furthermore

$$\nabla_{\mathcal{L}'}(t) \subseteq \nabla_{\mathcal{L}'}(t) \cap \nabla_{\mathcal{L}}(t) = \nabla_{\mathcal{L}' \cap \mathcal{L}}(t)$$

For *ii*): Assume $\mathcal{L}' \Subset \mathcal{L}$. Then, $Tr(\mathcal{L}) \subseteq Tr(\mathcal{L} \cup \mathcal{L}')$ and for $t \in Tr(\mathcal{L})$,

$$\nabla_{\mathcal{L} \cup \mathcal{L}'}(t) \;=\; \nabla_{\mathcal{L}}(t) \cup \nabla_{\mathcal{L}'}(t) \;=\; \nabla_{\mathcal{L}}(t)$$

Therefore, $\mathcal{L}' \cup \mathcal{L} \Subset \mathcal{L}$. Part *i*) and $Tr(\mathcal{L}) \subseteq Tr(\mathcal{L}')$ imply

$$Tr(\mathcal{L} \cap \mathcal{L}') = Tr(\mathcal{L}) \cap Tr(\mathcal{L}') = Tr(\mathcal{L})$$

For $t \in Tr(\mathcal{L})$,

$$\nabla_{\mathcal{L} \cap \mathcal{L}'}(t) \;=\; \nabla_{\mathcal{L}}(t) \cap \nabla_{\mathcal{L}'}(t) \;=\; \nabla_{\mathcal{L}'}(t)$$

Hence, $\mathcal{L}' \cap \mathcal{L} \Subset \mathcal{L}$.

Now assume $\mathcal{L}' \cap \mathcal{L} \Subset \mathcal{L}$ and $\mathcal{L}' \cup \mathcal{L} \Subset \mathcal{L}$. It follows

$$Tr(\mathcal{L}) \subseteq Tr(\mathcal{L} \cap \mathcal{L}') \subseteq Tr(\mathcal{L}')$$

and for $t \in Tr(\mathcal{L})$, $\nabla_{\mathcal{L}'}(t) \subseteq \nabla_{\mathcal{L}' \cup \mathcal{L}}(t) \subseteq \nabla_{\mathcal{L}}(t)$. Hence, $\mathcal{L}' \Subset \mathcal{L}$.

For *iii*): Let $t \in Tr(\mathcal{L}'' \cap \mathcal{L})$. Then, $t \in Tr(\mathcal{L})$ and because of $\mathcal{L}' \Subset \mathcal{L}$ we obtain $t \in Tr(\mathcal{L}')$. Hence,

$$\begin{array}{rcl} \nabla_{\mathcal{L}'' \cap \mathcal{L}}(t) & = & \nabla_{\mathcal{L}''}(t) \cap \nabla_{\mathcal{L}}(t) \\ & \subseteq & \nabla_{\mathcal{L}'}(t) \cap \nabla_{\mathcal{L}}(t) \\ & \subseteq & \nabla_{\mathcal{L}'}(t) \end{array}$$

For *iv*): For $t \in Tr(\mathcal{L}' \cap \mathcal{L}) \setminus \{\varepsilon\}$ we conclude

$$\begin{array}{rcl} \nabla_{\mathcal{L}' \cap \mathcal{L}}(front(t)) & = & \nabla_{\mathcal{L}'}(front(t)) \cap \nabla_{\mathcal{L}}(front(t)) \\ & = & \nabla_{\mathcal{L}'}(front(t)) \\ & = & \{h \rhd \{1, ..., \#t \perp 1\} \mid h \in \nabla_{\mathcal{L}'}(t)\} \\ & = & \{h \rhd \{1, ..., \#t \perp 1\} \mid h \in \nabla_{\mathcal{L}' \cap \mathcal{L}}(t)\} \end{array}$$

□

Definition 11 *Given two specifications* $S^A = (Op, St^A, In, Out, \mathit{INIT}^A, \_^{S^A})$ *and* $S^C = (Op, St^C, In, Out, \mathit{INIT}^C, \_^{S^C})$*, we define the restricted-use specification*

$$S^C[S^A] = (Op, St^C, In, Out, \mathit{INIT}^C, \_^{S^C[S^A]})$$

of S^C *under* S^A *as follows. For every operation* $op \in Op$*,*

$$op^{S^C[S^A]} = \{(\langle \iota, t\rangle, \langle t', \omega\rangle) \in op^{S^C} \mid \mathit{pre}_{S^A}(\langle \iota, op\rangle) \neq \varnothing\}$$

In the case of a forward refinement, $S^C[S^A]$ specifies the behaviour of S^C for traces accepted by S^A. We can think of this as projecting S^A on S^C and then using S^C as we would have S^A.

Proposition 6 *For specifications* S^A *and* S^C*:*

i) $S^C[S^A] \sqsubseteq_{id} S^C$ *with the identity id on* St^C

ii) $S^A \sqsubseteq_{abs} S^C \Leftrightarrow S^A \sqsubseteq_{abs} S^C[S^A]$

iii) S^C *demonic* $\Rightarrow S^C[S^A]$ *demonic*

Proof. Part *i*) follows from the definition of refinement. From *i*) and the transitivity of refinement, it follows that $S^A \sqsubseteq_{abs} S^C[S^A]$ implies $S^A \sqsubseteq_{abs} S^C$. For the converse, assume $S^A \sqsubseteq_{abs} S^C$. We prove $S^A \sqsubseteq_{abs} S^C[S^A]$: let $op \in Op$.

$(DR1)$: Let $s \in \mathit{pre}_{S^A}(\langle \iota, op\rangle)$ and $r \in St^C$ with $(r, s) \in abs$. With $(DR1)$ for the refinement $op^{S^A} \sqsubseteq_{abs} op^{S^C}$ we find $r \in \mathit{pre}_{S^C}(\langle \iota, op\rangle)$. Hence, $r \in \mathit{pre}_{S^C[S^A]}(\langle \iota, op\rangle)$.

$(DR2)$: Let $(\langle \iota, r\rangle\langle r', \omega\rangle) \in op^{S^C[S^A]}$ with $(r, s) \in abs$ and $s \in \mathit{pre}_{S^A}(\langle \iota, op\rangle)$. Hence, $(\langle \iota, r\rangle\langle r', \omega\rangle) \in op^{S^C}$ and from $(DR2)$ for the refinement $op^{S^A} \sqsubseteq_{abs} op^{S^C}$ we find $s' \in St^A$ such that $(\langle \iota, s\rangle\langle s', \omega\rangle) \in op^{S^A}$ and $(r', s') \in abs$.

The remaining obligation $(SR2)$ is satisfied because $\mathit{INIT}^{S^C[S^A]} = \mathit{INIT}^{S^C}$.

For *iii*): Note that if $\mathit{pre}_{S^A}(\langle \iota, op\rangle) \neq \varnothing$, then $\mathit{pre}_{S^C}(\langle \iota, op\rangle) = \mathit{pre}_{S^C[S^A]}(\langle \iota, op\rangle)$. Therefore, for $t \in Tr(S^C[S^A]) \setminus \{\varepsilon\}$, we have $\mathit{ptrace}_{S^C}(t) = \mathit{ptrace}_{S^C[S^A]}(t)$.

□

If we use the ordering $\Subset$ as the underlying semantics of specification refinement and forward refinement with obligations $(SR1)$ and $(SR2)$ as the refinement technique, then Theorem 1 below proves the soundness of forward refinement for demonic specifications. We first prove two lemmas.

Lemma 1 *Assume specifications* S^A *and* S^C*. If* S^A *is demonic and there exists an abstraction relation* $abs : St^C \leftrightarrow St^A$ *such that* $S^A \sqsubseteq_{abs} S^C$*, then for every* $h \in \mathcal{L}_{S^A} \cap \mathcal{L}_{S^C}$*:*

$$\forall\, r \in \mathit{final}_{S^C}(h)\ \exists\, s \in \mathit{final}_{S^A}(h) : (r, s) \in abs$$

Proof. We prove this by induction on the length of the histories $h \in \mathcal{L}_{S^A} \cap \mathcal{L}_{S^C}$.

Base case $(h = \varepsilon)$: We have $\mathit{final}_{S^A}(\varepsilon) = \mathit{INIT}^A$ and $\mathit{final}_{S^C}(\varepsilon) = \mathit{INIT}^C$. In this case, the assertion is exactly $(SR2)$.

Induction step: Assume $h\langle\langle \iota, op\rangle, \omega\rangle \in \mathcal{L}_{S^A} \cap \mathcal{L}_{S^C}$ and $r' \in \mathit{final}_{S^C}(h\langle\langle \iota, op\rangle, \omega\rangle)$. We find $r \in \mathit{final}_{S^C}(h)$ such that $(\langle \iota, r\rangle, \langle r', \omega\rangle) \in op^{S^C}$. The induction hypothesis gives us

$s \in final_{S^A}(h)$ with $(r,s) \in abs$. Since S^A is demonic, $s \in pre_{S^A}(\langle \iota, op \rangle)$. Applying $(DR2)$ we find $s' \in St^A$ with $(r',s') \in abs$ and $(\langle \iota, s \rangle, \langle s', \omega \rangle) \in op^{S^A}$. Therefore, $s' \in final_{S^A}(h\langle \langle \iota, op \rangle, \omega \rangle)$.

□

Lemma 2 *Assume specifications S^A and S^C. If S^A is demonic and there exists an abstraction relation $abs : St^C \leftrightarrow St^A$ such that $S^A \sqsubseteq_{abs} S^C$, then for every $t \in Tr(\mathcal{L}_{S^A}) \setminus \{\varepsilon\}$:*

a) $\varnothing \neq ptrace_{S^C}(front(t)) \subseteq pre_{S^C}(last(t))$

b) $\forall\, r \in ptrace_{S^C}(t)\ \exists\, s \in ptrace_{S^A}(t) : (r,s) \in abs$

Proof. We prove the assertion by induction on the length of traces $t \in Tr(\mathcal{L}_{S^A}) \setminus \{\varepsilon\}$:

Base case ($t = \langle \iota, op \rangle$): Since we assume there exists at least one initial state for each specification, $\textsc{Init}^C$ is non-empty and so $\varnothing \neq \textsc{Init}^C = ptrace_{S^C}(\varepsilon) = ptrace_{S^C}(front(t))$. Let $r \in ptrace_{S^C}(front(t))$. Because of $(SR2)$, we find $s \in \textsc{Init}^A$ with $(r,s) \in abs$. S^A is demonic and so $\textsc{Init}^A \subseteq pre_{S^A}(\langle \iota, op \rangle)$. Hence, $s \in pre_{S^A}(\langle \iota, op \rangle)$ and condition $(DR1)$ implies $r \in pre_{S^C}(\langle \iota, op \rangle)$. This proves a). To prove b), assume $r' \in ptrace_{S^C}(t)$. Then there exists $r \in \textsc{Init}^C$ and $\omega \in Out$ such that $(\langle \iota, r \rangle \langle r', \omega \rangle) \in op^{S^C}$. Then $(SR2)$ implies that there exists $s \in \textsc{Init}^A$ with $(r,s) \in abs$. Since S^A is demonic, we have $s \in pre_{S^A}(t)$ and $(DR2)$ ensures the existence of $s' \in ptrace_{S^A}(t)$ with $(\langle \iota, s \rangle \langle s', \omega \rangle) \in op^{S^A}$ and $(r',s') \in abs$.

Induction step: Let $t\langle \iota, op \rangle \in Tr(\mathcal{L}_{S^A})$ with $t \neq \varepsilon$. By the induction hypothesis $a)$, we find $\varnothing \neq ptrace_{S^C}(front(t)) \subseteq pre_{S^C}(last(t))$. Hence, $\varnothing \neq ptrace_{S^C}(t)$. Let $r \in ptrace_{S^C}(t)$. Because of induction hypothesis $b)$ we find an element $s \in ptrace_{S^A}(t)$ with $(r,s) \in abs$. S^A is demonic and so, $s \in pre_{S^A}(\langle \iota, op \rangle)$. The condition $(DR1)$ implies $r \in pre_{S^C}(\langle \iota, op \rangle)$ which concludes the proof of $a)$. For $b)$, let $r' \in ptrace_{S^C}(t\langle \iota, op \rangle)$. Then, there exists $r \in ptrace_{S^C}(t)$ and $\omega \in Out$ such that $(\langle \iota, r \rangle, \langle r', \omega \rangle) \in op^{S^C}$. The induction hypothesis $b)$ ensures the existence of $s \in ptrace_{S^A}(t)$ with $(r,s) \in abs$. S^A is demonic, hence $s \in pre_{S^A}(\langle \iota, op \rangle)$. With $(DR2)$ we find $s' \in St^A$ such that $(r',s') \in abs$ and $(\langle \iota, s \rangle, \langle s', \omega \rangle) \in op^{S^A}$. Hence, $s' \in ptrace_{S^A}(t\langle \iota, op \rangle)$.

□

We can now prove the soundness of forward refinement for demonic specifications.

Theorem 1 *Assume specifications S^A and S^C. If S^A is demonic and there exists an abstraction relation $abs : St^C \leftrightarrow St^A$ such that $S^A \sqsubseteq_{abs} S^C$, then*

i) $S^C[S^A]$ is demonic and $\mathcal{L}_{S^C[S^A]} = \mathcal{L}_{S^A} \cap \mathcal{L}_{S^C}$.

ii) $\mathcal{L}_{S^C} \Subset \mathcal{L}_{S^A}$.

Proof. To prove $\mathcal{L}_{S^C} \Subset \mathcal{L}_{S^A}$, we show the following properties:

1) $Tr(\mathcal{L}_{S^A}) \subseteq Tr(\mathcal{L}_{S^C})$

2) $\forall\, h \in \mathcal{L}_{S^A}, h' \in \mathcal{L}_{S^C} : \mathcal{I}(h) = \mathcal{I}(h') \Rightarrow h' \in \mathcal{L}_{S^A}$

Observe that part *a*) of Lemma 2 implies 1). We prove assertion 2) by induction on the length of the histories $h \in \mathcal{L}_{S^A}$ and $h' \in \mathcal{L}_{S^C}$.

Base case ($h = h' = \varepsilon$): This follows from the fact that ε belongs to every language.

Induction step: Assume $h\langle\langle \iota, op\rangle, \omega\rangle \in \mathcal{L}_{S^A}$ and $h'\langle\langle \iota, op\rangle, \omega'\rangle \in \mathcal{L}_{S^C}$ with $\mathcal{I}(h) = \mathcal{I}(h')$. Because of the induction hypothesis, $h' \in \mathcal{L}_{S^A} \cap \mathcal{L}_{S^C}$. Let $r \in \mathit{final}_{S^C}(h')$ and $r' \in St^C$ with $(\langle \iota, r\rangle, \langle r', \omega'\rangle) \in op^{S^C}$. It follows from Lemma 1 that there exists $s \in \mathit{final}_{S^A}(h')$ such that $(r, s) \in \mathit{abs}$. S^A is demonic, hence $s \in \mathit{pre}_{S^A}(\langle \iota, op\rangle)$. $(DR2)$ gives us $s' \in St^A$ with $(\langle \iota, s\rangle, \langle s', \omega'\rangle) \in op^{S^A}$. Hence, $h'\langle\langle \iota, op\rangle, \omega'\rangle \in \mathcal{L}_{S^A}$.

To prove *i*), first we show $\mathcal{L}_{S^C[S^A]} = \mathcal{L}_{S^A} \cap \mathcal{L}_{S^C}$. The inclusions $\mathcal{L}_{S^A} \cap \mathcal{L}_{S^C} \subseteq \mathcal{L}_{S^C[S^A]}$ and $\mathcal{L}_{S^C[S^A]} \subseteq \mathcal{L}_{S^C}$ are obvious. We prove $\mathcal{L}_{S^C[S^A]} \subseteq \mathcal{L}_{S^A}$ by induction on the length of $h \in \mathcal{L}_{S^C[S^A]}$.

Base case ($h = \varepsilon$): Again, this follows from the fact that ε belongs to every language.

Induction step: Let $h\langle\langle \iota, op\rangle, \omega\rangle \in \mathcal{L}_{S^C[S^A]}$. Then, $h \in \mathcal{L}_{S^A} \cap \mathcal{L}_{S^C}$ according to our induction hypothesis. We find $r \in \mathit{final}_{S^C[S^A]}(h)$ with $r \in \mathit{pre}_{S^C[S^A]}(\langle \iota, op\rangle)$. Then, $\mathit{pre}_{S^A}(\langle \iota, op\rangle) \neq \varnothing$ and $r \in \mathit{final}_{S^C}(h)$. Because of Lemma 1 we find $s \in \mathit{final}_{S^A}(h)$ with $(r, s) \in \mathit{abs}$. S^A is demonic, hence $s \in \mathit{pre}_{S^A}(\langle \iota, op\rangle)$. Applying $(DR2)$ we find $s' \in St^A$ such that $(\langle \iota, s\rangle, \langle s', \omega\rangle) \in op^{S^A}$. Hence, $h\langle\langle \iota, op\rangle, \omega\rangle \in \mathcal{L}_{S^A}$.

To prove that $S^C[S^A]$ is demonic we take a trace $t \in \mathit{Tr}(\mathcal{L}_{S^C[S^A]}) \setminus \{\varepsilon\}$. From what we proved before it follows $t \in \mathit{Tr}(\mathcal{L}_{S^A} \cap \mathcal{L}_{S^C}) \setminus \{\varepsilon\}$. Because of $\mathit{pre}_{S^A}(\mathit{last}(t)) \neq \varnothing$ we have $\mathit{pre}_{S^C}(\mathit{last}(t)) = \mathit{pre}_{S^C[S^A]}(\mathit{last}(t))$. Hence, by Lemma 2 a),

$$\begin{array}{lcl} \mathit{ptrace}_{S^C[S^A]}(\mathit{front}(t)) & \subseteq & \mathit{ptrace}_{S^C}(\mathit{front}(t)) \\ & \subseteq & \mathit{pre}_{S^C}(\mathit{last}(t)) \\ & = & \mathit{pre}_{S^C[S^A]}(\mathit{last}(t)) \end{array}$$

□

We have seen that $\mathcal{L}_{S^C} \Subset \mathcal{L}_{S^A}$ is a necessary condition for forward refinement as defined above. It is not sufficient for forward refinement of nondeterministic specifications in general. With respect to our semantics we have a sound, but not a complete refinement technique. Nevertheless, forward refinement is a complete method if we restrict ourselves to semi-deterministic specifications.

Theorem 2 *Assume a demonic, semi-deterministic specification S^A and a demonic specification S^C. Then the following two conditions are equivalent.*

i) $\mathcal{L}_{S^C} \Subset \mathcal{L}_{S^A}$

ii) $S^A \sqsubseteq S^C$

Proof. *ii)* $\Rightarrow$ *i)* is a consequence of Theorem 1 *ii)*.

i) $\Rightarrow$ *ii)*: For demonic specifications S^A and S^C we define a relation $\mathit{abs} : St^C \leftrightarrow St^A$ by

$$(r, s) \in \mathit{abs} \;\text{ iff }\; \exists\, h \in \mathcal{L}_{S^C} \cap \mathcal{L}_{S^A} \;:\; r \in \mathit{final}_{S^C}(h) \wedge s \in \mathit{final}_{S^A}(h)$$

We are going to prove the refinement relation $S^A \sqsubseteq_{\mathit{abs}} S^C$. Assume a certain operation $op \in Op$.

(DR1) and (DR2) : Let $(r,s) \in abs$ and $s \in pre_{S^A}(\langle \iota, op \rangle)$. There exists a history $h \in \mathcal{L}_{S^A} \cap \mathcal{L}_{S^C}$ with $s \in final_{S^A}(h)$ and $r \in final_{S^C}(h)$. Therefore $\mathcal{I}(h)\langle \iota, op \rangle \in Tr(\mathcal{L}_{S^A})$. Since $\mathcal{L}_{S^C} \Subset \mathcal{L}_{S^A}$ we have $\mathcal{I}(h)\langle \iota, op \rangle \in Tr(\mathcal{L}_{S^C})$. Because S^C is demonic we may conclude $ptrace_{S^C}(\mathcal{I}(h)) \subseteq pre_{S^C}(\langle \iota, op \rangle)$, and so $r \in pre_{S^C}(\langle \iota, op \rangle)$.

Assume now $r' \in St^C$ and $\omega \in Out$ with $(\langle \iota, r \rangle, \langle r', \omega \rangle) \in op^{S^C}$. Hence we get $h\langle \langle \iota, op \rangle, \omega \rangle \in \mathcal{L}_{S^C}$ and $r' \in final_{S^C}(h\langle \langle \iota, op \rangle, \omega \rangle)$. From $\mathcal{L}_{S^C} \Subset \mathcal{L}_{S^A}$ we may conclude $h\langle \langle \iota, op \rangle, \omega \rangle \in \mathcal{L}_{S^C} \cap \mathcal{L}_{S^A}$ and so $final_{S^A}(h\langle \langle \iota, op \rangle, \omega \rangle) \neq \varnothing$. S^A is semi-deterministic and so we can find a state $s' \in final_{S^A}(h\langle \langle \iota, op \rangle, \omega \rangle)$ such that $(\langle \iota, s \rangle \langle s', \omega \rangle) \in op^{S^A}$, and we have $(r', s') \in abs$ according to our definition.

(SR2) : This is satisfied because for $r \in Init^C = final_{S^C}(\varepsilon)$ and $s \in Init^A = final_{S^A}(\varepsilon)$ we have $(r,s) \in abs$.

□

Note that the implication $i) \Rightarrow ii)$ of Theorem 2 holds without the assumption that S^A is demonic. Note also that if condition $i)$ or condition $ii)$ holds, that we can always use the abstraction relation

$$(r,s) \in abs \quad \text{iff} \quad \exists\, h \in \mathcal{L}_{S^C} \cap \mathcal{L}_{S^A} \,:\, r \in final_{S^C}(h) \wedge s \in final_{S^A}(h)$$

to show that $S^A \sqsubseteq_{abs} S^C$.

For the specifications and languages in the appendices, since $SA2$ is semi-deterministic and both $\mathcal{L}_A$ and $\mathcal{L}_B$ are demonic, and $\mathcal{L}_B \Subset \mathcal{L}_A$, we can conclude $SA2 \sqsubseteq SB1$. The abstraction relation that can be used to show this is

$$abs = \{(i, \perp) \,:\, i \in \mathbb{Z}\} \cup \{(i,i) \,:\, i \in \mathbb{Z}\}$$

Similarly, we can use the above theorem to prove $SA2 \sqsubseteq SA1$ and the same abstraction relation applies in this case.

The theorem also proves that there is no abstraction relation so that $SB1$ forward refines to either $SA1$ or $SA2$. Similarly, since it is not the case that $\mathcal{L}_C \Subset \mathcal{L}_F$, Theorem 1 proves that there is no abstraction relation so that $SF1$ forward refines to $SC1$. Note that we cannot use Theorem 2 for the last case, because $SC1$ is not demonic.

5. State-abstraction for semi-deterministic specifications

We now formalise our notion of state abstractness for demonic, semi-deterministic specifications. We show that languages and semi-deterministic specifications define right congruences in a natural way. Moreover, a partial ordering on these right-congruences characterises forward refinement with abstraction functions. This reflects our understanding of refinement proofs with respect to state abstraction.

Definition 12 *a) For any equivalence relation R on a set M and $r \in M$ we denote the corresponding equivalence class by $\overline{r} = \{m \in M \mid rRm\}$ and the quotient space $\{\overline{r} \mid r \in M\}$ by M/R. For equivalence relations R_1, R_2 on M and subsets M_1, M_2 of M, $R_1 \rhd M_1 \preceq R_2 \rhd M_2$ (read "relation R_1 restricted to M_1 refines relation R_2 restricted to M_2") if each equivalence class of R_1 restricted to M_1 is a subset of some equivalence class of R_2 restricted to M_2. If no restriction occurs we write $R_1 \preceq R_2$. Note that $\preceq$ defines a partial ordering on pairs of equivalence relations and subsets of M.*

b) A relation $R : \mathcal{H} \leftrightarrow \mathcal{H}$ *is right invariant if* $(\forall\, z \in \mathcal{H})(xRy \Rightarrow xzRyz)$. *An equivalence relation that is right invariant is called a right congruence. In this case the concatenation of any equivalence class* $\overline{h} \in R$ *with a history* $z \in \mathcal{H}$ *is well-defined by* $\overline{h}\, z \mathrel{\widehat{=}} \overline{hz}$.

c) We say that a right congruence $R : \mathcal{H} \leftrightarrow \mathcal{H}$ *defines language* $\mathcal{L}$ *if* $\mathcal{L} = \mathcal{H}$ *or* $\mathcal{L} = \mathcal{H} \backslash \overline{y}$, *for some* $y \in \mathcal{H}$.

d) For a language $\mathcal{L} \subseteq \mathcal{H}$ *and* $x, y \in \mathcal{H}$, $xR_{\mathcal{L}}y$ *iff* $(\forall\, z \in \mathcal{H})(xz \in \mathcal{L} \Leftrightarrow yz \in \mathcal{L})$.

e) For a specification S *we define a reflexive and symmetric relation* R^1_S,

$$xR^1_S y \quad \textit{iff} \quad (x, y \in \mathcal{L}_S \wedge \mathit{final}_S(x) \cap \mathit{final}_S(y) \neq \varnothing) \vee (x, y \notin \mathcal{L}_S)$$

on histories $x, y \in \mathcal{H}$. *Its transitive closure will be denoted by* R_S.

An arbitrary right congruence R typically defines more than one language, and every right congruence R on $\mathcal{H}$ trivially defines the language $\mathcal{H}$. Consider the language $\mathcal{L}_A$ and specifications $SA1$ and $SA2$ in the appendix. The equivalence relation $R_{\mathcal{L}_A}$ contains one equivalence class for all $x \notin \mathcal{L}_A$, one equivalence class for all histories $x \in \mathcal{L}_A$ ending in $\langle \mathit{random}, \perp \rangle$, and one equivalence class for every $v \in \mathbb{Z}$ containing all histories ending in $\langle \mathit{val}, v \rangle$ (the equivalence class for the integer 0 also contains the empty history ε). The equivalence relation R_{SA1} contains two equivalence classes: one for all $x \notin \mathcal{L}_A$ and the second one for all $x \in \mathcal{L}_A$. To see that all $x \in \mathcal{L}_A$ belong to one equivalence class, note that $\langle \mathit{random}, \perp \rangle \in \mathcal{L}_A$ and that $\mathit{final}_{SA1}(\langle \mathit{random}, \perp \rangle) = \mathbb{Z}$. Therefore, since $\mathit{final}_{SA1}(x) \in \mathbb{Z}$ for all $x \in \mathcal{L}_A$, we have

$$\mathit{final}_{SA1}(x) \cap \mathit{final}_{SA1}(\langle \mathit{random}, \perp \rangle) \neq \varnothing$$

For $SA2$, on the other hand, we have $R_{SA2} = R_{\mathcal{L}_A}$.

The following proposition states the correspondence between $R_{\mathcal{L}}$ and R_S.

Proposition 7 *i) For every language* $\mathcal{L}$, $R_{\mathcal{L}}$ *is a right congruence that defines* $\mathcal{L}$.

ii) Let R *be a right congruence that defines a language* $\mathcal{L}$. *Then* $R \preceq R_{\mathcal{L}}$.

iii) For every semi-deterministic specification S, R_S *is a right congruence that defines the language* $\mathcal{L}_S$ *with* $R_S \preceq R_{\mathcal{L}_S}$.

Proof. Assertion *i*) follows directly from the definition.

For *ii*): Let uRw for $u, w \in \mathcal{H}$. Assume further $z \in \mathcal{H}$ with $uz \in \mathcal{L}$. R is a right congruence and therefore $uzRwz$. Furthermore, R is a right congruence that defines $\mathcal{L}$, which means that we cannot have $uz \in \mathcal{L}$, $uzRwz$, and $wz \notin \mathcal{L}$. Hence, $wz \in \mathcal{L}$ and therefore $uR_{\mathcal{L}}\, w$.

For *iii*): We prove R_S is right invariant. Let $xR_S y$ with $x, y \in \mathcal{L}_S$. This means that there exist elements $x_i \in \mathcal{L}_S$, $i = 0, ..., n$ with

$$x = x_0\ ,\ x_n = y\ ,\ \mathit{final}(x_i) \cap \mathit{final}(x_{i+1}) \neq \varnothing\ ,\ i = 0, ..., n \perp 1$$

S is semi-deterministic and so we get for $z \in \mathcal{H}$, $xz \in \mathcal{L}_S \Leftrightarrow yz \in \mathcal{L}_S$, and if $xz \in \mathcal{L}_S$

$$final(x_i z) \cap final(x_{i+1} z) \neq \varnothing \ , \ i = 0, ..., n \perp 1$$

and hence $xzR_S yz$. Finally, since R_S is a right invariance that defines $\mathcal{L}_S$ it follows from *ii*) that $R_S \preceq R_{\mathcal{L}_S}$.

□

$SA1$ is an example of a specification where R_{SA1} is not right invariant and does not refine $R_{\mathcal{L}_A}$, which is because $SA1$ is not semi-deterministic.

We are going to reduce the nondeterminism of a semi-deterministic specification to real determinism in the state space, i.e., we transform the specification into a state-deterministic one. This will be done by using the equivalence relation that is induced on the state space via the above introduced equivalence relation R_S.

Definition 13 *Given a semi-deterministic specification*

$$S = (Op, St, In, Out, \textsc{Init}, \_^S)$$

we define an equivalence relation E_S on the state space St by $sE_S s'$ iff

$$\exists\, h \in \mathcal{L}_S \ : \ s, s' \in \cup\{final_S(h') \mid h' R_S h\} \ \vee \ s, s' \in St \setminus \cup\{final_S(h') \mid h' \in \mathcal{L}_S\}$$

We denote the corresponding quotient spaces by

$$\overline{St} = St/E_S \ , \ \overline{\textsc{Init}} = \textsc{Init}/E_S$$

In addition, we get a corresponding specification $\overline{S} = (Op, \overline{St}, In, Out, \overline{\textsc{Init}}, \_^{\overline{S}})$ if we define,

$$op^{\overline{S}} = \{(\langle \iota, \overline{s} \rangle, \langle \overline{s}', \omega' \rangle) \mid (\langle \iota, s \rangle, \langle s', \omega' \rangle) \in op^S\}, \textit{ for all } op \in Op$$

For the specification $SE1$ from the appendix, the state space of $\overline{SE1}$ consists of equivalence classes of states of $SE1$. Thus, each state of $\overline{SE1}$ contains a set of states of type $se1$ defined by

$se1$
$v : \mathbb{Z}$
$stuck : \mathbb{B}$

For example, the initial state of $\overline{SE1}$ is the equivalence class with the singleton set containing the element $s \in se1$ such that $s.v = 0$ and $\neg\ s.stuck$. Moreover, the states of $\overline{SE1}$ contain all equivalence classes that are a singleton set with one element s such that $\neg\ s.stuck$, plus the one equivalence class consisting of all elements s such that $s.v$ is any integer and $s.stuck$ is *true*. The latter equivalence class is a state in $\overline{SE1}$ because all these states in $SE1$ really represent the same abstract state (i.e., they are indistinguishable with respect to future behaviour).

In general, the correspondence between S and $\overline{S}$ is formulated in the following Proposition.

Proposition 8 *Let S be a semi-deterministic specification. Then,*

a) $\overline{S}$ is state-deterministic with $\mathcal{L}_{\overline{S}} = \mathcal{L}_S$ and $R_{\overline{S}} = R_S$

b) S is state-deterministic iff $S = \overline{S}$

c) S is demonic iff $\overline{S}$ is demonic

Proof. We first prove the following two properties:

i) $\forall\, s \in St,\ h \in \mathcal{L}_S\ :\ s \in \mathit{final}_S(h) \ \Rightarrow\ \overline{s} = \cup\{\mathit{final}_S(h') \mid h' R_S h\}$

ii) $\forall\, s \in St,\ h \in \mathcal{L}_{\overline{S}}\ :\ \overline{s} \in \mathit{final}_{\overline{S}}(h) \ \Rightarrow\ \overline{s} = \cup\{\mathit{final}_S(h') \mid h' R_S h\}$

Assertion *i)* is an immediate consequence from the definition of R_S and E_S. We prove *ii)* by induction on the length of $h \in \mathcal{L}_{\overline{S}}$.

Base case ($h = \varepsilon$): Let $\overline{s} \in \mathit{final}_{\overline{S}}(\varepsilon)$. Because of $\mathit{final}_{\overline{S}}(\varepsilon) = \overline{\mathit{INIT}}$ we find $s_0 \in \mathit{INIT}$ with $s E_S s_0$. From *i)* and $s_0 \in \mathit{final}_S(\varepsilon)$ we conclude

$$\overline{s} \;=\; \overline{s_0} \;=\; \cup\{\mathit{final}_S(h') \mid h' R_S \varepsilon\}$$

Induction step: Assume $h \in \mathcal{L}_{\overline{S}}$, $\mathit{last}(h) = \langle\langle \iota, op\rangle, \omega\rangle$ and $\overline{s} \in \mathit{final}_{\overline{S}}(h)$. Then we find $\overline{s_0} \in \mathit{final}_{\overline{S}}(\mathit{front}(h))$ with $(\langle \iota, \overline{s_0}\rangle, \langle \overline{s}, \omega\rangle) \in op^{\overline{S}}$. Our induction hypothesis gives us $\overline{s_0} = \cup\{\mathit{final}_S(h') \mid h' R_S\, \mathit{front}(h)\}$. The definition of $op^{\overline{S}}$ guarantees the existence of $s_0' \in \overline{s_0}$ and $s' \in \overline{s}$ with $(\langle \iota, s_0'\rangle, \langle s', \omega\rangle) \in op^S$. Hence, we find $h_0 \in \mathcal{L}_S$ with $s_0' \in \mathit{final}_S(h_0)$ and $h_0 R_S\, \mathit{front}(h)$. Note that $s' \in \mathit{final}_S(h_0\langle\langle \iota, op\rangle, \omega\rangle)$ and because of *i)* and the right invariance of R_S,

$$\begin{array}{rcl} \overline{s} \;=\; \overline{s'} & = & \cup\{\mathit{final}_S(h') \mid h' R_S h_0\langle\langle \iota, op\rangle, \omega\rangle\} \\ & = & \cup\{\mathit{final}_S(h') \mid h' R_S h\} \end{array}$$

This finishes the proof of *ii)*, which shows that $\overline{S}$ is state-deterministic. The inclusion $\mathcal{L}_S \subseteq \mathcal{L}_{\overline{S}}$ is obvious. For $h \in \mathcal{L}_{\overline{S}}$ we find $\overline{s} \in \mathit{final}_{\overline{S}}(h)$; from *ii)* we can conclude $\overline{s} = \cup\{\mathit{final}_S(h') \mid h' R_S h\}$ and therefore $\mathit{final}_S(h) \neq \varnothing$. Hence, $h \in \mathcal{L}_S$.

To see $R_S = R_{\overline{S}}$ we take $h_1, h_2 \in \mathcal{H}$. Note the following equivalences:

$$\begin{array}{rcl} h_1 R_S h_2 \wedge h_1, h_2 \in \mathcal{L}_S & \Leftrightarrow & \exists\, s \in \mathit{final}_S(h_1) : \overline{s} = \cup\{\mathit{final}_S(h') \mid h' R_S h_1\} \wedge \\ & & \qquad \overline{s} = \cup\{\mathit{final}_S(h') \mid h' R_S h_2\} \\ & \Leftrightarrow & \exists\, s \in St : \mathit{final}_{\overline{S}}(h_1) = \mathit{final}_{\overline{S}}(h_2) = \{\overline{s}\} \\ & \Leftrightarrow & h_1 R_{\overline{S}} h_2 \wedge h_1, h_2 \in \mathcal{L}_{\overline{S}} \end{array}$$

The first equivalence follows from *i)*, the second from $\mathcal{L}_S = \mathcal{L}_{\overline{S}}$ and *ii)*, and the third from the fact that $\overline{S}$ is state-deterministic.

The assertion b) follows from *i)* and *ii)*, and c) follows from Proposition 3 and $\mathcal{L}_S = \mathcal{L}_{\overline{S}}$:

$$S \text{ demonic } \Leftrightarrow \mathcal{L}_S \text{ demonic } \Leftrightarrow \mathcal{L}_{\overline{S}} \text{ demonic } \Leftrightarrow \overline{S} \text{ demonic}$$

□

For the specifications in the appendix, note that $SE1$ is semi-deterministic, but not state-deterministic, and hence $SE1 \neq \overline{SE1}$.

We now introduce a relation on pairs $(R, \mathcal{L})$ where $R : \mathcal{H} \leftrightarrow \mathcal{H}$ is a right congruence that defines a language $\mathcal{L}$.

Definition 14 $(R_1, \mathcal{L}_1) \leq (R_2, \mathcal{L}_2)$ *iff* $\mathcal{L}_1 \Subset \mathcal{L}_2 \ \wedge \ R_1 \rhd (\mathcal{L}_1 \cap \mathcal{L}_2) \preceq R_2$

$R_1 \rhd (\mathcal{L}_1 \cap \mathcal{L}_2)$ is an equivalence relation on $\mathcal{L}_1 \cap \mathcal{L}_2$, and the above definition requires that every equivalence class of $R_1 \rhd (\mathcal{L}_1 \cap \mathcal{L}_2)$ is a subset of an equivalence class of R_2. Note that the restriction of R_1 to $\mathcal{L}_1 \cap \mathcal{L}_2$ can be canonically extended to a right congruence on $\mathcal{H}$ that defines $\mathcal{L}_1 \cap \mathcal{L}_2$. Note also that if $\mathcal{L}_1 = \mathcal{L}_2$ then the relation $(R_1, \mathcal{L}_1) \leq (R_2, \mathcal{L}_2)$ reduces to $R_1 \preceq R_2$.

Proposition 9 *The relation* $\leq$ *defines a partial ordering on pairs* $(R, \mathcal{L})$ *as above.*

Proof. The reflexivity and antisymmetry of $\leq$ are immediate from the definition. For the transitivity assume $(R_1, \mathcal{L}_1) \leq (R_2, \mathcal{L}_2) \leq (R_3, \mathcal{L}_3)$. Then, $\mathcal{L}_1 \Subset \mathcal{L}_2 \Subset \mathcal{L}_3$ and so, with Proposition 5 *iii*), $\mathcal{L}_1 \cap \mathcal{L}_3 \subseteq \mathcal{L}_2$. Hence,

$$\begin{array}{lcl} R_1 \rhd (\mathcal{L}_1 \cap \mathcal{L}_3) & = & R_1 \rhd (\mathcal{L}_1 \cap \mathcal{L}_2 \cap \mathcal{L}_3) \\ & \preceq & R_2 \rhd (\mathcal{L}_2 \cap \mathcal{L}_3) \\ & \preceq & R_3 \end{array}$$

□

We will use this partial ordering to generalise the main results of [9] to demonic, semi-deterministic specifications.

Definition 15 *Let* $\mathcal{L}$ *be a language. We define the corresponding abstraction lattice by*

$$\mathcal{A}L_{\mathcal{L}} = \langle \{ R \mid R \textit{ right congruence that defines } \mathcal{L} \}, \preceq \rangle$$

Proposition 10 $\mathcal{A}L_{\mathcal{L}}$ *is a complete lattice with greatest element* $R_{\mathcal{L}}$ *and least element* $R^{-}_{\mathcal{L}}$*,*

$$h \, R^{-}_{\mathcal{L}} \, h' \textit{ iff } (h, h' \in \mathcal{L} \wedge h = h') \vee h, h' \in \mathcal{H} \setminus \mathcal{L}$$

Proof. Note that $\inf\{R_i \mid i \in I\} = \cap_{i \in I} R_i$ and $\sup\{R_i \mid i \in I\} = trcl(\cup_{i \in I} R_i)$ for any family $R_i \in \mathcal{A}L_{\mathcal{L}}$, $i \in I$, where *trcl* denotes the transitive closure of a relation. In this lattice, according to Proposition 7, $R_{\mathcal{L}}$ is the greatest element; the least element is the partition of $\mathcal{L}$ consisting entirely of singleton sets, $R^{-}_{\mathcal{L}}$.

□

We will see that the abstraction lattice characterises state abstraction on deterministic languages and their semi-deterministic specifications. If states are viewed as history summaries, then $R_{\mathcal{L}}$ summarises the histories as much as possible, and the minimum element does no summarisation at all. Proposition 7 tells us that the abstraction lattice for language $\mathcal{L}$ contains an element for every semi-deterministic specification S with language $\mathcal{L}$, i.e., R_S.

Consider the specification $SE1$ from the appendix. It is semi-deterministic and fully abstract in the sense that $R_{\mathcal{L}_E} = R_{SE1}$. To obtain a specification that is not fully abstract, we can simply change the specification of *random* from

random

$\Delta(v)$
$\neg \, stuck \Rightarrow v' = v + 1$

to

random

$\Delta(v)$
$v' = v + 1$

The new specification $SE2$ always increments the value of v in *random* and is thus state-deterministic. $SE1$ only increments the value of v when *stuck* is *false*, and leaves the value of v' unspecified when *stuck* is *true*. Note that this does not change the language defined by the specification, because when *stuck* is *true*, the future behaviour does not depend on the value of v any more. Now $(R_{SE2}, \mathcal{L}_E) \leq (R_{SE1}, \mathcal{L}_E)$ because every two histories distinguished by R_{SE1} are also distinguished by R_{SE2}. However, the converse is not true, since R_{SE2} distinguishes histories that R_{SE1} does not. Therefore, for $\mathcal{L}_E$ we have:

$$(R^{-}_{\mathcal{L}_E}, \mathcal{L}_E) \leq (R_{SE2}, \mathcal{L}_E) \leq (R_{SE1}, \mathcal{L}_E) = (R_{\mathcal{L}_E}, \mathcal{L}_E)$$

Unfortunately, the lattice $\mathcal{AL}_{\mathcal{L}}$ is not sufficient as a domain for data refinement proofs with nondeterministic languages. During a forward refinement, traces might lose corresponding output sequences. This is the reason why we have to consider all languages $\mathcal{L}'$ with $\mathcal{L}' \Subset \mathcal{L}$.

Definition 16 *We define the partially ordered set*

$$\mathcal{AS}_{\mathcal{H}} = \langle \{ (R, \mathcal{L}) \mid R \in \mathcal{AL}_{\mathcal{L}} \wedge \mathcal{L} \in \mathcal{L}an^{d}_{\mathcal{H}} \}, \leq \rangle$$

and call it the state abstraction poset of $\mathcal{H}$.

The greatest element of $\mathcal{AS}_{\mathcal{H}}$ is $(\{(x, y) \mid x = y = \varepsilon \vee x, y \in \mathcal{H} \setminus \{\varepsilon\}\}, \{\varepsilon\})$. In general, $\mathcal{AS}_{\mathcal{H}}$ is neither a lattice, nor a meet semi-lattice, nor a join semi-lattice, due to the nondeterminism in the underlying refinement ordering. Note that by adding a bottom element we obtain a chain complete poset, i.e. a poset where every chain has a supremum. In this poset, the infimum of a chain exists if for each language in the chain there is a smaller language in the chain which is finite, in the sense that the number of outputs for a given input is always finite.

We propose that the abstraction poset is used to characterise state abstractness on demonic, semi-deterministic specifications. Specifically, for two demonic, semi-deterministic specifications S^A, S^C we will say that S^C is less state abstract than S^A if

$$(R_{S^C}, \mathcal{L}_{S^C}) \leq (R_{S^A}, \mathcal{L}_{S^A})$$

For the language $\mathcal{L}_E$ from the appendix, we have shown a number of elements in the state abstraction lattice. However, for all these, the language $\mathcal{L}_E$ is the same. For an example of two elements in a state abstraction poset for which the languages are not the same, consider the language $SG1$ obtained from $SA1$ by adding the operation *zero* and the state variable *stuck* that behave in the same way as in $SE1$. $SG1$ is not semi-deterministic. We therefore change $SG1$ to $SG2$ using the same technique that was used

to turn $SA1$ into a state-deterministic specification $SA2$ by introducing the special value $\perp$ to indicate that *random* has been called without being followed by a call to *val*. Then $SG2$ is state-deterministic and $(R_{SE1}, \mathcal{L}_E) \leq (R_{SG2}, \mathcal{L}_G)$. Note that $\mathcal{L}_E \neq \mathcal{L}_G$, but that $\mathcal{L}_E \Subset \mathcal{L}_G$.

We have seen that every semi-deterministic specification defines a right-congruence in a natural way. The converse is true as well. Every element of $\mathcal{AS}_{\mathcal{H}}$ determines a semi-deterministic specification in the following sense.

Definition 17 *For a right congruence $R : \mathcal{H} \to \mathcal{H}$ that defines a language $\mathcal{L} \subseteq \mathcal{H}$ we define a specification $S(R, \mathcal{L}) = (Op, St^{S(R,\mathcal{L})}, In, Out, \textit{INIT}^{S(R,\mathcal{L})}, \_^{S(R,\mathcal{L})})$ with state and initialisation spaces*

$$St^{S(R,\mathcal{L})} = \{\overline{h} \mid h \in \mathcal{L}\} \ , \ \textit{INIT}^{S(R,\mathcal{L})} = \{\overline{\varepsilon}\}$$

Then, for each operation $op \in Op$ we define the following state transitions on St:

$$op^{S(R,\mathcal{L})} = \{(\langle \iota, t\rangle, \langle t', \omega\rangle) \mid \iota \in In \wedge \omega \in Out \wedge t, t' \in St^{S(R,\mathcal{L})} \wedge t' = t\,\langle\langle \iota, op\rangle, \omega\rangle\}$$

Proposition 11 *Let $R : \mathcal{H} \to \mathcal{H}$ be a right congruence that defines a language $\mathcal{L} \subseteq \mathcal{H}$. Then the following properties hold.*

i) $S(R, \mathcal{L})$ is state-deterministic with $R_{S(R,\mathcal{L})} = R$ and $\mathcal{L}_{S(R,\mathcal{L})} = \mathcal{L}$.

ii) $S(R, \mathcal{L})$ demonic $\Leftrightarrow$ $\mathcal{L}$ demonic.

Proof. For i): By induction over the length of histories in $\mathcal{L}_{S(R,\mathcal{L})}$, respectively $\mathcal{L}$, we prove

a) $\forall\, h \in \mathcal{L}_{S(R,\mathcal{L})} : h \in \mathcal{L} \wedge \textit{final}_{S(R,\mathcal{L})}(h) = \{\overline{h}\}$

b) $\mathcal{L} \subseteq \mathcal{L}_{S(R,\mathcal{L})}$

For a): Base case ($h = \varepsilon$): $\textit{final}_{S(R,\mathcal{L})}(\varepsilon) = \textit{INIT}^{S(R,\mathcal{L})} = \{\overline{\varepsilon}\}$.

Induction step: Let $h \in \mathcal{L}_{S(R,\mathcal{L})}$ with $\textit{last}(h) = \langle\langle \iota, op\rangle, \omega\rangle$. Since $\mathcal{L}_{S(R,\mathcal{L})}$ is prefix-closed, $\textit{front}(h) \in \mathcal{L}_{S(R,\mathcal{L})}$ and by the induction hypothesis,

$$\textit{final}_{S(R,\mathcal{L})}(\textit{front}(h)) = \{\overline{\textit{front}(h)}\}$$

R is a right invariance, hence $\overline{\textit{front}(h)}\,\textit{last}(h) = \overline{h}$. We therefore have $\textit{final}_{S(R,\mathcal{L})}(h) \neq \varnothing$ and there exists $t' \in St$ such that $(\langle \iota, \overline{\textit{front}(h)}\rangle, \langle t', \omega\rangle) \in op^{S(R,\mathcal{L})}$. The definition of $op^{S(R,\mathcal{L})}$ gives us $t' = \overline{\textit{front}(h)}\,\textit{last}(h) = \overline{h}$. Therefore, $h \in \mathcal{L}$ and $\textit{final}_{S(R,\mathcal{L})}(h) = \{\overline{h}\}$.

For b): Induction step: Let $h \in \mathcal{L}$ with $\textit{last}(h) = \langle\langle \iota, op\rangle, \omega\rangle$. By the induction hypothesis, we have $\textit{front}(h) \in \mathcal{L}_{S(R,\mathcal{L})}$. Because of a), $\textit{final}_{S(R,\mathcal{L})}(\textit{front}(h)) = \{\overline{\textit{front}(h)}\}$. Hence, $(\langle \iota, \overline{\textit{front}(h)}\rangle, \langle \overline{h}, \omega\rangle) \in op^{S(R,\mathcal{L})}$ and we are done with

$$h = \textit{front}(h)\,\textit{last}(h) \in \mathcal{L}_{S(R,\mathcal{L})}$$

From a) follows it that $S(R,\mathcal{L})$ is state-deterministic; a) and b) imply $\mathcal{L} = \mathcal{L}_{S(R,\mathcal{L})}$. Now, for $h_1, h_2 \in \mathcal{H}$,

$$\begin{array}{rcl} h_1 R h_2 \wedge h_1, h_2 \in \mathcal{L} & \Leftrightarrow & \mathit{final}_{S(R,\mathcal{L})}(h_1) = \{\overline{h_1}\} = \mathit{final}_{S(R,\mathcal{L})}(h_2) \\ & \Leftrightarrow & h_1 R_{S(R,\mathcal{L})} h_2 \wedge h_1, h_2 \in \mathcal{L}_{S(R,\mathcal{L})} \end{array}$$

and therefore, $R = R_{S(R,\mathcal{L})}$.

For ii): With i) and Proposition 3 we get that $S(R,\mathcal{L})$ is demonic iff $\mathcal{L}$ is demonic.

□

We next show that the ordering $\leq$ on the state-abstraction poset $\mathcal{AS}_{\mathcal{H}}$ has a concrete interpretation in the context of demonic, semi-deterministic specifications.

Theorem 3 *Assume demonic, semi-deterministic specifications S^A and S^C. Then the following properties are equivalent.*

i) $(R_{S^C}, \mathcal{L}_{S^C}) \leq (R_{S^A}, \mathcal{L}_{S^A})$

ii) $\overline{S}^A \sqsubseteq_{abs_1} \overline{S}^C$, *with a partial function* $abs_1 : \overline{St}^C \nrightarrow \overline{St}^A$

iii) $S(R_{S^A}, \mathcal{L}_{S^A}) \sqsubseteq_{abs_2} S(R_{S^C}, \mathcal{L}_{S^C})$, *with a function* $abs_2 : (\mathcal{L}_{S^A} \cap \mathcal{L}_{S^C})/R_{S^C} \to \mathcal{L}_{S^A}/R_{S^A}$

Proof. By Proposition 8 we get the demonic, state-deterministic specifications $\overline{S}^A$ and $\overline{S}^C$ with $\mathcal{L}_{\overline{S}^A} = \mathcal{L}_{S^A}$, $\mathcal{L}_{\overline{S}^C} = \mathcal{L}_{S^C}$, $R_{\overline{S}^A} = R_{S^A}$ and $R_{\overline{S}^C} = R_{S^C}$.

i) $\Rightarrow$ *ii)*: With $R_{S^C} \rhd (\mathcal{L}_{S^C} \cap \mathcal{L}_{S^A}) \preceq R_{S^A}$ and $\mathcal{L}_{S^C} \Subset \mathcal{L}_{S^A}$ there exists a natural embedding

$$abs_2 : (\mathcal{L}_{S^C} \cap \mathcal{L}_{S^A})/R_{S^C} \to \mathcal{L}_{S^A}/R_{S^A} \ , \ abs_2(\overline{h}) = \overline{h} \tag{2}$$

Now we define a relation $abs_1 : \overline{St}^C \leftrightarrow \overline{St}^A$, by

$$(t,s) \in abs_1 \text{ iff } \exists\, h \in \mathcal{L}_{S^A} \cap \mathcal{L}_{S^C} \ : \ \{s\} = \mathit{final}_{\overline{S}^A}(h) \wedge \{t\} = \mathit{final}_{\overline{S}^C}(h)$$

From the remark following Theorem 2, we conclude $\overline{S}^A \sqsubseteq_{abs_1} \overline{S}^C$. It remains to prove that abs_1 is a partial function from $\overline{St}^C$ to $\overline{St}^A$. Let $(t,s_1), (t,s_2) \in abs_1$, then there exist $h_1, h_2 \in \mathcal{L}_{S^A} \cap \mathcal{L}_{S^C}$ with

a) $\mathit{final}_{\overline{S}^C}(h_1) = \{t\} = \mathit{final}_{\overline{S}^C}(h_2)$

b) $\mathit{final}_{\overline{S}^A}(h_1) = \{s_1\}$, $\mathit{final}_{\overline{S}^A}(h_2) = \{s_2\}$

From a) it follows that $h_1 R_{S^C} h_2$. Then, the embedding abs_2 in (2) gives us $h_1 R_{S^A} h_2$ which is equivalent to $h_1 R_{\overline{S}^A} h_2$. Recall that $\overline{S}^A$ is state-deterministic, and hence $s_1 = s_2$.

ii) $\Rightarrow$ *i)*: Assume $\overline{S}^A \sqsubseteq_{abs} \overline{S}^C$ with a partial function $abs_1 : \overline{St}^C \nrightarrow \overline{St}^A$. By applying Theorem 1 we get $\mathcal{L}_{S^C} \Subset \mathcal{L}_{S^A}$. Then, because of Lemma 1 we know that the abstraction function abs_1 satisfies for every $h \in \mathcal{L}_{S^C} \cap \mathcal{L}_{S^A}$ the following condition:

c) $\forall\, t \in \overline{St}^C \ : \ \{t\} = \mathit{final}_{\overline{S}^C}(h) \Rightarrow t \in \mathrm{dom}(abs_1) \wedge \{abs_1(t)\} = \mathit{final}_{\overline{S}^A}(h)$

Obligation c) asserts that the embedding (2) is well-defined. This finishes the proof of $ii) \Rightarrow i)$.

$i) \Leftrightarrow iii)$: Because of Propositions 11, 8 we have $\overline{S(R_{S^A}, \mathcal{L}_{S^A})} = S(R_{S^A}, \mathcal{L}_{S^A})$ and $\overline{S(R_{S^C}, \mathcal{L}_{S^C})} = S(R_{S^C}, \mathcal{L}_{S^C})$. Then, equivalence $i) \Leftrightarrow iii)$ is an immediate consequence of equivalence $i) \Leftrightarrow ii)$ with the specifications $S(R_{S^A}, \mathcal{L}_{S^A})$ and $S(R_{S^C}, \mathcal{L}_{S^C})$.

□

From statement c) it follows that any abstraction function abs_1 that satisfies $ii)$, is uniquely determined on the set $\bigcup \{ \mathit{final}_{\overline{S}^C}(h) \mid h \in \mathcal{L}_{S^A} \cap \mathcal{L}_{S^C} \}$ and satisfies

$$\forall\, h \in \mathcal{L}_{S^A} \cap \mathcal{L}_{S^C}, t \in \overline{St}^C :$$
$$\{t\} = \mathit{final}_{\overline{S}^C}(h) \Rightarrow (t \in \mathrm{dom}(abs_1) \wedge \{abs_1(t)\} = \mathit{final}_{\overline{S}^A}(h))$$

Obviously, for two demonic, semi-deterministic specifications S^A and S^C, equality in $\mathcal{AS}_{\mathcal{H}}$ means $R_{S^A} = R_{S^C}$ and $\mathcal{L}_{S^A} = \mathcal{L}_{S^C}$. From Theorem 3 we can now deduce what this equality means in terms of the underlying state spaces: two demonic, semi-deterministic specifications S^A and S^C are equal in $\mathcal{AS}_{\mathcal{H}}$ iff there is a partial injection $abs : \overline{St}^C \nrightarrow \overline{St}^A$ such that $\overline{S}^A \sqsubseteq_{abs} \overline{S}^C \sqsubseteq_{abs^{-1}} \overline{S}^A$.

The following special case of Theorem 3 extends the main result of [9] for deterministic and complete languages: for a demonic language $\mathcal{L}$, two elements in the abstraction lattice $\mathcal{AL}_{\mathcal{L}}$ are related if and only if there exists a functional refinement for the underlying demonic, semi-deterministic specifications. Recall that a semi-deterministic specification is demonic exactly if it defines a demonic language.

Corollary 1 *Assume demonic, semi-deterministic specifications S^A and S^C that define the same language. Then the following properties are equivalent.*

i) $R_{S^C} \preceq R_{S^A}$

ii) There exists a partial function $abs : \overline{St}^C \nrightarrow \overline{St}^A$ *such that* $\overline{S}^A \sqsubseteq_{abs} \overline{S}^C$.

6. Completeness of refinement

We have seen that refinement with obligations $(SR1)$ and $(SR2)$ is a sound and complete method for demonic, semi-deterministic specifications. But what happens if the specifications are not semi-deterministic? In this case, forward refinement is not complete. In other words, there exist demonic specifications S^A and S^C with languages $\mathcal{L}_{S^C} \Subset \mathcal{L}_{S^A}$ where S^A cannot be refined to S^C with $(SR1)$ and $(SR2)$. For example, $SA1$ does not refine to $SA2$, even though both are demonic and define the same language.

This reflects the similarity with common forward refinement techniques which are not complete, unless an adequate backward refinement technique is added [13,10,15].

In this section, we define a backward refinement technique to also obtain a complete proof method in our framework. We show that this method together with forward refinement is sound and complete, i.e., for specifications S^A and S^C with languages $\mathcal{L}_{S^C} \Subset \mathcal{L}_{S^A}$ we have a refinement from S^A to S^C with a combination of forward and backward refinement. To accomplish this, we construct intermediate specifications.

We first define a notion of backward refinement for operations in our context.

Definition 18 *Given two specifications* $S^A = (Op, St^A, In, Out, \textsc{Init}^A, \_^{S^A})$ *and* $S^C = (Op, St^C, In, Out, \textsc{Init}^C, \_^{S^C})$ *with a relation*

$$abs : St^C \leftrightarrow St^A$$

and an operation $op \in Op$*, we say that* op^{S^A} *backward refines to* op^{S^C} ($op^{S^A} \sqsubseteq'_{abs} op^{S^C}$) *if the following obligations are fulfilled.*

(DR1') $\forall \iota \in In : pre_{S^A}(\langle \iota, op \rangle) \neq \varnothing \Rightarrow (pre_{S^C}(\langle \iota, op \rangle) \neq \varnothing \wedge$
$(\forall t \in post_{S^C}(\langle \iota, op \rangle)\, \exists s \in post_{S^A}(\langle \iota, op \rangle) : (t, s) \in abs) \wedge$
$(\forall t \in post_{S^C}(\langle \iota, op \rangle),\ s \in St^A : (t, s) \in abs \Rightarrow s \in post_{S^A}(\langle \iota, op \rangle)))$

(DR2') $\forall \iota \in In, \omega \in Out, s, s' \in St^A, t \in St^C :$
$((t, s) \in abs \wedge (\langle \iota, s \rangle, \langle s', \omega \rangle) \in op^{S^A}) \Rightarrow$
$(\exists t' \in St^C : (\langle \iota, t \rangle, \langle t', \omega \rangle) \in op^{S^C} \wedge (t', s') \in abs)$

(DR3') $\forall \iota \in In, \omega \in Out, s' \in St^A, t, t' \in St^C :$
$((t', s') \in abs \wedge s' \in post_{S^A}(\langle \iota, op \rangle) \wedge (\langle \iota, t \rangle, \langle t', \omega \rangle) \in op^{S^C}) \Rightarrow$
$(\exists s \in St^A : (\langle \iota, s \rangle, \langle s', \omega \rangle) \in op^{S^A} \wedge (t, s) \in abs)$

This relation defines a preorder on operations which is different from the common backward refinement notions [8,13,10,11,15,17], as explained in Section 7.

Definition 19 *We say that a specification* S^A *can be backward refined to specification* S^C *if there exists an abstraction relation abs, as above, such that*

(SR1') $\forall op \in Op : op^{S^A} \sqsubseteq'_{abs} op^{S^C}$

(SR2') $(\forall s \in \textsc{Init}^A\, \exists t \in \textsc{Init}^C : (t, s) \in abs) \wedge$
$(\forall s \in St^A, t \in \textsc{Init}^C : (t, s) \in abs \Rightarrow s \in \textsc{Init}^A)$

In this case, we write $S^A \sqsubseteq'_{abs} S^C$ *or simply* $S^A \sqsubseteq' S^C$*.*

Note that we use the symbol $\sqsubseteq'$ in different contexts for operation and specification refinement.

As for forward refinement, backward refinement as defined above defines a preorder.

Proposition 12 *The relation* $\sqsubseteq'$ *defines a preorder on specifications* S*.*

The following propositions and lemmas are used to show that forward refinement $\sqsubseteq$ and backward refinement $\sqsubseteq'$ together form a sound and complete proof method for specification refinement. We first concentrate on the soundness of backward refinement and then define the intermediate specifications that are needed to show that forward and backward refinement together are complete.

Now, recall the restricted-use specification $S^C[S^A]$ from Definition 11. Similar to forward refinement, $S^C[S^A]$ defines how S^A is simulated in S^C under a backward refinement.

Proposition 13 *For specifications* S^A *and* S^C*:*

i) $S^C[S^A] \sqsubseteq'_{id} S^C$ *with the identity id on* St^C

ii) $S^A \sqsubseteq'_{abs} S^C \Leftrightarrow S^A \sqsubseteq'_{abs} S^C[S^A]$

Proof. *i)* is straightforward and that $S^A \sqsubseteq'_{abs} S^C[S^A]$ implies $S^A \sqsubseteq'_{abs} S^C$ is a consequence of *i)* and the transitivity of $\sqsubseteq'$. It remains to prove that $S^A \sqsubseteq'_{abs} S^C$ implies $S^A \sqsubseteq'_{abs} S^C[S^A]$. We assume $S^A \sqsubseteq'_{abs} S^C$ and an operation $op \in Op$:

For $(DR1')$: Let $pre_{S^A}(\langle \iota, op \rangle) \neq \varnothing$. Because of $(DR1')$ for the refinement $S^A \sqsubseteq'_{abs} S^C$ we have $pre_{S^C}(\langle \iota, op \rangle) \neq \varnothing$ and hence by definition of $S^C[S^A]$, $pre_{S^C[S^A]}(\langle \iota, op \rangle) \neq \varnothing$. Now, assume $r \in post_{S^C[S^A]}(\langle \iota, op \rangle)$. Then, $r \in post_{S^C}(\langle \iota, op \rangle)$ and because of $(DR1')$, there exists $s \in post_{S^A}(\langle \iota, op \rangle)$ such that $(r, s) \in abs$. But, if $r \in post_{S^C[S^A]}(\langle \iota, op \rangle)$ and $s \in St^A$ with $(r, s) \in abs$, then $r \in post_{S^C}(\langle \iota, op \rangle)$. Again, because of $(DR1')$ we can conclude that $s \in post_{S^A}(\langle \iota, op \rangle)$.

For $(DR2')$: Let $(r, s) \in abs$ and $(\langle \iota, s \rangle, \langle s', \omega \rangle) \in op^{S^A}$. $(DR2')$ implies the existence of $r' \in St^C$ such that $(\langle \iota, r \rangle, \langle r', \omega \rangle) \in op^{S^C}$. Hence, by the definition of $S^C[S^A]$, $(\langle \iota, r \rangle, \langle r', \omega \rangle) \in op^{S^C[S^A]}$.

For $(DR3')$: Let $(r', s') \in abs$ and $(\langle \iota, r \rangle, \langle r', \omega \rangle) \in op^{S^C[S^A]}$ with $s' \in post_{S^A}(\langle \iota, op \rangle)$. Then, $(\langle \iota, r \rangle, \langle r', \omega \rangle) \in op^{S^C}$ and by condition $(DR3')$, there exists $s \in St^A$ such that $(\langle \iota, s \rangle, \langle s', \omega \rangle) \in op^{S^A}$ and $(r, s) \in abs$.

$(SR2')$ is fulfilled, because S^C and $S^C[S^A]$ have the same initialisation sets.

□

The next theorem states the soundness of backward refinement. Before that, we prove two lemmas.

Lemma 3 *Assume specifications* S^A *and* S^C. *If there exists an abstraction relation* $abs : St^C \leftrightarrow St^A$ *such that* $S^A \sqsubseteq'_{abs} S^C$, *then for every* $h \in \mathcal{L}_{S^C[S^A]}$:

i) $\forall\, s \in final_{S^A}(h)\ \exists\, r \in final_{S^C[S^A]}(h) : (r, s) \in abs$

ii) $\forall\, s \in St^A,\ r \in final_{S^C[S^A]}(h) : (r, s) \in abs \Rightarrow s \in final_{S^A}(h)$

iii) $final_{S^A}(h) \neq \varnothing$

Proof. We prove our assertion by induction on the length of $h \in \mathcal{L}_{S^C[S^A]}$:

Base case $(h = \varepsilon)$: In this case *i)* and *ii)* follow from $(SR2')$ and *iii)* is fulfilled because $INIT^A \neq \varnothing$.

Induction step: Let $h\langle \langle \iota, op \rangle, \omega \rangle \in \mathcal{L}_{S^C[S^A]}$.

For *i)*: Assume $s' \in final_{S^A}(h\langle \langle \iota, op \rangle, \omega \rangle)$. Let $s \in final_{S^A}(h)$ with $(\langle \iota, s \rangle, \langle s', \omega \rangle) \in op^{S^A}$. From the induction hypothesis *i)* we can conclude $r \in final_{S^C[S^A]}(h)$, and thus $r \in final_{S^C}(h)$, with $(r, s) \in abs$. Because of $(DR2')$ we find $r' \in St^C$ such that $(\langle \iota, r \rangle, \langle r', \omega \rangle) \in op^{S^C}$, and hence $(\langle \iota, r \rangle, \langle r', \omega \rangle) \in op^{S^C[S^A]}$ with $(r', s') \in abs$.

For *ii)*: Let $s' \in St^A$, $r' \in final_{S^C[S^A]}(h\langle \langle \iota, op \rangle, \omega \rangle)$ with $(r', s') \in abs$. Hence, there exists $r \in final_{S^C[S^A]}(h)$ with $(\langle \iota, r \rangle, \langle r', \omega \rangle) \in op^{S^C[S^A]}$. According to the definition of $S^C[S^A]$ we have $pre_{S^A}(\langle \iota, op \rangle) \neq \varnothing$. Because of $(DR1')$ we get $s' \in post_{S^A}(\langle \iota, op \rangle)$. $(DR3')$ then leads to $s \in St^A$ with $(\langle \iota, s \rangle, \langle s', \omega \rangle) \in op^{S^A}$ and $(r, s) \in abs$. Our induction hypothesis *ii)* then allows us to conclude $s \in final_{S^A}(h)$ and hence, $s' \in final_{S^A}(h\langle \langle \iota, op \rangle, \omega \rangle)$.

For *iii)*: From $final_{S^C[S^A]}(h\langle\langle \iota, op\rangle, \omega\rangle) \neq \varnothing$, we can deduce $pre_{S^A}(\langle \iota, op\rangle) \neq \varnothing$. We then fix $r' \in final_{S^C[S^A]}(h\langle\langle \iota, op\rangle, \omega\rangle)$. Then, $r' \in post_{S^C[S^A]}(\langle \iota, op\rangle)$. Because of $(DR1')$ we find $s' \in post_{S^A}(\langle \iota, op\rangle)$ with $(r', s') \in abs$. From *ii)* we conclude $s' \in final_{S^A}(h\langle\langle \iota, op\rangle, \omega\rangle)$.

□

Lemma 4 *Assume specifications S^A and S^C. If there exists an abstraction relation $abs : St^C \leftrightarrow St^A$ such that $S^A \sqsubseteq'_{abs} S^C$, then for every $h \in \mathcal{L}_{S^A}$:*

$$\forall\, s \in final_{S^A}(h)\ \exists\, r \in final_{S^C[S^A]}(h)\ :\ (r, s) \in abs$$

Proof. We prove our assertion by induction on the length of $h \in \mathcal{L}_{S^A}$:

Base case ($h = \varepsilon$): In this case our assertion is fulfilled because of $(SR2')$.

Induction step: Let $h\langle\langle \iota, op\rangle, \omega\rangle \in \mathcal{L}_{S^A}$. Let $s' \in final_{S^A}(h\langle\langle \iota, op\rangle, \omega\rangle)$. Then, we find $s \in final_{S^A}(h)$ with $(\langle \iota, s\rangle, \langle s', \omega\rangle) \in op^{S^A}$. By the induction hypothesis, we can find an $r \in final_{S^C[S^A]}(h)$ with $(r, s) \in abs$. Because of $(DR2')$ there exists $r' \in St^C$ such that $(\langle \iota, r\rangle, \langle r', \omega\rangle) \in op^{S^C[S^A]}$ and $(r', s') \in abs$.

□

Theorem 4 *Let S^A and S^C be specifications with $S^A \sqsubseteq'_{abs} S^C$. Then,*

i) $\mathcal{L}_{S^C} \Subset \mathcal{L}_{S^A}$ *and* $\mathcal{L}_{S^C[S^A]} = \mathcal{L}_{S^A} = \mathcal{L}_{S^A} \cap \mathcal{L}_{S^C}$

ii) $S^C[S^A]$ *is demonic if*

$$S^A \text{ is demonic and } \forall\, t \in \mathit{INIT}^C\, \exists\, s \in St^A\ :\ (t, s) \in abs \tag{3}$$

Proof. For *i)*: By the definition of $S^C[S^A]$, $\mathcal{L}_{S^C[S^A]} \subseteq \mathcal{L}_{S^C}$. Furthermore, from Lemma 4, $\mathcal{L}_{S^A} \subseteq \mathcal{L}_{S^C[S^A]}$ and from Lemma 3, $\mathcal{L}_{S^C[S^A]} \subseteq \mathcal{L}_{S^A}$. This proves $\mathcal{L}_{S^C[S^A]} = \mathcal{L}_{S^A} = \mathcal{L}_{S^A} \cap \mathcal{L}_{S^C}$.

For $\mathcal{L}_{S^C} \Subset \mathcal{L}_{S^A}$ we prove

$$\forall\, t \in Tr(\mathcal{L}_{S^A})\ :\ \nabla_{\mathcal{L}_{S^C}}(t) \subseteq \nabla_{\mathcal{L}_{S^A}}(t)$$

by induction on the length of the traces:

Induction step: Let $t = t'\langle \iota, op\rangle \in Tr(\mathcal{L}_{S^A})$ and $h = h'\langle\langle \iota, op\rangle, \omega\rangle \in \nabla_{\mathcal{L}_{S^C}}(t)$. Then we find $r \in final_{S^C}(h')$, $r' \in St^C$ with $(\langle \iota, r\rangle, \langle r', \omega\rangle) \in op^{S^C}$. Since $pre_{S^A}(\langle \iota, op\rangle) \neq \varnothing$ and $(DR1')$, there exists $s' \in post_{S^A}(\langle \iota, op\rangle)$ with $(r', s') \in abs$. Then, $(DR3')$ gives us the existence of $s \in St^A$ with $(\langle \iota, s\rangle, \langle s', \omega\rangle) \in op^{S^A}$ and $(r, s) \in abs$. Finally, our induction hypothesis together with Lemma 3 *ii)* ensure that $s \in final_{S^A}(h')$, and therefore $h \in \mathcal{L}_{S^A}$.

For *ii)*: Let $t\langle \iota, op\rangle \in Tr(S^C[S^A])$. We have to show $ptrace_{S^C[S^A]}(t) \subseteq pre_{S^C[S^A]}(\langle \iota, op\rangle)$. Let $r \in ptrace_{S^C[S^A]}(t)$.

First case, $t = \varepsilon$: Then, $r \in \mathit{INIT}^C$. Because of condition (3) we find $s \in St^A$, and $(SR2')$ implies $s \in \mathit{INIT}^A$. S^A is demonic and so, $s \in pre_{S^A}(\langle \iota, op\rangle)$. $(DR2')$ implies $r \in pre_{S^C[S^A]}(\langle \iota, op\rangle)$.

Second case, $t \neq \varepsilon$: Because of $(DR1')$ we find $s \in ptrace_{S^A}(t)$ with $(r, s) \in abs$. S^A is demonic, hence $s \in pre_{S^A}(\langle \iota, op\rangle)$ and with $(DR2')$ we conclude $r \in pre_{S^C[S^A]}(\langle \iota, op\rangle)$.

□

The above result proves the soundness of backward refinement with respect to the ordering $\Subset$. To prove the completeness of the combination of forward and backward refinement, we construct two intermediate specifications.

Definition 20 *Let* $S = (Op,\ St,\ In,\ Out,\ \textsc{Init},\ \_^S)$ *be a specification. We define the corresponding tight specification* $\widehat{S} = (Op, St, In, Out, \textsc{Init}, \_^{\widehat{S}})$ *with*

$$op^{\widehat{S}} = \{(\langle \iota, s\rangle, \langle s', \omega'\rangle) \in op^S \mid \exists\, h \in \mathcal{L}_S \ :\ s \in \mathit{final}_S(h)\}\,,\ \ \textit{for all } op \in Op$$

Each $op^{\widehat{S}}$ is created by restricting the pre- and postconditions of the operation op^S as much as possible without changing the overall behaviour of the specification S. It is obvious that $\widehat{S}$ has exactly the same behaviour as S.

Lemma 5 *Assume a specification* S. *Then* $S \sqsubseteq_{abs} \widehat{S}$ *and* $\widehat{S} \sqsubseteq_{abs} S$ *with*

$$(r, s) \in abs \ \ \textit{iff}\ \ (r = s \wedge \exists\, h \in \mathcal{L}_S \ :\ s \in \mathit{final}_S(h))$$

Additionally, $\mathcal{L}_S = \mathcal{L}_{\widehat{S}}$ *and* $\widehat{S}$ *is demonic iff* S *is.*

Proof. The proof is a straightforward verification of the data refinement rules $(DR1)$, $(DR2)$, and $(SR2)$.

For $(DR1)$ and $(DR2)$: Let $(r, s) \in abs$. Then, $r = s$ and there exists $h \in \mathcal{L}_S$ with $s \in \mathit{final}_S(h)$. By definition of $\widehat{S}$, we get $s \in \mathit{pre}_S(\langle \iota, op\rangle) \Leftrightarrow s \in \mathit{pre}_{\widehat{S}}(\langle \iota, op\rangle)$.

In addition, we have

$$(\langle \iota, s\rangle, \langle s', \omega\rangle) \in op^{\widehat{S}} \wedge (s', s') \in abs \Leftrightarrow (\langle \iota, s\rangle, \langle s', \omega\rangle) \in op^S \wedge (s', s') \in abs$$

$(SR2)$ is obvious because the same initialisation sets are used in S and $\widehat{S}$.

Obviously, $\mathcal{L}_{\widehat{S}} \subseteq \mathcal{L}_S$ and to show $\mathcal{L}_S \subseteq \mathcal{L}_{\widehat{S}}$, we prove $\forall\, h \in \mathcal{L}_S \ :\ \mathit{final}_S(h) = \mathit{final}_{\widehat{S}}(h)$ by induction on the length of $h \in \mathcal{L}_S$:

Base case $(h = \varepsilon)$: $\mathit{final}_S(\varepsilon) = \textsc{Init} = \mathit{final}_{\widehat{S}}(\varepsilon)$.

Induction step: Let $h\langle\langle \iota, op\rangle, \omega\rangle \in \mathcal{L}_S$. From the induction hypothesis, $\mathit{final}_S(h) = \mathit{final}_{\widehat{S}}(h)$. The definition of $op^{\widehat{S}}$ then implies $\mathit{final}_S(h\langle\langle \iota, op\rangle, \omega\rangle) = \mathit{final}_{\widehat{S}}(h\langle\langle \iota, op\rangle, \omega\rangle)$.

Of course, this implies for every trace $t \in Tr(\mathcal{L}_s)$ that $\mathit{ptrace}_S(t) = \mathit{ptrace}_{\widehat{S}}(t)$ and the remaining assertion, S demonic iff $\widehat{S}$ demonic, is an immediate consequence.

□

With the same notations as above we prove that $\widehat{S}$ backward refines to $S(R^-_{\mathcal{L}_S}, \mathcal{L}_S)$. Recall that the state space $St^{S(R^-_{\mathcal{L}_S}, \mathcal{L}_S)}$ can be identified with the set $\mathcal{L}_S \cup \{\mathcal{H} \setminus \mathcal{L}_S\}$.

Lemma 6 *Let* S *be a specification. Then* $\widehat{S} \sqsubseteq'_{abs} S(R^-_{\mathcal{L}_S}, \mathcal{L}_S)$ *with*

$$(h, s) \in abs \ \ \textit{iff}\ \ s \in \mathit{final}_S(h)$$

Proof. For $(DR1')$: Let $s \in post_{\widehat{S}}(\langle \iota, op \rangle)$. Because of the tight definition of $op^{\widehat{S}}$ there exists a history $h \in \mathcal{L}_S$ with $last(\mathcal{I}(h)) = \langle \iota, op \rangle$ such that $s \in final_S(h)$. Hence $(h, s) \in abs$ and $h \in post_{S(R^-_{\mathcal{L}_S}, \mathcal{L}_S)}(\langle \iota, op \rangle)$.

If $h \in post_{S(R^-_{\mathcal{L}_S}, \mathcal{L}_S)}(\langle \iota, op \rangle)$, then $h \in \mathcal{L}_S$ and thus there exists $s \in final_S(h)$. Therefore, $s \in post_{\widehat{S}}(\langle \iota, op \rangle)$ and $(h, s) \in abs$.

Let $h' \in post_{S(R^-_{\mathcal{L}_S}, \mathcal{L}_S)}(\langle \iota, op \rangle)$ and $s' \in St$ with $(h', s') \in abs$. Thus, $s' \in final_S(h')$ and we find $h \in \mathcal{L}_S$, $\omega \in Out$ such that $h' = h\langle \langle \iota, op \rangle, \omega \rangle$. Hence, there exists $s \in final_S(h)$ with $(\langle \iota, s \rangle, \langle s', \omega \rangle) \in op^S$. Then, $(\langle \iota, s \rangle, \langle s', \omega \rangle) \in op^{\widehat{S}}$.

For $(DR2')$: Assume $(h, s) \in abs$ and $(\langle \iota, s \rangle, \langle s', \omega \rangle) \in op^{\widehat{S}}$. Therefore, $s \in final_S(h)$ and hence $h\langle \langle \iota, op \rangle, \omega \rangle \in \mathcal{L}_S$. Hence, $s' \in final_S(h\langle \langle \iota, op \rangle, \omega \rangle)$. We conclude that $(h\langle \langle \iota, op \rangle, \omega \rangle, s') \in abs$ and $(\langle \iota, h \rangle, \langle h\langle \langle \iota, op \rangle, \omega \rangle, \omega \rangle) \in op^{S(R^-_{\mathcal{L}_S}, \mathcal{L}_S)}$.

For $(DR3')$: Let $(h', s') \in abs$ and $(\langle \iota, h \rangle, \langle h', \omega \rangle) \in op^{S(R^-_{\mathcal{L}_S}, \mathcal{L}_S)}$. Then, $s' \in final_S(h')$ according to our definition, and there exists $s \in final_S(h)$ such that $(\langle \iota, s \rangle, \langle s', \omega \rangle) \in op^{\widehat{S}}$.

For $(SR2')$: Note that $final_S(\varepsilon) = \textsc{Init}$ and $\textsc{Init}^{S(R^-_{\mathcal{L}_S}, \mathcal{L}_S)} = \{\varepsilon\}$. Hence, $s \in \textsc{Init}$ iff $(\varepsilon, s) \in abs$.

□

Lemma 7 *Let S^A and S^C be specifications with demonic S^C and let $\mathcal{L}_{S^C} \Subset \mathcal{L}_{S^A}$. Then there is a forward refinement $S(R^-_{\mathcal{L}_{S^A}}, \mathcal{L}_{S^A}) \sqsubseteq_{abs} S^C$ with*

$$(r, h) \in abs \;\; \textit{iff} \;\; h \in \mathcal{L}_{S^A} \cap \mathcal{L}_{S^C} \wedge r \in final_{S^C}(h)$$

Proof. This follows from Proposition 11 and Theorem 2 where, as we remarked, $i) \Rightarrow ii)$ holds for not necessarily demonic S^A.

□

Finally, we formulate the completeness Theorem of the combination of forward and backward refinement which is a consequence of Lemmas 5, 6 and 7.

Theorem 5 *Let S^A and S^C be specifications, S^C demonic with $\mathcal{L}_{S^C} \Subset \mathcal{L}_{S^A}$. Then there exist abstraction relations abs_i, $i = 1, 2, 3$ such that*

$$S^A \sqsubseteq_{abs_1} \widehat{S}^A \sqsubseteq'_{abs_2} S(R^-_{\mathcal{L}_{S^A}}, \mathcal{L}_{S^A}) \sqsubseteq_{abs_3} S^C$$

If we restrict ourselves to demonic specifications, this completeness result together with Theorems 1 and 4 shows that forward and backward refinement together form a sound and complete proof method with respect to the ordering $\Subset$ on demonic languages.

Theorem 6 *Let S^A and S^C be demonic specifications. S^A refines to S^C by using forward and backward refinement iff $\mathcal{L}_{S^C} \Subset \mathcal{L}_{S^A}$.*

This theorem shows that there is a refinement proof that $SA1 \sqsubseteq SA2$ using a combination of forward and backward refinement. In fact, with the abstraction relation

$$abs = \{(\bot, i) : i \in \mathbb{Z}\} \cup \{(i, i) : i \in \mathbb{Z}\}$$

we can show that $SA1 \sqsubseteq' SA2$. Note that this abstraction relation is the inverse of the abstraction relation that we used to show $SA2 \sqsubseteq SA1$, and that there is no need to construct any intermediate state machines in this case.

7. Conclusions and related work

In this paper, we have extended the abstraction lattice proposed by Hoffman and Strooper [9] to cover languages that are not necessarily deterministic and complete (Definitions 15 and 16). We have defined a partially ordered set of languages and right congruences that characterises state abstractness on demonic and semi-deterministic specifications and we have shown that the VDM and Z notion of (forward) refinement with abstraction functions from the concrete to the abstract state spaces is sound and complete with respect to this partial order for demonic, semi-deterministic specifications (Theorem 3). Finally, we have defined a notion of backward refinement (Definition 19), similar to the common backward refinement notions, and shown that forward and backward refinement together are sound and complete techniques for refining demonic specifications (Theorem 6).

The refinement relation we used on demonic specifications combines common forward refinement as it is known from Z and VDM [12,17] with our notion of backward refinement, which is a modification of the classical backward refinement [13,10,15]. The combination leads to a refinement notion on the underlying demonic languages that is different from the classical refinement semantics for the state machines generated by Z specifications [8,17] or specifications with predicate transformer semantics [5,17]. In relational semantics [17], operations in Z and VDM are interpreted as total operations, where the states that fulfill the precondition are in their specified relation to the states fulfilling the postcondition and the states that do not satisfy the precondition are related to every possible state in the state space. With total operations every combination is possible and the semantics of the underlying state machine is defined as the set of all behaviours that can be derived by performing the extended operations in sequence. In this semantics, refinement is defined as a selection process on the histories that belong to the same trace, and forward and backward refinement are sound and complete refinement methods in this semantics [8,17].

By providing a different notion of backward refinement (Definition 19) and by limiting ourselves to demonic specifications (Definition 6) we can ensure that traces never disappear during the refinement. Refinement in this sense is a selection process on the histories that belong to the same trace (Definition 10). This is not the case for refinements with forward and backward refinement in the usual sense, where histories are selected, but traces may disappear as a consequence of the refinement. Note that for demonic specifications, S^A refines to S^C with forward and backward refinement in the usual sense [17] does not necessarily imply $\mathcal{L}_{S^C} \Subset \mathcal{L}_{S^A}$: it is possible that the traces in $Tr(\mathcal{L}_{S^A})$ are not a subset of $Tr(\mathcal{L}_{S^C})$, and so traces can disappear during the refinements. In complete analogy to the classical refinement result [8,10,11,17], forward and backward refinement in our context build a sound and complete refinement method with respect to the ordering relation $\Subset$ on the languages (Theorems 5 and 6).

By restricting the set of specifications to demonic and semi-deterministic specifications, forward refinement in its own becomes even a sound and complete method (Theorem 2). This corresponds to similar results in [8,10], for so-called canonical specifications.

In [10], inputs and outputs do not occur explicitly, a finite alphabet of operations with bounded nondeterminism is assumed, and a special symbol $\perp$ is introduced in the state space to simulate divergences. The semantics of state machine refinement is the improved

failures model of CSP [3,7]: a process is represented as a pair (F, D) of failures F and divergences D. Thus, process (F_1, D_1) refines to process (F_2, D_2) iff $F_2 \subseteq F_1$ and $D_2 \subseteq D_1$. This implies that the traces of the concrete process are a subset of the traces defined by the abstract process. The refinement notions on the operation level are down-simulation and up-simulation which are similar to forward and backward simulation, respectively. Down-simulation achieves that the concrete traces are a subset of the abstract ones and hence is stronger than forward simulation in our context. Neither down-simulation nor up-simulation are complete refinement methods on their own, but it is shown that down-simulation together with up-simulation are sound and complete with respect to state machine refinement in the improved failures model.

In [11], the trace model for refinement relies on divergences and not on failures. Again there is no occurrence of input and outputs and a special symbol $\perp$ appears in the state space to simulate divergences. A process is represented as a tuple (T, D) with traces T and divergences D. The process (T_1, D_1) refines to process (T_2, D_2) iff $T_2 \subseteq T_1$ and $D_2 \subseteq D_1$. Downward and upward simulation define the two refinement notions at the operation level. They are slightly different to the respective notions in [10]. This is due to the missing notion of failures. But again the traces of the abstract process are a subset of the traces of the concrete process. One of the main results of this paper is again that downward and upward simulation together form a complete proof method when the divergence model is used for state machine refinement.

Gardiner and Morgan [5] use predicate transformer semantics instead of relational semantics [8]. The predicate transformer for a specification statement can be interpreted as a relation on the state space that relates the states in the precondition with the corresponding states in the postcondition, and that relates all states that do not satisfy the precondition with all possible states (this is called chaotic behaviour) [14,17]. Composition of predicate transformers then means composition of total operations which naturally leads to a demonic specification and the refinement relation on operations with predicate transformer semantics can be interpreted as subset relation on total operations, similar as for the classical forward and backward refinement in relational semantics [8,17]. One main difference with [8] is that Gardiner and Morgan prove that one single data refinement method, which they call cosimulation, is sound and complete when all operations are total. Cosimulation can be interpreted as refinement with an abstraction relation on the power sets of the state spaces instead of an abstraction relation on the state spaces in conventional forward and backward refinement. Our forward and backward refinement notions are stronger and hence less expressive than cosimulation when used on their own, but they lead to a complete refinement method when used in combination.

Acknowledgements

We thank Alena Griffiths and the anonymous referees for their suggestions on earlier versions of the paper. Part of this research was funded by the Australian Research Council Large Grants A49600176 and A49937045.

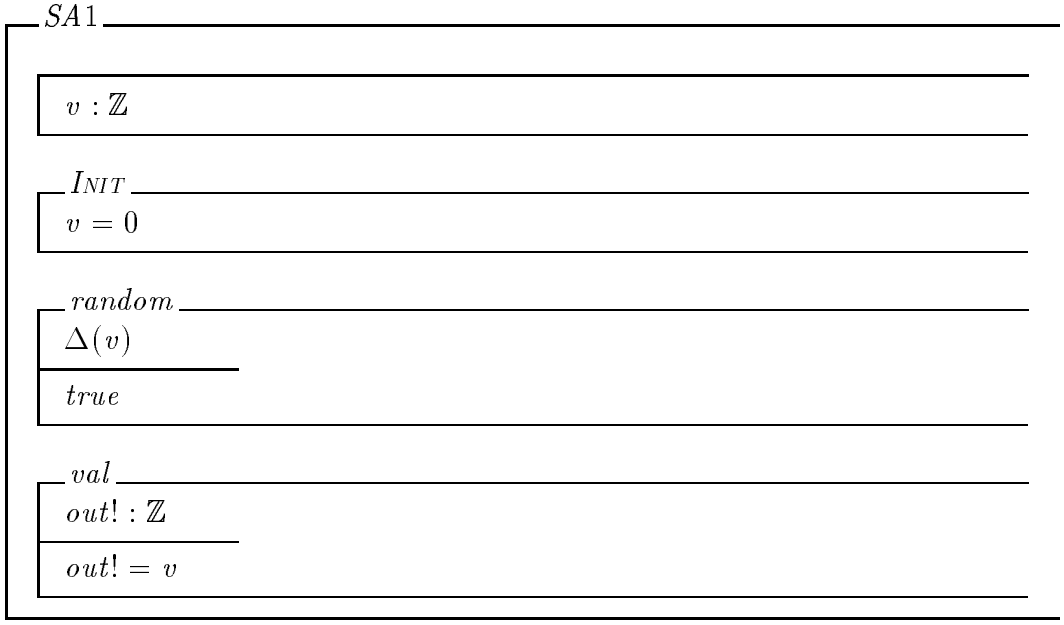
REFERENCES

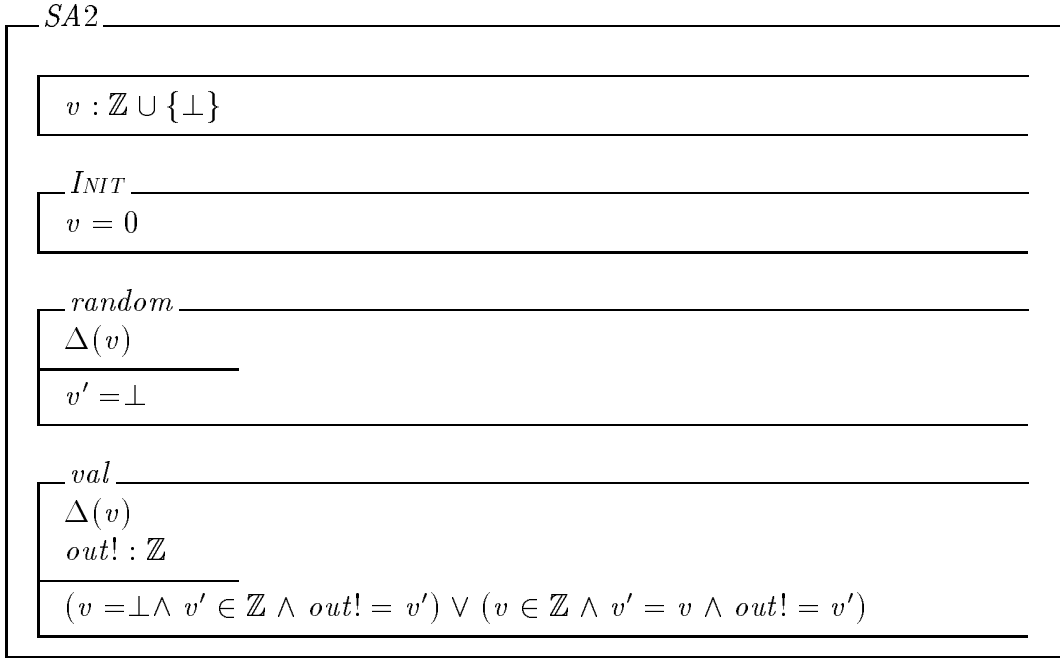
1. M. Abadi and L. Lamport. The existence of refinement mappings. In *Proc. of the 3rd Annual Symposium on Logic in Computer Science*, pages 165–175. IEEE Computer

Society, 1988.
2. R. J. R. Back. On correct refinement of programs. *Journal of Computer and System Sciences*, 23:49–68, 1981.
3. S.D. Brookes and A.W. Roscoe. An improved failures model for communicating processes. *Lecture Notes in Computer Science*, 197, 1985.
4. R. Duke and G. Rose. *Formal Object-Oriented Specification and Design Using Object-Z.* Software Verification Research Centre, The University of Queensland, 1995.
5. P. H. B. Gardiner and C. Morgan. A single complete rule for data refinement. *Formal Aspects of Computing*, 5(4):367–382, 1993.
6. C.A.R. Hoare. Proof of correctness of data representations. *Acta Informatica*, 1(4):271–281, 1972.
7. C.A.R. Hoare. *Communicating sequential processes.* Prentice-Hall, 1985.
8. C.A.R. Hoare, He Jifeng, and J. W. Sanders. Prespecifications in data refinement. *Information Processing Letters*, 25:71–76, 1987.
9. D.M. Hoffman and P.A. Strooper. State abstraction and modular software development. In G.E. Kaiser, editor, *Proc. of the Third ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pages 53–61. ACM Press, 1995.
10. He Jifeng. Process simulation and refinement. *Formal Aspects of Computing*, 1:229–241, 1989.
11. He Jifeng. Various simulations and refinements. In J.W. de Bakker, C., W.P. de Roever, and G. Rozenberg, editors, *Stepwise Refinement of Distributed Systems: Models, Formalisms, Correctness*, volume 430 of *Lecture Notes in Computer Science*, pages 340–360. Springer-Verlag, 1989.
12. C.B. Jones. *Systematic Software Development Using VDM.* Prentice-Hall, second edition, 1990.
13. M. B. Josephs. A state-based approach to communicating processes. *Distributed Computing*, 3:9–18, 1988.
14. Steve King. Z and the refinement calculus. In *Proceedings of VDM'90, Lecture Notes in Computer Science 428*, pages 308–312. Springer-Verlag, 1990.
15. N. Lynch and F. Vaandrager. Forward and backward simulation for timing-based systems. In J.W. de Bakker, C. Huizing, W.P. de Roever, and G. Rozenberg, editors, *Real-Time: Theory in Practice*, volume 600 of *Lecture Notes in Computer Science*, pages 397–445. Springer-Verlag, 1991.
16. D.L. Parnas and P.C. Clements. A rational design process: How and why to fake it. *IEEE Transactions on Software Engineering*, SE-12(2):251–257, February 1986.
17. J. Woodcock and J. Davies. *Using Z: Specification, Refinement , and Proof.* Prentice-Hall, 1996.
18. J.C.P. Woodcock. The rudiments of algorithm refinement. *The Computer Journal*, 35(5):441–450, 1992.

A. Language $\mathcal{L}_A$ and specifications *SA*1 and *SA*2

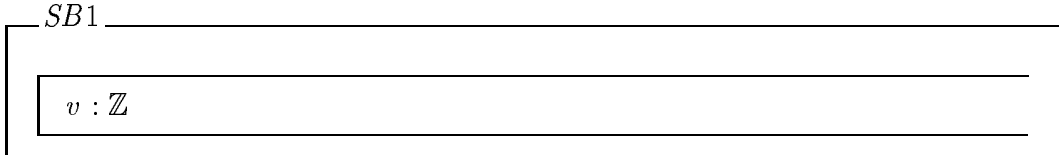
The language $\mathcal{L}_A$ contains two operations: *random* generates a random integer value, and *val* returns the value generated by the last call to *random* as an output. If no call to *random* has been made, *val* returns 0.

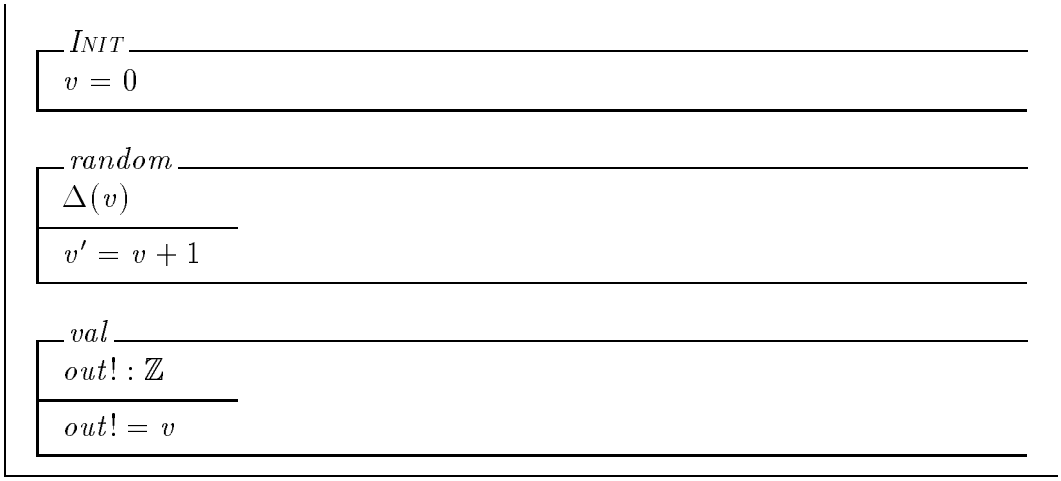




B. Language $\mathcal{L}_B$ and specification $SB1$

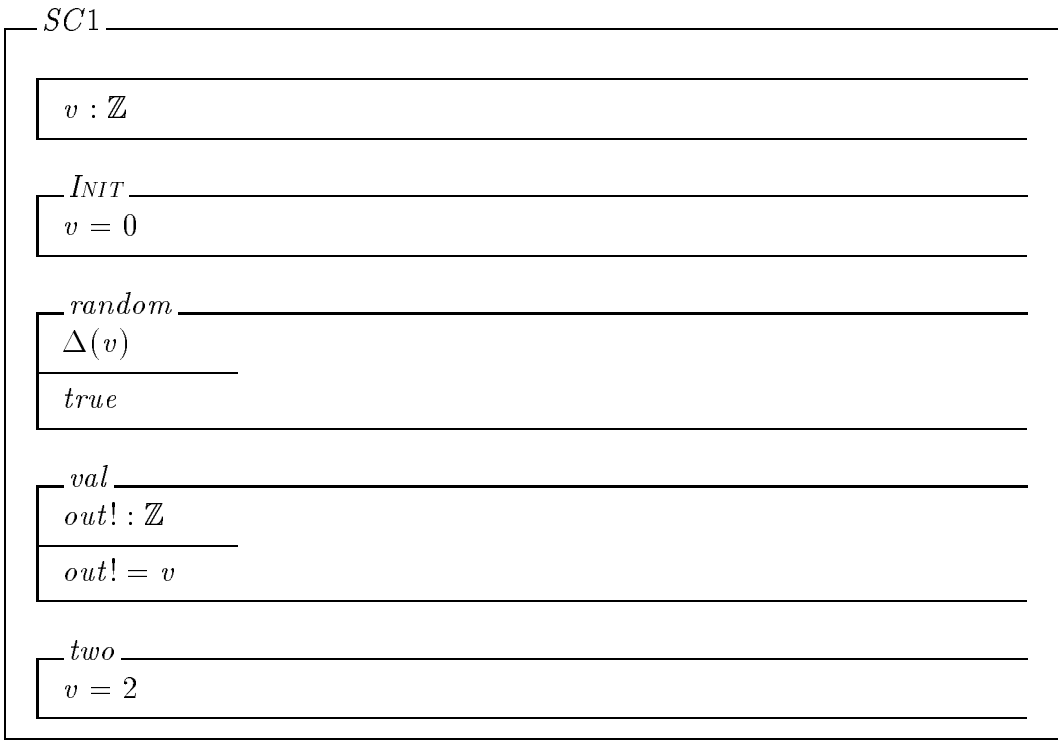
The language $\mathcal{L}_B$ is a subset of $\mathcal{L}_A$. The behaviour of *random* is changed so that it is now deterministic and acts like a counter, incrementing the value of the counter each time *random* is called.





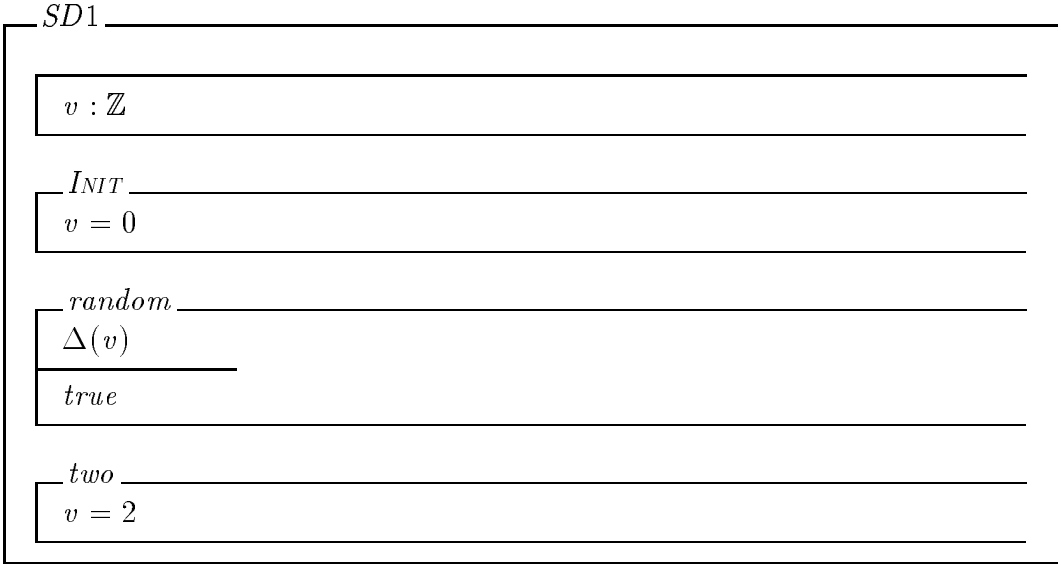
C. Language $\mathcal{L}_C$ and specification $SC1$

The language $\mathcal{L}_C$ is like $\mathcal{L}_A$ except that it has the additional operation *two*, which does not affect the behaviour of the other operations, but which does have a precondition that states that the value generated by the last call to *random* should be 2.



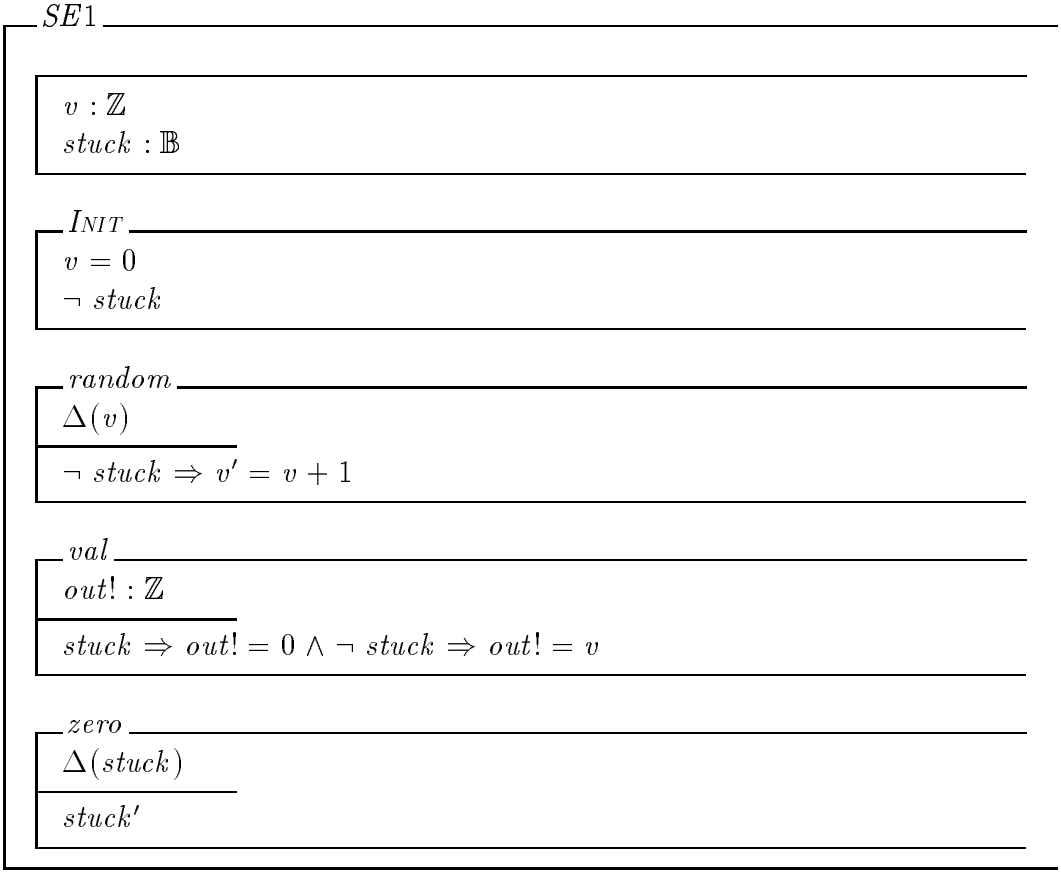
D. Language $\mathcal{L}_D$ and specification $SD1$

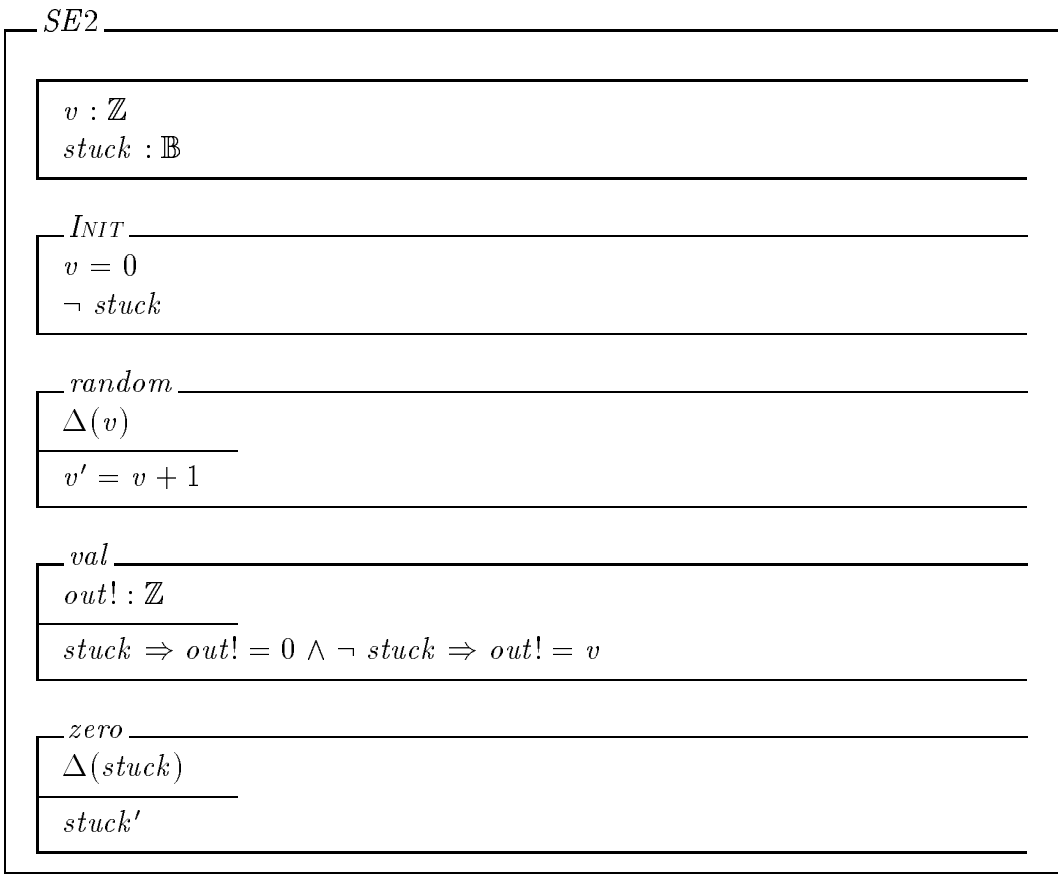
The language $\mathcal{L}_C$ is like $\mathcal{L}_D$ except that the operation *val* has been removed.



E. Language $\mathcal{L}_E$ and specifications $SE1$ and $SE2$

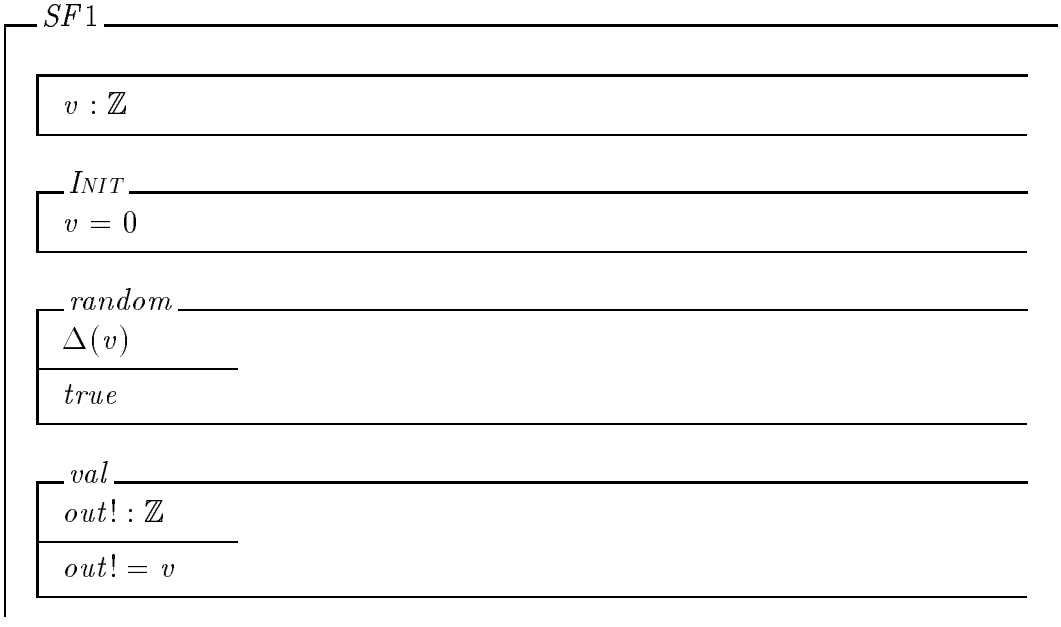
The language $\mathcal{L}_E$ is obtained from $\mathcal{L}_B$ by adding a call *zero* that ensures that the output parameter *out*! of *val* is always 0 after a call to *zero* has been made. When *zero* has not been called yet, *val* returns the number of calls to *random* that have been made.





F. Language $\mathcal{L}_F$ and specification $SF1$

The language $\mathcal{L}_F$ is like $\mathcal{L}_C$ except that instead of testing that the value of v is 2 in *two* it sets the value of v to 2.



two

$\Delta(v)$

$v' = 2$

G. Language $\mathcal{L}_G$ and specifications $SG1$ and $SG2$

The language $\mathcal{L}_G$ is obtained from $\mathcal{L}_A$ by adding a call *zero* that ensures that the output parameter *out*! of *val* is always 0 after a call to *zero* has been made. When *zero* has not been called yet, *val* returns the value generated by the last call to *random* (or 0 if *val* has never been called).

*SG*1

$v : \mathbb{Z}$
$stuck : \mathbb{B}$

Init

$v = 0$
$\neg\ stuck$

random

$\Delta(v)$

$true$

val

$out! : \mathbb{Z}$

$stuck \Rightarrow out! = 0 \land \neg\ stuck \Rightarrow out! = v$

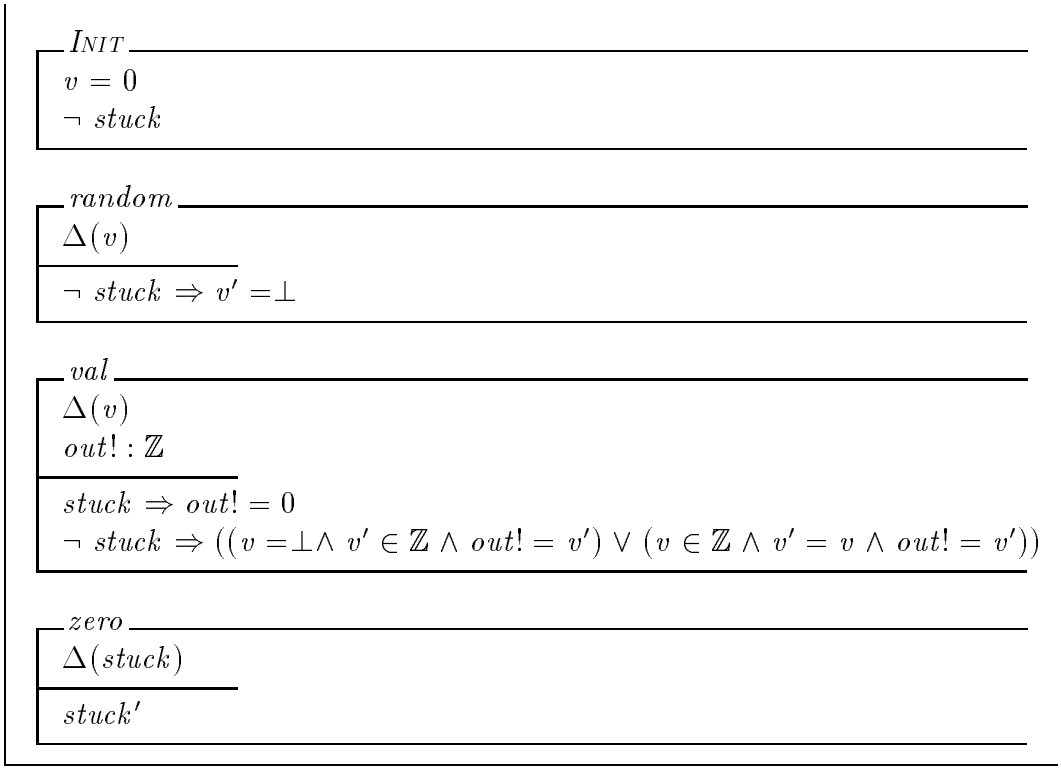
zero

$\Delta(stuck)$

$stuck'$

*SG*2

$v : \mathbb{Z} \cup \{\bot\}$
$stuck : \mathbb{B}$


INIT
v = 0
¬ stuck
random
Δ(v)
¬ stuck ⇒ v′ =⊥
val
Δ(v)
out! : ℤ
stuck ⇒ out! = 0
¬ stuck ⇒ ((v =⊥∧ v′ ∈ ℤ ∧ out! = v′) ∨ (v ∈ ℤ ∧ v′ = v ∧ out! = v′))
zero
Δ(stuck)
stuck′