Experiences with Combining Formalisms in VVSL

C.A. Middelburg
PTT Research, Neher Laboratories
P.O. Box 421, 2260 AK Leidschendam, The Netherlands

Abstract

This paper primarily reports on semantic aspects of how a formal specification of the PCTE interfaces has been achieved in a situation where only a combination of existing formalisms could meet the needs. The motivations for combining a VDM specification language with a language of temporal logic, for translating the resulting language, called VVSL, to an extended COLD-K and for translating it also (partially) to the language of the logic MPL_ω are briefly outlined. The main experiences from this work on combination and transformation of formalisms are presented. Some important experiences with the application of VVSL to the formal specification of the PCTE interfaces and otherwise are also mentioned.

Keywords & Phrases:
formal specification languages, model-oriented specification, pre- and post-conditions, inter-conditions, temporal logic, transformational semantics, logical semantics.

1987 CR Categories:
D.2.1, D.2.2, D.3.1, F.3.1, F.3.2, F.4.1

1 Introduction

A large software system often needs a precise specification of its intended behaviour. A precise specification provides a reference point against which the correctness of the system concerned can be established — either by verifying it a posteriori, or preferably by developing it hand in hand with a correctness proof. A precise specification also makes it easier to reason about the system. Moreover, it is possible to reason about the system before its development is undertaken. This possibility opens up a way to increase the confidence that the system will match the inherently informal, user's requirements. If a change to an existing software system is contemplated, then the consequences of the change have to be taken into account. But without a precise specification, it is often difficult to grasp the consequences of a change.

In order to achieve precision, a specification must be written in a formal specification language. A formal specification language needs a mathemetically precise and complete description of the semantics of the language.

In practice, the creation of a precise specification is sometimes doomed to fail by absence of a formal specification language that meets the needs. In some cases, the problem may be solved by combining several languages. However, it is not sufficient to combine the languages syntactically. In order to achieve a formal specification, they must also be combined semantically. This means that the semantic bases of the languages have to be integrated. This is generally hard, since the bases of many languages are not organized in an orthogonal and elementary way. Besides, they tend to have different mathematical origins. This paper reports about these matters from the experiences with VVSL [Mid89d].

1.1 Background

VVSL is a specification language which combines two other languages both syntactically and semantically. It is the specification language that has been used in the ESPRIT project "VDM for Interfaces of the PCTE" (abbreviated to VIP). This project was concerned with describing in a mathematically precise manner the

PCTE interfaces [PCT86], using a VDM specification language as far as possible. The PCTE interfaces have been defined as a result of the ESPRIT project “A Basis for a Portable Common Tool Environment”. The PCTE interfaces aim to support the coordination and integration of software engineering tools. They address topics such as an object management system, a common user interface and distribution. The objectives in producing a formal specification of the PCTE interfaces can be summarised as follows:

- to support implementors of PCTE, tool builders using PCTE primitives, etc. by giving them access to a precise description of the interfaces;
- to identify weaknesses in the PCTE interfaces and to suggest improvements;
- to provide a basis for long-term evolution of PCTE.

These objectives provided the main reasons for structuring the specification of the interfaces:

- Unstructured, the specification will be too large to have any chance of being reasonably understandable by its intended ‘users’. Division into ‘functional units’ with well-defined interfaces enhances understandability.
- Weaknesses in the current design should be identified and improvements suggested. Composing the functional units from instantiations of a small number of orthogonal and generic ‘underlying semantic units’ supports such improvements.
- PCTE is currently rather language (C) and operating-system (UNIX) specific. Evolution away from these influences will improve PCTE. Isolating the language- and operating-system-oriented parts supports such evolution.

For structuring specifications, VVSL has modularization constructs and parameterization constructs. They are very similar to those of the kernel design language COLD-K [Jon89b]. The modularization mechanism permits two modules to have parts of their state in common, including hidden parts. After appraisal of the current trends in modular structuring with respect to the specification of the PCTE interfaces, the structuring features of COLD-K were in high favour with the VIP project. A short survey of the current trends in modular structuring is given in an appendix.

In VDM specification languages, operations may yield results which depend on a state and may change that state. Operations are always regarded as *atomic*, i.e. not to interact with some environment during execution. Therefore, intermediate states do not contain essential details about the behaviour of an operation. Only the initial state and final state matter. In the case of the PCTE interfaces, not all operations are as isolated as this. For some operations, termination, final state and/or results partly depend on the interference of concurrently executed operations through a partially shared state. In these cases, intermediate states do contain essential details about the behaviour of the operation concerned. Although it may be considered inelegant to have such details externally visible, many aspects of this kind cannot be regarded as being internal in the case of the PCTE interfaces. Adding a rely- and a guarantee-condition (which can be used to express simple safety properties) to the usual pre- and post-condition pair of operations, as proposed in [Jon83], was found to be inadequate for specifying the PCTE operations. At least some of the additional expressive power, that is usually found in languages of temporal logic, was considered necessary.

In VVSL, a language for structured VDM specifications is combined with a language of temporal logic in order to support implicit specification of non-atomic operations. The language of temporal logic has been inspired by various temporal logics based on linear and discrete time [LPZ85, HM87, BK85, Fis87]. The design of VVSL aimed at obtaining a well-defined combination that can be considered a VDM specification language with additional syntactic constructs which are only needed in the presence of non-atomic operations and with an appropriate interpretation of both atomic and non-atomic operations which covers the original VDM interpretation.

VVSL without its modularization and parameterization constructs is referred to as *flat* VVSL. The *structuring sublanguage of* VVSL consists of the modularization and parameterization constructs complementing flat VVSL.

In the VIP project, VVSL has been provided with a well-defined semantics by defining a translation to COLD-K extended with constructs which are required for translation of the VVSL constructs that are only needed in the presence of non-atomic operations. The report [BM88] contains both the definition of this translation and the definition of the COLD-K extensions. In a follow-up project, VVSL has been provided with a well-defined semantics in another way. In [Mid89c], flat VVSL has been given a logical semantics by defining a translation to the language of the logic MPL$_\omega$ [KR89]. In [Mid90], the structuring sublanguage of VVSL has been given a semantics by defining a translation to the terms of a calculus, which is obtained by putting a variant of lambda calculus, called $\lambda\pi$-calculus [Fei89], on top of a specialization of a general model of specification modules, called Description Algebra [Jon89a]. MPL$_\omega$, Description Algebra and $\lambda\pi$-calculus are also used for the formal definition of COLD-K in [FJKR87].

1.2 Structure of the Paper

Sections 2, 3 and 4 deal informally with the combination of a VDM specification language with a language of temporal logic in VVSL. Section 2 presents some features of the VDM specification language; only features that are strongly involved in the combination are treated. Section 3 outlines the motivation for combining the two languages and sketches how this is actually done. Section 4 describes the temporal language in some detail.

Sections 5 and 6 introduce the transformations to the extended COLD-K and the language of MPL$_\omega$. Section 5 outlines the motivation for transforming VVSL to the extended COLD-K and gives as an example the translation to COLD-K for the operation definitions of the VDM specification language that has been incorporated in VVSL. Section 6 outlines the motivation for transforming flat VVSL to the language of MPL$_\omega$ and gives as an example the interpretation in MPL$_\omega$ for the logical expressions of flat VVSL.

Sections 7, 8 and 9 deal with formal aspects of the combination. Section 7 sketches the extensions of COLD-K that are required for transforming full VVSL to a COLD-K-like language. Sections 8 and 9 give as examples the translation to the extended COLD-K for the temporal formulae and the operation definitions of VVSL. For comparison, the interpretation in MPL$_\omega$ is also given.

2 VVSL: the VDM Specification Language

2.1 Connections with other VDM Specification Languages

The major VDM specification languages are presented in [BJ82] (VDM specification language with domain-theoretic semantics) and [Jon86] (VDM specification language with set-theoretic semantics). The latter VDM specification language is closely related to Z [Spi88]. The forthcoming standard VDM specification language BSI/VDM SL [BSI92] unifies the major VDM specification languages. A proposal for the formal semantics of BSI/VDM SL is presented in [Lar92]. In the first version of this proposal, the semantics of the STC VDM Reference Language defined in [Mon85] and the proposal for modularization and parameterization in BSI/VDM SL presented in [Bea88] were taken as the starting point. Inadequacies of the predecessors of this proposal for modularization and parameterization were the main reason to choose something quite different for modularization and parameterization in VVSL. The chosen modularization and parameterization constructs are very similar to those of COLD-K; which is manifest in the translation rules given in [Mid89d]. Meanwhile modularization has been removed from the proposal for the formal semantics of BSI/VDM SL.

The flat VDM specification language that has been incorporated in VVSL is roughly a restricted version of BSI/VDM SL. It is very similar to the language used in [Jon86]. One can define types, functions working on values of these types, state variables which can take values of these types, and operations which may interrogate and modify the state variables. In the remainder of this section, a short introduction to state variables and (atomic) operations is given. For a more complete presentation, see e.g. [Jon86].

2.2 State Variables and Operations

In the VDM specification language that has been incorporated in VVSL, like in other VDM specification languages, operation is a general name for imperative programs and meaningful parts thereof (e.g. procedures). Unlike functions, operations may yield results which depend on a *state* and may change that state. The states concerned have a fixed number of named components, called state variables, attached to them. In all states, a value is associates with each of these state variables. Operations change states by modifying the value of state variables. Each state variable can only take values from a fixed type. State variables correspond to programming variables of imperative programs.

State Variables
A *state variable* is interpreted as a function from states to values, that assigns to each state the value taken by the state variable in that state.

A state variable is declared by a variable definition of the following form:

$$v\colon t.$$

It introduces a name for the state variable and defines the type from which the state variable can take values.
A *state invariant* and an *initial condition*, of the form

$$\mathsf{inv}\, E_{inv} \qquad \text{and} \qquad \mathsf{init}\, E_{init},$$

respectively, can be associated with a collection of variable definitions. The state invariant is a restriction on what values the state variables can take in any state. The initial condition is a restriction on what values the state variables can take initially, i.e. before any modification by operations.

Operations
An *operation* is interpreted as an input/output relation, i.e. a relation between 'initial' states, tuples of argument values, 'final' states and tuples of result values.

An operation is implicitly specified by an operation definition of the following form:

$$\begin{array}{l} op(x_1\colon t_1, \ldots, x_n\colon t_n)\; x_{n+1}\colon t_{n+1}, \ldots, x_m\colon t_m \\ \quad \mathsf{ext\ rd}\; v_1\colon t'_1, \ldots, \mathsf{rd}\; v_k\colon t'_k, \mathsf{wr}\; v_{k+1}\colon t'_{k+1}, \ldots, \mathsf{wr}\; v_l\colon t'_l \\ \quad \mathsf{pre}\; E_{pre} \\ \quad \mathsf{post}\; E_{post}. \end{array}$$

The header introduces a name for the specified operation and defines the types of its arguments and results. The header also introduces names for the argument values and result values to be used within the body. The *external clause* indicates which state variables are of concern to the behaviour of the operation and also indicates which of those state variables may be modified by the operation. The *pre-condition* defines the inputs, i.e. the combinations of initial state and tuples of argument values, for which the operation should terminate, and the *post-condition* defines the possible outputs, i.e. combinations of final state and tuple of result values, from each of these inputs. Operations are potentially non-deterministic: the post-condition may permit more than one output from the same input. The pre-condition may be absent, in which case the operation should terminate for all inputs (i.e. it is equivalent to the pre-condition true). In the post-condition, one refers to the value of a state variable v in the initial state by $\overleftarrow{v}$ and to its value in the final state by v.

An initial state may lead to a final state via some intermediate states. However, one cannot refer to these intermediate states in operation definitions. The underlying idea is that intermediate states do not contain essential details about the behaviour of the operation being defined, since operations are always regarded as being *atomic*, i.e. not to interact with some environment during execution. Atomic operations may certainly be implemented as combinations of sub-operations, provided that the whole remains insensitive to interference.

3 VVSL: Combining VDM and Temporal Logic

3.1 Motivation

Sometimes, operations are not as isolated as this. An important case that occurs in practice is that termination and/or the possible outputs depend on both the input and the interference of concurrently executed operations through state variables. In that case, intermediate states do contain essential details about the behaviour of the operation being defined. Although it is usually considered inelegant to have such details visible, it happens in practice. The PCTE interfaces constitute a striking example. A language of temporal logic seems a useful language for specifying such *non-atomic* operations implicitly.

In VVSL, a formula from a language of temporal logic can be used as a *dynamic constraint* associated with a collection of state variable definitions or as an *inter-condition* associated with an operation definition. With a dynamic constraint, global restrictions can be imposed on the set of possible histories of values taken by the state variables being defined. With an inter-condition, restrictions can be imposed on the set of possible histories of values taken by the state variables during the execution of the operation being defined in an interfering environment.

The temporal language has been inspired by a temporal logic from Lichtenstein, Pnueli and Zuck that includes operators referring to the *past* [LPZ85], a temporal logic from Moszkowski that includes the *chop* operator [HM87], a temporal logic from Barringer and Kuiper that includes *transition* propositions [BK85] and a temporal logic from Fisher with models in which *finite stuttering* can not be recognized [Fis87]. The operators referring to the past, the chop operator and the transition propositions obviate the need to introduce auxiliary state variables acting as *history variables*, *control variables* and *scheduling variables*, respectively. The above-mentioned temporal logics are all based on linear and discrete time. Temporal logics based on linear and discrete time are further explored and better understood with respect to their adequacy for specifying interacting parts of imperative programs than temporal logics based on branching time [EH86] and temporal logics based on real time [BKP86, Sta88]. Therefore temporal logics based on branching or real time have not influenced the temporal language of VVSL directly. For more details on the temporal language, see Section 4. In the remainder of this section, it is sketched how the VDM specification language and the language of temporal logic are combined in VVSL.

3.2 Computations

For atomic operations, it is appropriate to interpret them as input/output relations. This so-called relational interpretation is the usual one for VDM specification languages. For non-atomic operations, such an interpretation is no longer appropriate, since intermediate states contain essential details about the behaviour of the operation; e.g. the possible outputs depend on the input as well as the interference of concurrently executed operations through state variables. Non-atomic operations require an operational interpretation as sets of computations which represent possible histories of values taken by the state variables during execution of the operation concerned in possible interfering environments.

A *computation* of an operation is a non-empty finite or infinite sequence of states and connecting labelled transitions. The transition labels indicate which transitions are effected by the operation itself and which are effected by the environment. The transitions of the former kind are called *internal* steps, those of the latter kind are called *external* steps. In every step some state variables that are relevant for the behaviour of the operation have to change, unless the step is followed by infinitely many steps where such changes do not happen. In other words, 'finite stuttering' is excluded. In the case of an internal step, the state variables which change can only be write variables. In the case of an external step, they can be read variables and write variables. The computation can be seen as generated by the operation and the environment working interleaved but labelled from the viewpoint of the operation.

The introduction of transition labels for distinguishing between internal and external steps is significant. Such a distinction is essential to achieve an *open* semantics of a non-atomic operation, i.e. a semantics which models the behaviour of the operation in all possible environments. The kind of transition labelling, which is presented here, is introduced by Barringer, Kuiper and Pnueli in [BKP84].

The exclusion of finite stuttering corresponds to the view that if nothing actually happens then one can not tell that time has passed, unless nothing happens for an infinitely long time. It makes computations much like computations in 'real time' models based on the view that things happen at a finite rate, viz. the model of the temporal logic of the reals with the 'finite variability' restriction [BKP86] and the model of the temporal logic for 'conceptual state specifications' with the 'local finiteness' restriction [Sta88].

3.3 Inter-conditions

In full VVSL, an operation is implicitly specified by an operation definition of the following form:

$op(x_1 : t_1, \ldots, x_n : t_n)\ x_{n+1} : t_{n+1}, \ldots, x_m : t_m$
  ext rd $v_1 : t'_1, \ldots,$ rd $v_k : t'_k,$ wr $v_{k+1} : t'_{k+1}, \ldots,$ wr $v_l : t'_l$
  pre E_{pre}
  post E_{post}
  inter φ_{inter}.

That is, an inter-condition is added to the usual operation definition. This inter-condition defines the possible computations of the operation.

For atomic operations, only the relational interpretation is relevant. Therefore the relational interpretation of an operation is maintained in VVSL. This interpretation is characterized by the external clause (for atomic operations), the pre-condition and the post-condition. The operation has in addition the operational interpretation, which is mainly characterized by the external clause (for non-atomic operations) and the inter-condition. The inter-condition is a temporal formula which must hold initially for the computations from the operational interpretation. This corresponds to a notion of validity for temporal formulae which is 'anchored' at the initial state of the computation (see [MP89]). The inter-condition can be used to express that the operation is atomic. However, this may also be indicated by leaving out the inter-condition. This means that atomic operations can be implicitly specified as in other VDM specification languages. The possible computations of an atomic operation have at most one transition and their transitions are always internal steps.

The computations from the operational interpretation must agree with the relational interpretation. To be more precise, its finite computations must have a first and last state between which the input/output relation according to the relational interpretation holds and its infinite computations must have a first state which belongs to the domain of this relation. The inter-condition expresses a restriction on the set of computations that agree with the relational interpretation. The requirement on the infinite computations means that the pre-condition does not always define the inputs for which the operation necessarily terminates (in any valid interpretation). For non-atomic operations, the pre-condition defines the inputs for which the operation possibly terminates. In other words, it defines the inputs for which termination may not be ruled out completely by interference.

For non-atomic operations the values taken by a read variable in the initial state and the final state must be allowed to be different, since a read variable may be changed by the environment. This has as a consequence that the external clause does not contribute to the characterization of the relational interpretation of non-atomic operations. It contributes only to the characterization of the operational interpretation. Read variables cannot be changed during an internal step but can be changed during external steps. Write variables can be changed during any step. Only read and write variables are relevant for the behaviour.

With the combined possibilities of the external clause and the inter-condition, non-atomic operations can be defined while maintaining as much of the VDM style of specification as possible.

The pre-condition of a non-atomic operation only defines the inputs for which the operation possibly terminates. This allows that the operation only terminates due to interference of concurrently executed operations. Moreover, the post-condition of a non-atomic operation will be rather weak in general, for inputs must often be related to many outputs which should only occur due to certain interference of concurrently executed operations. The inter-condition is mainly used to describe which interference is required for termination and/or the occurrence of such outputs.

Apart from finite stuttering, the operational interpretation of interfering operations characterized by a rely- and a guarantee-condition, as proposed in [Jon83], can also be characterized by an inter-condition of the following form:

$$\mathsf{inter}\ \Box((\mathsf{is}\text{-}E \ \Rightarrow\ \bigcirc\varphi_{rely}) \wedge (\mathsf{is}\text{-}I \ \Rightarrow\ \bigcirc\varphi_{guar})),$$

where the temporal formulae φ_{rely} and φ_{guar} are the original rely- and guarantee-condition with each occurrences of an expression $\overleftarrow{v}$ replaced by the temporal term $\ominus\, v$. Rely- and guarantee-conditions can only be used to express invariance properties of state changes in steps made by the environment of the operation concerned and invariance properties of state changes in steps made by the operation itself. This is often inadequate; e.g. for operations that should wait until something occurs, such as some PCTE primitives.

3.4 Dynamic Constraints

In full VVSL, a *dynamic constraint*, of the form

$$\mathsf{dyn}\,\varphi_{dyn},$$

can be associated with a collection of variable definitions. A dynamic constraint is a restriction on what histories of values taken by the state variables can occur.

The role of dynamic constraints is similar to that of state invariants. State invariants impose restrictions on what values the state variables can take. Therefore they should be preserved by the relational interpretation of all operations. Dynamic constraints impose restrictions on what histories of values taken by the state variables can occur. Likewise they should be preserved by the operational interpretation of all operations. A dynamic constraint is a temporal formula which must hold always for the computations of any operation.

4 VVSL: the Language of Temporal Logic

In this section a short overview is given of the language of temporal logic that can be used in VVSL. The temporal language is treated in isolation, i.e. the connections with the remainder of VVSL (sketched in Section 3) are reduced as far as possible.

4.1 Temporal Formulae

The syntax of the temporal language is outlined by the following production rules from the complete grammar of VVSL, which is given in Chapter 3 of [BM88]:

$$\begin{aligned}\varphi ::= {} & \mathsf{is}\text{-}I \mid \mathsf{is}\text{-}E \mid \tau \mid \tau_1 = \tau_2 \mid \varphi_1 ; \varphi_2 \mid \bigcirc\varphi \mid \varphi_1 \mathcal{U} \varphi_2 \mid \ominus\varphi \mid \varphi_1 \mathcal{S} \varphi_2 \mid \\ & \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \exists x \in t \cdot \varphi \mid \mathsf{let}\, x\colon t \triangleq \tau \,\mathsf{in}\, \varphi \ ,\\ \tau ::= {} & e \mid \bigcirc\tau \mid \ominus\tau \mid f(\tau_1,\ldots,\tau_n) \ .\end{aligned}$$

In order to be a well-formed temporal formula, a temporal term τ (third alternative of first production rule) must have type $\mathbf{B}$ (which denotes the set of boolean values).

4.2 Computations

Computations are rather loosely described in Section 3. More accuracy is needed for a description of the intended meaning of temporal formulae.

A model of a complete VVSL specification is a structure $\mathcal{A}$ in which, among other things, a special pre-defined name State is associated with a non-empty set $\mathsf{State}^{\mathcal{A}}$ (of states).

A (*labelled*) *computation* w.r.t. $\mathcal{A}$ is a pair $\langle\sigma,\lambda\rangle$ where σ is a non-empty finite or infinite sequence over $\mathsf{State}^{\mathcal{A}}$ and λ is a sequence over the set $\{\mathbf{I},\mathbf{E}\}$ (of *transition labels*) whose length is *1* less than the length

of σ, if σ is finite, and is infinite otherwise.
The transitions labels correspond directly to the two transition propositions is-I (is internal step) and is-E (is external step).

The usual representation of a finite computation $\langle\langle s_0, \ldots, s_n\rangle, \langle l_0, \ldots, l_{n-1}\rangle\rangle$ is

$$s_0 \overset{l_0}{\to} s_1 \to \cdots \to s_{n-1} \overset{l_{n-1}}{\to} s_n,$$

and the usual representation of an infinite computation $\langle\langle s_0, s_1, \ldots\rangle, \langle l_0, l_1, \ldots\rangle\rangle$ is

$$s_0 \overset{l_0}{\to} s_1 \overset{l_1}{\to} \cdots.$$

If $\gamma = s_0 \overset{l_0}{\to} s_1 \to \cdots \to s_{n-1} \overset{l_{n-1}}{\to} s_n$ then the *length* of γ, $|\gamma|$, is defined to be the number of states in γ, i.e. $|\gamma| = n + 1$. If γ is infinite, we write $|\gamma| = \omega$.
Furthermore, the notations $\mathit{pref}(\gamma, i)$ and $\mathit{suff}(\gamma, i)$ are used to denote $s_0 \overset{l_0}{\to} s_1 \to \cdots \to s_{i-1} \overset{l_{i-1}}{\to} s_i$ and $s_i \overset{l_i}{\to} s_{i+1} \to \cdots \to s_{n-1} \overset{l_{n-1}}{\to} s_n$ (in the finite case) or $s_i \overset{l_i}{\to} s_{i+1} \overset{l_{i+1}}{\to} \cdots$ (in the infinite case), respectively.

4.3 Satisfaction of Temporal Formulae

The notation $\langle\gamma, i\rangle \models_g \varphi$ will be used to indicate the truth of temporal formula φ at position i in computation γ under assignment g. By an *assignment* is meant a function which assigns to each value name (i.e. variable in the mathematical sense) a value belonging to the appropriate type.

The meaning of the temporal formulae is now outlined by the inductive rules for the temporal operators ; (chop), $\bigcirc$ (next), $\mathcal{U}$ (until), $\ominus$ (previous) and $\mathcal{S}$ (since) from the definition of satisfaction:

$$\begin{array}{rl}
\langle\gamma, i\rangle \models_g \varphi_1 ; \varphi_2 \text{ iff} & \text{for some } j,\ i \le j < |\gamma|,\ \langle \mathit{pref}(\gamma, j), i\rangle \models_g \varphi_1 \text{ and } \langle \mathit{suff}(\gamma, j), 0\rangle \models_g \varphi_2, \\
& \text{or } |\gamma| = \omega \text{ and } \langle\gamma, i\rangle \models_g \varphi_1, \\
\langle\gamma, i\rangle \models_g \bigcirc\varphi \text{ iff} & i + 1 < |\gamma| \text{ and } \langle\gamma, i+1\rangle \models_g \varphi, \\
\langle\gamma, i\rangle \models_g \varphi_1 \mathcal{U} \varphi_2 \text{ iff} & \text{for some } k,\ i \le k < |\gamma|,\ \langle\gamma, k\rangle \models_g \varphi_2 \text{ and} \\
& \text{for every } j,\ i \le j < k,\ \langle\gamma, j\rangle \models_g \varphi_1, \\
\langle\gamma, i\rangle \models_g \ominus\varphi \text{ iff} & i > 0 \text{ and } \langle\gamma, i-1\rangle \models_g \varphi, \\
\langle\gamma, i\rangle \models_g \varphi_1 \mathcal{S} \varphi_2 \text{ iff} & \text{for some } k,\ 0 \le k \le i,\ \langle\gamma, k\rangle \models_g \varphi_2 \text{ and} \\
& \text{for every } j,\ k < j \le i,\ \langle\gamma, j\rangle \models_g \varphi_1.
\end{array}$$

The rules for the logical connectives and quantifiers are as usual.

The notations $\Diamond\varphi$ (eventually), $\Box\varphi$ (henceforth) and their counterparts for the past are defined as abbreviations:

$$\begin{aligned}
\Diamond\varphi &\triangleq \mathsf{true}\ \mathcal{U}\ \varphi, \\
\Box\varphi &\triangleq \neg(\Diamond\neg\varphi), \\
\diamondminus\varphi &\triangleq \mathsf{true}\ \mathcal{S}\ \varphi, \\
\boxminus\varphi &\triangleq \neg(\diamondminus\neg\varphi).
\end{aligned}$$

5 Transforming VVSL to COLD-K

5.1 Motivation

COLD-K provides modularization and parameterization mechanisms which are adequate for writing large specifications in state-based styles and have firm mathematical foundations. This modularization mechanism permits two modules to have parts of their state in common, including hidden parts. COLD-K is a formal specification language which is meant to be used as the kernel of user-oriented versions of the language (attuned to e.g. different styles of specification or different implementation languages), each being an extension with features of a purely syntactic nature. A VDM specification language that is restricted to first-order functions can be considered to be a user-oriented version of COLD-K.

This means that by giving a translation from the flat VDM specification language that have been incorporated in VVSL to COLD-K, one gets 'for free' suitable features for structuring VDM specifications. Besides, it is obvious that for the most part this translation is relatively easy. In other words, apart from the constructs for the definition of non-atomic operations, to provide VVSL with a well-defined semantics by defining a translation to COLD-K is an attractive approach in case a well-defined semantics must be made available at short notice.

Because of the combination with a language of temporal logic, the situation is more complicated. The additional constructs cannot be translated to COLD-K. COLD-K has to be extended first. It is far from obvious that COLD-K can be extended straightforwardly to a suitable basis for full VVSL, but insoluble problems are not to be expected either. This complication makes the approach less attractive, but it remains a reasonable alternative under the constraints of the VIP project which hardly allow to develop a semantic basis for VVSL.

5.2 Translation rules

The translation from VVSL constructs to COLD-K constructs has been defined by means of schematic production rules, called *translation rules*. Presenting the definition of the translation in this way, emphasizes the syntactic nature of the translation.

The left-hand side of a translation rule is a VVSL construct enclosed by the special brackets $\langle\!|, |\!\rangle$, which may contain variables for subconstructs. The right-hand side is a COLD-K construct, which may contain these variables enclosed by the special brackets $\langle\!|, |\!\rangle$ for subconstructs (except for variables ranging over constructs solely consisting of an *identifier*, which may occur without enclosing brackets). The left-hand side and right-hand side of a translation rule are separated by the arrow $\Rightarrow\!\!\!\rhd$.

The translations of a VVSL construct C are the terminal productions of $\langle\!| C |\!\rangle$. In general, the translation is not unique.

The special brackets $\langle\!|, |\!\rangle$ denote a *translation operator* which maps meaningful VVSL constructs to meaningful COLD-K constructs. The resemblance of the special brackets with the 'semantic brackets' $[\![,]\!]$ is intentional. It is meant to strengthen the intuition of translation operators as meaning functions. In the complete definition of the translation, an auxiliary translation operator is used, which is denoted there by the special brackets $\{\!|, |\!\}$. Thus the translation of the declarative aspects of definitions and the translation of the definitional aspects of definitions could be split.

5.3 Example

An example of a VVSL construct with straightforward translation to COLD-K is the operation definition for atomic operations. The translation is outlined by the following translation rules from the complete definition of the translation from VVSL to the extended COLD-K, which is given in Chapter 3 of [BM88]:

$$
\begin{array}{l}
\langle\!| op(x_1 : t_1, \ldots, x_n : t_n)\ x_{n+1} : t_{n+1}, \ldots, x_m : t_m \\
\quad \mathsf{ext\ rd}\ v_1 : t'_1\ , \ldots, \mathsf{rd}\ v_k : t'_k\ ,\ \mathsf{wr}\ v_{k+1} : t'_{k+1}\ , \ldots, \mathsf{wr}\ v_l : t'_l\ \ \mathsf{pre}\ E_1\ \ \mathsf{post}\ E_2 |\!\rangle \ \ \Rightarrow\!\!\!\rhd \\
\quad\quad \mathsf{proc}\ op : t_1 \times \cdots \times t_n \to t\ \ \mathsf{mod}\ v_{k+1} : \to t'_{k+1}, \ldots, v_l : \to t'_l \\
\quad\quad \mathsf{axiom} \\
\quad\quad\quad \mathsf{forall}\ x_1 : t_1, \ldots, x_n : t_n\ \ (\langle\!| E_1 |\!\rangle = \mathsf{true}\ \Rightarrow\ (\langle\, op(x_1, \ldots, x_n)\,\rangle\ \mathsf{true})) \\
\quad\quad \mathsf{axiom} \\
\quad\quad\quad \mathsf{forall}\ x_1 : t_1, \ldots, x_n : t_n \\
\quad\quad\quad\quad (\langle\!| E_1 |\!\rangle = \mathsf{true}\ \Rightarrow\ ([\,\mathsf{let}\ x_{n+1} : t_{n+1}, \ldots, x_m : t_m;\ x_{n+1}, \ldots, x_m := op(x_1, \ldots, x_n)\,]\ \langle\!| E_2 |\!\rangle = \mathsf{true})).
\end{array}
$$

Especially the COLD-K modification rights construct and its assertion constructs corresponding to the box and diamond operators of dynamic logic [Har84] make this translation straightforward. Acquainted with dynamic logic, the resemblance with the definition of satisfaction for operation definitions in Appendix C of [Jon86] seems clear. However, the box and diamond operators of dynamic logic are not exactly those of COLD-K. This means that it is not trivial to show that this translation captures the intended meaning of operation definitions, although it may be intuitively clear.

6 Transforming VVSL to the Language of MPL$_\omega$

6.1 Motivation

For various VVSL constructs, translation to COLD-K is not straightforward. Amongst the less obvious to translate are the logical expressions. Because their value can be either true, false or *undefined*, the classical meaning of the logical connectives and quantifiers has to be extended. This must be done as in LPF (see [Che86, Jon86]):

$\neg E$	is true	if E is false	$E \vee E'$	is true	if E is true or E' is true
	is false	if E is true		is false	if E is false and E' is false
	is *undefined*	otherwise,		is *undefined*	otherwise,

$\exists x \in t \cdot E$	is true	if for some value c of type t, E is true when x is interpreted as c
	is false	if for each value c of type t, E is false when x is interpreted as c
	is *undefined*	otherwise.

The other logical connectives and quantifiers are expressible by $\neg$, $\vee$ and $\exists$ in the classical way.

The approach to the translation of logical expressions is connected with the treatment of three-valued predicates in classical two-valued logic which is described in [Bli88].

The translation is outlined by the following translation rules from the complete definition of the translation from VVSL to the extended COLD-K, which is given in Chapter 3 of [BM88]:

$\langle\!| \neg E |\!\rangle \Rightarrow$
 some y_1: $\mathbf{B}$
 (forall y_2: $\mathbf{B}$ $((\langle\!| E |\!\rangle = \mathsf{true} \;\Leftrightarrow\; y_2 = \mathsf{false})$ and $(\langle\!| E |\!\rangle = \mathsf{false} \;\Leftrightarrow\; y_2 = \mathsf{true}) \quad \Leftrightarrow\; y_1 = y_2))$,

$\langle\!| E_1 \vee E_2 |\!\rangle \Rightarrow$
 some y_1: $\mathbf{B}$
 (forall y_2: $\mathbf{B}$
 $((\langle\!| E_1 |\!\rangle = \mathsf{true}$ or $\langle\!| E_2 |\!\rangle = \mathsf{true} \;\Leftrightarrow\; y_2 = \mathsf{true})$ and
 $(\langle\!| E_1 |\!\rangle = \mathsf{false}$ and $\langle\!| E_2 |\!\rangle = \mathsf{false} \;\Leftrightarrow\; y_2 = \mathsf{false}) \;\Leftrightarrow\; y_1 = y_2))$,

$\langle\!| \exists\, x \in t \cdot E |\!\rangle \Rightarrow$
 some y_1: $\mathbf{B}$
 (forall y_2: $\mathbf{B}$
 ((exists x: t $(\langle\!| E |\!\rangle = \mathsf{true}) \;\Leftrightarrow\; y_2 = \mathsf{true})$ and
 (forall x: t $(\langle\!| E |\!\rangle = \mathsf{false}) \;\Leftrightarrow\; y_2 = \mathsf{false}) \quad \Leftrightarrow\; y_1 = y_2))$.

In order to express "the unique y of sort T such that assertion A holds" in COLD-K one has to write some y': T (forall y: T $(A \;\Leftrightarrow\; y' = y)$).

With this in mind it is intuitively clear that this translation captures the intended meaning for all cases that should not yield an undefined result. The other cases are not intuitively clear. In order to show (even informally) that the translation captures the intended meaning completely, the translation from VVSL to COLD-K has to be composed with the translation from COLD-K to the language of MPL$_\omega$ or a complete proof system for a 'COLD-K logic' (with COLD-K assertions as formulae) has to be devised. The first alternative results in a direct interpretation in MPL$_\omega$. This means that it provides an interpretation accessible to a larger public. After all, MPL$_\omega$ is well related to classical first-order logic.

For various other VVSL constructs, it is also difficult to show that the translation to COLD-K captures the intended meaning. Further translation to MPL$_\omega$ seems needed in all cases.

6.2 Example

The interpretation of logical expressions in MPL$_\omega$ is context dependent. The notation $[\![E]\!]^C_{\vec{s},y}$ is used to denote the MPL$_\omega$ formula expressing the fact that the evaluation of the logical expression E in a context where we have visible names as given by C and state(s) $\vec{s}$ yields value y.
$\mathbb{B}$ is used as a special sort symbol representing the domain of boolean values, and *tt* and *ff* are used as special constant symbols representing the boolean values.

The interpretation of logical expressions in MPL$_\omega$ is outlined by the following defining equations from the complete definition of the interpretation of flat VVSL in MPL$_\omega$, which is given in [Mid89c]:

$$[\![\neg E]\!]^C_{\vec{s},y} \ := \ \forall y' {:} \mathbb{B}(([\![E]\!]^C_{\vec{s},tt} \leftrightarrow y' = f\!f) \wedge ([\![E]\!]^C_{\vec{s},f\!f} \leftrightarrow y' = tt) \leftrightarrow y' = y),$$

$$[\![E_1 \vee E_2]\!]^C_{\vec{s},y} \ := \ \forall y' {:} \mathbb{B}(([\![E_1]\!]^C_{\vec{s},tt} \vee [\![E_2]\!]^C_{\vec{s},tt} \leftrightarrow y' = tt) \wedge ([\![E_1]\!]^C_{\vec{s},f\!f} \wedge [\![E_2]\!]^C_{\vec{s},f\!f} \leftrightarrow y' = f\!f) \leftrightarrow y' = y),$$

$$[\![\exists\ x \in t\ \cdot\ E]\!]^C_{\vec{s},y} \ := \ \forall y' {:} \mathbb{B}((\exists x' {:}\, t^C([\![E]\!]^{C \cup \{d\}}_{\vec{s},tt}) \leftrightarrow y' = tt) \wedge (\forall x' {:}\, t^C([\![E]\!]^{C \cup \{d\}}_{\vec{s},f\!f}) \leftrightarrow y' = f\!f) \leftrightarrow y' = y).$$

In each of these equations, y' is a fresh variable symbol of MPL$_\omega$. In the last equation, x' is a fresh variable symbol of MPL$_\omega$ corresponding to the value name x (this correspondence is fixed in the 'declaration' d).

Although there is a striking resemblance between the translation to COLD-K and the interpretation in MPL$_\omega$, there is a big difference. It is easy to show that for all cases that should yield an undefined result, the right-hand sides of these equations are logically equivalent to $\forall y' {:} \mathbb{B}(y' \neq y)$, which is in turn equivalent to $\neg(y{\downarrow})$, i.e. y is undefined.

7 COLD-K Extensions

In Section 3, it is sketched how a VDM specification language is combined with a language of temporal logic in VVSL. In order to formalize this, an extended COLD-K as well as the translation of the additional constructs to the extended language have been defined. In this section, the extensions of COLD-K are sketched. In Sections 8 and 9, the translation to the extended COLD-K for the temporal formulae and the operation definitions of VVSL is outlined.

The required COLD-K extensions relate to the mathematical foundations, the language constructs, and their meaning. They are formally defined in Chapter 4 of [BM88]. In this section, only aspects with a close connection to the temporal language of VVSL are briefly outlined. Some familiarity with the mathematical foundations of COLD-K is assumed. They are given in [KR89, Jon89a, Fei89].

7.1 The Mathematical Foundations

Roughly, modules in COLD-K (called classes) correspond to presentations of MPL$_\omega$ theories, called *class descriptions*. These theory presentations are of a special kind, since there are always special standard symbols with associated axioms. There are the special sort symbol State representing the state space, the special function symbol s0 representing the initial state, and special predicate symbols of several kinds representing relations on states. This allows program variables to correspond to functions with an argument of sort State and procedures to correspond to predicates with two arguments of sort State. For the extended COLD-K, we have to generalize from class descriptions. That is, additional special standard symbols are needed.

The following additional symbols are introduced:

1. Comp: a special sort symbol; representing the domain of computations.

2. st_n (for all $n < \omega$): a special function symbol; $\mathsf{st}_n(c)$ represents the $(n + 1)$-th state of computation c.

3. int_n (for all $n < \omega$): a special predicate symbol; $\mathsf{int}_n(c)$ indicates that the $(n + 1)$-th state transition in computation c is an internal transition.

4. ext_n (for all $n < \omega$): a special predicate symbol; $\mathsf{ext}_n(c)$ indicates that the $(n + 1)$-th state transition in computation c is an external transition.

5. CComp: a set of variable symbols, which are called *computation symbols* and represent computations.

6. comp_p (for all $p \in \mathsf{CProc}$): a special predicate symbol; $\mathsf{comp}_p(x_1, \ldots, x_n, c, y_1, \ldots, y_m)$ indicates that the procedure call $p(x_1, \ldots, x_n)$ (executing interleaved with an environment) can generate computation c yielding objects $y_1, \ldots, y_m$.

7.2 The Language Constructs and their Meaning

The additional constructs are mainly assertions and expressions concerning *computations*. For the most part, they have COLD-K assertions and expressions concerning states as counterparts. The production rules for temporal assertions comprise the production rules for COLD-K assertions and production rules for assertions corresponding to the temporal formulae of VVSL. Similarly, the production rules for temporal expressions comprise the production rules for COLD-K expressions and production rules for expressions corresponding to the temporal terms of VVSL.

A temporal assertion or temporal expression has a context-dependent meaning. Like a COLD-K assertion or expression, the meaning in given context is a MPL$_\omega$ formula. The notation $[\![P]\!]^C_{c,k}$ is used to denote the MPL$_\omega$ formula that expresses the fact that the temporal assertion P holds at position k in computation c, in a context where we have visible symbols C. In [BM88] the notation *form*(P, C, c, k) is used instead of $[\![P]\!]^C_{c,k}$. The former notation is in the style of [FJKR87]. However, the latter notation is in conformance to the one used to denote the MPL$_\omega$ formulae corresponding to temporal formulae from the temporal language of VVSL. The notation *close*(φ, C) is used to denote the existential closure of MPL$_\omega$ formula φ with respect to the variable symbols that occur free in φ but are not in C. In [BM88] the notation *cform*(P, C, c, k) is used for *close*$([\![P]\!]^C_{c,k}, C)$. The existential closure of MPL$_\omega$ formulae is used to deal properly with the liberal scope rules for the names introduced by the let-expression of COLD-K.

Furthermore, the notation *prefix*(c, c', k) is used to denote the formula that expresses the fact that computation c' is the prefix of computation c ending at the $(k + 1)$-th state of c, and the notation *suffix*(c, c', k) to denote the formula that expresses the fact that computation c' is the suffix of computation c starting at the $(k + 1)$-th state of c.

The interpretation of temporal assertions in MPL$_\omega$ is outlined by the following defining equations from the complete definition of the interpretation of the temporal assertions and temporal expressions in MPL$_\omega$, which is given in Chapter 4 of [BM88]:

$$\begin{array}{l}
[\![P \ \mathsf{chop}\ Q]\!]^C_{c,k} := \\
\quad \exists c_1 \colon \mathsf{Comp}\ \exists c_2 \colon \mathsf{Comp} \\
\quad (\bigvee_n (\mathit{prefix}(c, c_1, n) \wedge \mathit{suffix}(c, c_2, n)) \wedge \mathit{close}([\![P]\!]^C_{c_1,k}, C) \wedge \mathit{close}([\![Q]\!]^C_{c_2,0}, C)) \vee \\
\quad \bigwedge_n (\mathsf{st}_n(c){\downarrow}) \wedge \mathit{close}([\![P]\!]^C_{c,k}, C),
\end{array}$$

$$[\![\mathsf{next}\ P]\!]^C_{c,k} := \ \mathsf{st}_{k+1}(c){\downarrow} \wedge \mathit{close}([\![P]\!]^C_{c,k+1}, C),$$

$$[\![P \ \mathsf{until}\ Q]\!]^C_{c,k} := \bigvee_n (\mathsf{st}_{k+n}(c){\downarrow} \wedge \mathit{close}([\![Q]\!]^C_{c,k+n}, C) \wedge \bigwedge_{m=0}^{n-1} (\mathit{close}([\![P]\!]^C_{c,k+m}, C))),$$

$$[\![\mathsf{prev}\ P]\!]^C_{c,k} := \begin{array}{ll} \mathit{close}([\![P]\!]^C_{c,k-1}, C) & \text{if } k > 0, \\ \bot & \text{otherwise,} \end{array}$$

$$[\![P \ \mathsf{since}\ Q]\!]^C_{c,k} := \bigvee_{l=0}^{k} (\mathit{close}([\![Q]\!]^C_{c,k-l}, C) \wedge \bigwedge_{m=1}^{l-1} (\mathit{close}([\![P]\!]^C_{c,k-m}, C))).$$

In the first equation, c_1 and c_2 are fresh computation symbols.

It is clear that the interpretation of these temporal assertions in MPL_ω is conformable to the intended meaning of the corresponding temporal formulae of VVSL described in Section 4. This means that the temporal language of VVSL and the temporal assertion language added to COLD-K are very closely connected. Because the extension of COLD-K with a temporal assertion language is only meant to obtain a semantic basis for full VVSL, it makes no sense to devise a rather different temporal assertion language.

Amongst the interesting temporal assertions that do not correspond to temporal formulae of VVSL are the temporal assertions of the form $[X]P$, where X is an expression (statement) and P is a temporal assertion. This makes the temporal assertion sublanguage of the extended COLD-K resembling process logic [HK82].

The other additional constructs are also constructs concerning computations which have original COLD-K constructs as counterparts: an extension of the constrained procedure bodies of COLD-K to non-atomic procedures and an extension of the axioms of COLD-K to computations.

8 Transforming Temporal Formulae

In this section the translation of the temporal formulae of VVSL to the temporal assertions of the extended COLD-K is outlined. For comparison, the direct interpretation in MPL_ω is also sketched.

8.1 Translation to the Extended COLD-K

The translation to the extended COLD-K for the temporal formulae is simple, due to the extension of COLD-K with corresponding temporal assertions.

The intended meaning of temporal formulae as described in Section 4 does not cover undefinedness. Like logical expressions, their value can be either true, false or *undefined*. The intention is actually that the logical connectives and quantifiers distinguish between false and *undefined* as described for logical expressions in Section 6, while the temporal operators identify false and *undefined*. Extending the meaning of the temporal operators in the same way as the classical logical operators would yield very obscure results.

The translation is outlined by the following translation rules from the complete definition of the translation from VVSL to the extended COLD-K, which is given in Chapter 3 of [BM88]:

$\langle\!|\varphi_1 \ ; \ \varphi_2|\!\rangle \ \Rightarrow$
$\quad \mathsf{some}\ y_1 : \mathbf{B}\ (\mathsf{forall}\ y_2 : \mathbf{B}\ (((\langle\!|\varphi_1|\!\rangle = \mathsf{true}\ \ \mathsf{chop}\ \langle\!|\varphi_2|\!\rangle = \mathsf{true}) \ \Leftrightarrow\ y_2 = \mathsf{true}) \ \Leftrightarrow\ y_1 = y_2)),$

$\langle\!|\bigcirc\ \varphi|\!\rangle \ \Rightarrow$
$\quad \mathsf{some}\ y_1 : \mathbf{B}\ (\mathsf{forall}\ y_2 : \mathbf{B}\ (((\mathsf{next}\ \langle\!|\varphi_1|\!\rangle = \mathsf{true}) \ \Leftrightarrow\ y_2 = \mathsf{true}) \ \Leftrightarrow\ y_1 = y_2)),$

$\langle\!|\varphi_1\ \mathcal{U}\ \varphi_2|\!\rangle \ \Rightarrow$
$\quad \mathsf{some}\ y_1 : \mathbf{B}\ (\mathsf{forall}\ y_2 : \mathbf{B}\ (((\langle\!|\varphi_1|\!\rangle = \mathsf{true}\ \ \mathsf{until}\ \langle\!|\varphi_2|\!\rangle = \mathsf{true}) \ \Leftrightarrow\ y_2 = \mathsf{true}) \ \Leftrightarrow\ y_1 = y_2)),$

$\langle\!|\ominus\ \varphi|\!\rangle \ \Rightarrow$
$\quad \mathsf{some}\ y_1 : \mathbf{B}\ (\mathsf{forall}\ y_2 : \mathbf{B}\ (((\mathsf{prev}\ \langle\!|\varphi_1|\!\rangle = \mathsf{true}) \ \Leftrightarrow\ y_2 = \mathsf{true}) \ \Leftrightarrow\ y_1 = y_2)),$

$\langle\!|\varphi_1\ \mathcal{S}\ \varphi_2|\!\rangle \ \Rightarrow$
$\quad \mathsf{some}\ y_1 : \mathbf{B}\ (\mathsf{forall}\ y_2 : \mathbf{B}\ (((\langle\!|\varphi_1|\!\rangle = \mathsf{true}\ \ \mathsf{since}\ \langle\!|\varphi_2|\!\rangle = \mathsf{true}) \ \Leftrightarrow\ y_2 = \mathsf{true}) \ \Leftrightarrow\ y_1 = y_2)).$

Because the temporal language of VVSL and the temporal assertion language added to COLD-K are very closely connected, this translation trivially captures the intended meaning. However it is not explanatory. It may seem that a direct interpretation of temporal formulae in MPL_ω (sketched below) is preferable. However, that interpretation does not fit together with the use of COLD-K as the starting point of a suitable semantic basis for full VVSL. In other words, the indirect interpretation of the temporal language in MPL_ω is needed to be able to do the same for the VDM specification language (which is motivated in Section 5).

8.2 Interpretation in MPL$_\omega$

As indicated above, almost no additional effort is required to obtain a direct interpretation of temporal formulae in MPL$_\omega$.

This is outlined by the following defining equations from the complete definition of the interpretation of flat VVSL in MPL$_\omega$, which is given in [Mid89c]:

$$\begin{array}{l}
[\![\varphi_1 \ ; \ \varphi_2]\!]^C_{c,k,y} \ := \\
\quad \forall y'{:}\,\mathbb{B} \\
\quad\quad (((\exists c_1{:}\,\mathsf{Comp}\ \exists c_2{:}\,\mathsf{Comp} \\
\quad\quad\quad (\bigvee_n (\mathit{prefix}(c, c_1, n) \wedge \mathit{suffix}(c, c_2, n)) \wedge [\![\varphi_1]\!]^C_{c_1,k,tt} \wedge [\![\varphi_2]\!]^C_{c_2,0,tt}) \vee \\
\quad\quad\quad \bigwedge_n (\mathsf{st}_n(c){\downarrow}) \wedge [\![\varphi_1]\!]^C_{c,k,tt}) \leftrightarrow y' = tt) \leftrightarrow y' = y),
\end{array}$$

$$[\![\bigcirc\ \varphi]\!]^C_{c,k,y} \ := \ \forall y'{:}\,\mathbb{B}((\mathsf{st}_{k+1}(c){\downarrow} \wedge [\![\varphi]\!]^C_{c,k+1,tt} \leftrightarrow y' = tt) \leftrightarrow y' = y),$$

$$[\![\varphi_1\ \mathcal{U}\ \varphi_2]\!]^C_{c,k,y} \ := \ \forall y'{:}\,\mathbb{B}((\bigvee_n (\mathsf{st}_{k+n}(c){\downarrow} \wedge [\![\varphi_2]\!]^C_{c,k+n,tt} \wedge \bigwedge_{m=0}^{n-1} ([\![\varphi_1]\!]^C_{c,k+m,tt})) \leftrightarrow y' = tt) \leftrightarrow y' = y),$$

$$[\![\ominus\ \varphi]\!]^C_{c,k,y} \ := \ \begin{array}{ll} \forall y'{:}\,\mathbb{B}(([\![\varphi]\!]^C_{c,k\text{-}1,tt} \leftrightarrow y' = tt) \leftrightarrow y' = y) & \text{if } k > 0 \\ f\!f = y & \text{otherwise,} \end{array}$$

$$[\![\varphi_1\ \mathcal{S}\ \varphi_2]\!]^C_{c,k,y} \ := \ \forall y'{:}\,\mathbb{B}((\bigvee_{l=0}^{k} ([\![\varphi_2]\!]^C_{c,k-l,tt} \wedge \bigwedge_{m=1}^{l-1} ([\![\varphi_1]\!]^C_{c,k-m,tt})) \leftrightarrow y' = tt) \leftrightarrow y' = y).$$

In each of these equations, y' is a fresh variable symbol of MPL$_\omega$. In the first equation, c_1 and c_2 are fresh computations symbols.

Owing to the close connection between the temporal formulae of VVSL and the temporal assertions of the extended COLD-K, there is almost no resemblance between the translation to the extended COLD-K and the interpretation in MPL$_\omega$. Intuitively, the former mainly solves some rather elementary differences between VDM specification languages and COLD-K and the latter mainly assigns (logical) meaning.

9 Transforming Definitions of (Non-atomic) Operations

In this section the translation of the operation definitions for non-atomic operations to the procedure definitions and axioms of the extended COLD-K is outlined. For comparison, the direct interpretation in MPL$_\omega$ is also sketched.

9.1 Translation to the Extended COLD-K

The translation to the extended COLD-K for the operation definitions for non-atomic operations is simple, due to the extensions of COLD-K with corresponding constructs: for expressing modification rights, an extension of the constrained procedure bodies of COLD-K to non-atomic procedures is added, and for characterizing computations, an extension of the axioms of COLD-K to computations is added.

The translation is outlined by the following translation rules from the complete definition of the translation from VVSL to the extended COLD-K, which is give in Chapter 3 of [BM88]:

$$\begin{array}{l}
\langle\!| \mathit{op}(x_1{:}\,t_1, \ldots, x_n{:}\,t_n)\ x_{n+1}{:}\,t_{n+1}, \ldots, x_m{:}\,t_m \\
\quad \mathsf{ext\ rd}\ v_1{:}\,t'_1\ , \ldots, \mathsf{rd}\ v_k{:}\,t'_k\ ,\ \mathsf{wr}\ v_{k+1}{:}\,t'_{k+1}\ , \ldots, \mathsf{wr}\ v_l{:}\,t'_l\ \ \mathsf{pre}\ E_1\ \ \mathsf{post}\ E_2\ \ \mathsf{inter}\ \varphi |\!\rangle \ \Rightarrow \\
\quad\quad \mathsf{proc}\ \mathit{op}{:}\,t_1 \times \cdots \times t_n \to t\ \ \mathsf{mod\ ext}\ v_1{:} \to t'_1\ , \ldots, v_k{:} \to t'_k\ \ \mathsf{int}\ v_{k+1}{:} \to t'_{k+1}, \ldots, v_l{:} \to t'_l \\
\quad\quad \mathsf{axiom} \\
\quad\quad\quad \mathsf{forall}\ x_1{:}\,t_1, \ldots, x_n{:}\,t_n\ \ (\langle\!| E_1 |\!\rangle = \mathsf{true}\ \Rightarrow\ (\langle\ \mathit{op}(x_1, \ldots, x_n)\ \rangle\ \mathsf{true}))
\end{array}$$

$$\begin{array}{l}
\textsf{axiom} \\
\quad \textsf{forall } x_1 \colon t_1, \ldots, x_n \colon t_n \\
\qquad (\{\!|E_1|\!\} = \textsf{true} \;\Rightarrow\; ([\,\textsf{let } x_{n+1} \colon t_{n+1}, \ldots, x_m \colon t_m;\; x_{n+1}, \ldots, x_m := op(x_1, \ldots, x_n)\,]\; \{\!|E_2|\!\} = \textsf{true})) \\
\textsf{caxiom} \\
\quad \textsf{forall } x_1 \colon t_1, \ldots, x_n \colon t_n \;\; (\{\!|E_1|\!\} = \textsf{true} \;\Rightarrow\; (\langle\, op(x_1, \ldots, x_n)\,\rangle\; \textsf{true until not(next true)})) \\
\textsf{caxiom} \\
\quad \textsf{forall } x_1 \colon t_1, \ldots, x_n \colon t_n \\
\qquad (\{\!|E_1|\!\} = \textsf{true} \;\Rightarrow\; ([\,\textsf{let } x_{n+1} \colon t_{n+1}, \ldots, x_m \colon t_m;\; x_{n+1}, \ldots, x_m := op(x_1, \ldots, x_n)\,]\; \{\!|\varphi|\!\} = \textsf{true})).
\end{array}$$

Here shows the lack of integration inherent to the use of COLD-K as the starting point of a semantic basis for full VVSL. As far as modification rights are concerned, definitions of atomic operations and definitions of non-atomic operations must be translated to different kinds of constrained procedure bodies. A smooth generalization of the original kind was not possible. For the same reason, two kinds of axioms are distinguished.

Semantically, this means that the indirect interpretation is actually twofold, while the original one can be derived from the new one. A single direct interpretation in MPL$_\omega$ (sketched below) seems more appropriate.

9.2 Interpretation in MPL$_\omega$

The direct interpretation of operation definitions (for atomic operations and non-atomic operations) in MPL$_\omega$ is much simpler than the indirect one. It consists of a formula corresponding to the external clause and a formula corresponding to each of the conditions from the definition.

The interpretation is outlined by the following defining equations from the complete definition of the interpretation of flat VVSL in MPL$_\omega$, which is given in [Mid89c]:

$$\begin{array}{l}
[\![op(x_1 \colon t_1, \ldots, x_n \colon t_n)\; x_{n+1} \colon t_{n+1}, \ldots, x_m \colon t_m \\
\quad \textsf{ext rd } v_1 \colon t'_1, \ldots, \textsf{rd } v_k \colon t'_k,\; \textsf{wr } v_{k+1} \colon t'_{k+1}, \ldots, \textsf{wr } v_l \colon t'_l \;\; \textsf{pre } E_1 \;\; \textsf{post } E_2 \;\; \textsf{inter } \varphi]\!]^C \;:= \\
\qquad \{\varphi_1, \ldots, \varphi_4\},
\end{array}$$

where:

$$\begin{array}{rcl}
\varphi_1 & = & \forall x'_1 \colon t_1^C, \ldots, x'_n \colon t_n^C, c \colon \textsf{Comp}, x'_{n+1} \colon t_{n+1}^C, \ldots, x'_m \colon t_m^C \\
& & \quad (op^C_{t_1^C \times \cdots \times t_n^C \to t_{n+1}^C \times \cdots \times t_m^C}(x'_1, \ldots, x'_n, c, x'_{n+1}, \ldots, x'_m) \to \\
& & \qquad vmod^C(\{v_1, \ldots, v_k\}, \{v_{k+1}, \ldots, v_l\}, c)), \\
\varphi_2 & = & \forall s \colon \textsf{State}, x'_1 \colon t_1^C, \ldots, x'_n \colon t_n^C \\
& & \quad ([\![E_1]\!]^{C \cup \{d_1, \ldots, d_n\}}_{\langle s \rangle, tt} \to \\
& & \qquad \exists c \colon \textsf{Comp}, x'_{n+1} \colon t_{n+1}^C, \ldots, x'_m \colon t_m^C \\
& & \qquad\; (\textsf{st}_0(c) = s \wedge \neg(\bigwedge_k (\textsf{st}_k(c)\downarrow)) \wedge op^C_{t_1^C \times \cdots \times t_n^C \to t_{n+1}^C \times \cdots \times t_m^C}(x'_1, \ldots, x'_n, c, x'_{n+1}, \ldots, x'_m))), \\
\varphi_3 & = & \forall s \colon \textsf{State}, x'_1 \colon t_1^C, \ldots, x'_n \colon t_n^C \\
& & \quad ([\![E_1]\!]^{C \cup \{d_1, \ldots, d_n\}}_{\langle s \rangle, tt} \to \\
& & \qquad \forall c \colon \textsf{Comp}, x'_{n+1} \colon t_{n+1}^C, \ldots, x'_m \colon t_m^C \\
& & \qquad\; (\textsf{st}_0(c) = s \wedge \neg(\bigwedge_k (\textsf{st}_k(c)\downarrow)) \wedge \\
& & \qquad\;\; op^C_{t_1^C \times \cdots \times t_n^C \to t_{n+1}^C \times \cdots \times t_m^C}(x'_1, \ldots, x'_n, c, x'_{n+1}, \ldots, x'_m) \to \\
& & \qquad\;\;\; \exists t \colon \textsf{State}(\bigvee_k (\textsf{st}_k(c) = t \wedge \neg(\textsf{st}_{k+1}(c)\downarrow)) \wedge [\![E_2]\!]^{C \cup \{d_1, \ldots, d_m\}}_{\langle s, t \rangle, tt}))), \\
\varphi_4 & = & \forall s \colon \textsf{State}, x'_1 \colon t_1^C, \ldots, x'_n \colon t_n^C \\
& & \quad ([\![E_1]\!]^{C \cup \{d_1, \ldots, d_n\}}_{\langle s \rangle, tt} \to \\
& & \qquad \forall c \colon \textsf{Comp}, x'_{n+1} \colon t_{n+1}^C, \ldots, x'_m \colon t_m^C \\
& & \qquad\; (\textsf{st}_0(c) = s \wedge op^C_{t_1^C \times \cdots \times t_n^C \to t_{n+1}^C \times \cdots \times t_m^C}(x'_1, \ldots, x'_n, c, x'_{n+1}, \ldots, x'_m) \to [\![\varphi]\!]^{C \cup \{d_1, \ldots, d_m\}}_{c, 0, tt})).
\end{array}$$

In these equations, x'_i is a fresh variable symbol of MPL$_\omega$ corresponding to the value name x_i (this correspondence is fixed in the declaration d_i). s and t are fresh state symbols (i.e. variable symbols representing states), and c is a fresh computation symbol.
The notation $vmod^C(R, W, c)$ is used to denote the formula that expresses the fact that during each step in

computation c variables from $R \cup W$ are changed and during internal steps variables other than variables from W are not changed.

These formulae reflect the intended meaning clearly. Formulae φ_1, φ_2 and φ_3 generalize the interpretation of external clause, pre-condition and post-condition from pairs of states to computations. Formulae φ_3 and φ_4 are similar, but the former (corresponding to the post-condition) deals only with the first and last state of computations and the latter (corresponding to the inter-condition) deals with computations as a whole.

The direct interpretation shows the integration which is made possible by the use of MPL_ω as the starting point of a semantic basis for full VVSL. Definitions of both atomic and non-atomic operations are translated to formulae of the same shape. Atomic operations are not treated differently from non-atomic ones. They are considered to have the default inter-condition $\bigcirc\,\mathsf{true} \;\Rightarrow\; (\mathsf{is}\text{-}I \wedge \bigcirc\neg\bigcirc\,\mathsf{true})$.

Semantically, this means that the direct interpretation is not twofold. Only the new one is left, but the original can be derived from it by abstracting from the intermediate states.

10 Experiences with the Application of VVSL

In the VIP project, VVSL has been used for the formal specification of the PCTE interfaces. Some experiences with this application of VVSL seem worth mentioning.

The division of the PCTE interfaces into functional units with well-defined interfaces, which could be reached by the use of the modularization and parameterization constructs of VVSL, has made the complexity explicit that is inherent in PCTE. This complexity was kept implicit in [PCT86].

The composition of these functional units from instantiations of a small number of orthogonal and generic underlying semantic units, requires the presence of suitable underlying semantic units. Devising them has revealed what the PCTE way of looking at coordination and integration of software engineering tools exactly is. This way of looking was hided from outsiders in [PCT86].

Experienced specifiers were used to have only a local definition mechanism available as structuring feature. For them, it was rather difficult to master a new style that takes advantage of the available modularization and parameterization mechanisms. Unexperienced specifiers could master such a style much easier.

Although the temporal language of VVSL was designed for use in the VIP project, it is a temporal language of general utility. In working on the formal specification of the PCTE interfaces, several frequently occurring patterns of inter-conditions were recognized. Some 'notational conventions' were added to get the syntax tailored to those patterns.

In a follow-up project, VVSL has also been used to formalize many of the basic concepts of the relational data model, an abstract external interface of a relational database management system and an abstract internal interface for the same system [Mid89b, Mid89a]. These formalizations are meant to provide for examples of the use of VVSL that are accessible to a larger public than the formal specification of the PCTE interfaces.

The idea of composing functional units from instantiations of a small number of orthogonal and generic underlying semantic units is elaborated in these specifications. A firm conclusion is that this approach not only supports adaptability of the specification concerned. It also supports reusability of its parts and it enhances comprehensibility. There is a strong connection between these effects and the goals of modularization techniques which are identified in [FJ90].[1] The related idea of using separation of state-independent aspects and state-dependent ones as a guideline in the division into functional units is also elaborated in these specifications. The impression is that the use of this guideline strengthens the effects mentioned above. For example, the specification might be understood more globally.

Also in working on the formal specification of the internal interface [Mid89a], several frequently occurring patterns of inter-conditions were recognized. These patterns differ considerably from the frequently occurring

[1] In [FJ90], these goals lead the authors to suggest criteria which should govern the choice of modular structure in a specification: comprehensibility of individual modules, suitability of modules for re-use, and intuitive clarity of the modular structure.

patterns which were recognized in on the formal specification of the PCTE interfaces.

11 Conclusions and Final Remarks

The language obtained by combining a language for structured VDM specifications with a language of temporal logic as sketched in this paper, has proved suitable for the formal specification of the PCTE interfaces [VIP88a, VIP88b]. VVSL, in particular the temporal language that can be used in VVSL, has been improved in the course of the work on the formal specification of the PCTE interfaces based on the feedback by the specifiers about their actual needs. This led to various preliminary versions of VVSL. This paper is concerned with the the final version. The preliminary versions of VVSL were also developed by the author. It is worth mentioning that the preliminary version of VVSL described in [Mid87] and the language described under the name EVDM in [Oli88] are the very same.

To provide VVSL with a wel-defined semantics by defining a translation to an extended COLD-K turned out to be a viable approach. A well-defined semantics was available in time. However, it has two important and related disadvantages:

- for various language constructs, it is difficult to show that the formally defined semantics of VVSL corresponds to the intended meaning;
- the formally defined semantics of VVSL is inaccessible to most people for which it was primarily meant (e.g. writers of informal introductions or reference manuals for VVSL, builders of tools for VVSL and specifiers finding ambiguities and incompletenesses in their introduction or reference manual).

In order to overcome these disadvantages, VVSL is also provided with an equivalent semantics by defining an interpretation in roughly the 'nucleus' of COLD-K, which consists of MPL_ω, Description Algebra and $\lambda\pi$-calculus, in a follow-up project. Flat VVSL has now been provided with a well-defined logical semantics by defining an interpretation in the logic MPL_ω. It seems accessible to a much larger public; the only prerequisite is familiarity with classical first order logic. The impression is that the interpretation in MPL_ω is better suited to all people for which a formal definition of VVSL is meant. For the new semantics of the structuring sublanguage, the prerequisites are familiarity with classical lambda calculus and Description Algebra or a similar model of specification modules (e.g. the models presented in [Wir86] or [BHK90]). It is not clear whether this alternative approach would have been usable for the VIP project; in particular it is doubtful whether a well-defined semantics would have been available in time.

The situation faced by the VIP project is in no way unique. In such situations, a relatively general semantic framework for specification languages is clearly missing. The framework should consist of a few orthogonal elements, based on assumptions which are generally met by specification languages, and some rules for composing the semantic basis for a particular specification language from instantiations of these elements. The elements of the general framework must be rather elementary to be usable in such a framework, but preferably not more elementary than strictly necessary. COLD-K is not such a general framework. Orthogonality is present, but it is far from elementary enough. The reason why its use for VVSL was no failure, should not be sought in its generality (illustrated in Section 9). The nucleus of COLD-K may be a first approximation. Orthogonality and genericity is present, but it may be partly too elementary.

Acknowledgements

The author is grateful to the METEOR project for the invitation to present this work at their final workshop. Thanks go to L.M.G. Feijs and H.B.M. Jonkers, both of Philips Research Laboratories Eindhoven, and G.R. Renardel de Lavalette of the University of Utrecht for enthusiastic help on COLD-related matters. Thanks also to J.A. Bergstra of the University of Amsterdam for the suggestion to devise a specification language which combines a language for structured VDM specifications with a language of temporal logic and to support it with a well-defined semantics by translation to COLD-K.

References

[Bea88] S. Bear. Structuring for the VDM specification language. In R. Bloomfield, L. Marshall, and R. Jones, editors, *VDM '88*, pages 2–25. Springer Verlag, LNCS 328, 1988.

[BG80] R.M. Burstall and J.A. Goguen. The semantics of Clear, a specification language. In D. Bjørner, editor, *Abstract Software Specifications*, pages 292–332. Springer Verlag, LNCS 86, 1980.

[BHK90] J.A. Bergstra, J. Heering, and P. Klint. Module algebra. *Journal of the ACM*, 37(2):335–372, 1990.

[BJ82] D. Bjørner and C.B. Jones. *Formal Specification and Software Development.* Prentice-Hall, 1982.

[BK85] H. Barringer and R. Kuiper. Hierarchical development of concurrent systems in a temporal logic framework. In S.D. Brookes, A.W. Roscoe, and G. Winskel, editors, *Seminar on Concurrency*, pages 35–61. Springer Verlag, LNCS 197, 1985.

[BKP84] H. Barringer, R. Kuiper, and A. Pnueli. Now you may compose temporal logic specifications. In *Proceedings of the 16th ACM Symposium on the Theory of Computing*, pages 51–63. Association of Computing Machinery, 1984.

[BKP86] H. Barringer, R. Kuiper, and A. Pnueli. A really abstract concurrent model and its temporal logic. In *Proceedings of the 13th ACM Symposium on the Principles of Programming Languages*, pages 173–183. Association of Computing Machinery, 1986.

[Bli88] A. Blikle. Three-valued predicates for software specification and validation. In R. Bloomfield, L. Marshall, and R. Jones, editors, *VDM '88*, pages 243–266. Springer Verlag, LNCS 328, 1988.

[BM88] J. Bruijning and C.A. Middelburg. VDM extensions: Final report. Report VIP.T.E.4.3, VIP, December 1988.

[BSI92] BSI IST/5/19. VDM specification language – Proto-standard. Doc. N-246, BSI, December 1992.

[Che86] J.H. Cheng. *A Logic for Partial Functions.* PhD thesis, University of Manchester, Department of Computer Science, 1986. Technical Report UMCS-86-7-1.

[EH86] E.A. Emerson and J.Y. Halpern. "Sometimes" and "not never" revisited: On branching versus linear time temporal logic. *Journal of the ACM*, 33(1):151–178, 1986.

[EM85] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification I: Equations and Initial Semantics.* Springer Verlag, EATCS Monograph, 1985.

[Fei89] L.M.G. Feijs. The calculus $\lambda\pi$. In M. Wirsing and J.A. Bergstra, editors, *Algebraic Methods: Theory, Tools and Applications*, pages 307–328. Springer Verlag, LNCS 394, 1989.

[Fis87] M.D. Fisher. Temporal logics for abstract semantics. Technical Report UMCS-87-12-4, University of Manchester, Department of Computer Science, 1987.

[FJ90] J.S. Fitzgerald and C.B. Jones. Modularizing the formal description of a database system. In D. Bjørner, C.A.R. Hoare, and H. Langmaack, editors, *VDM '90*, pages 189–210. Springer Verlag, LNCS 428, 1990.

[FJKR87] L.M.G. Feijs, H.B.M. Jonkers, C.P.J. Koymans, and G.R. Renardel de Lavalette. Formal definition of the design language COLD-K. Technical Report METEOR/t7/PRLE/7, METEOR, 1987.

[GH86] J.V. Guttag and J.J. Horning. Report on the Larch shared language. *Science of Computer Programming*, 6:103–134, 1986.

[Har84] D. Harel. Dynamic logic. In D. Gabbay and F. Guenther, editors, *Handbook of Philosophical Logic, Volume II*, chapter II.10. D. Reidel Publishing Company, 1984.

[HK82] D. Harel and D. Kozen. Process logic: Expressiveness, decidability, completeness. *Journal of Computer and System Sciences*, 25:144–170, 1982.

[HM87] R. Hale and B. Moskowski. Parallel programming in temporal logic. In J.W. de Bakker, A.J. Nijman, and P.C. Treleaven, editors, *Proceedings PARLE, Volume II*, pages 277–296. Springer Verlag, LNCS 259, 1987.

[Jon83] C.B. Jones. Specification and design of (parallel) programs. In R.E.A. Mason, editor, *IFIP '83*, pages 321–332. North-Holland, 1983.

[Jon86] C.B. Jones. *Systematic Software Development Using VDM.* Prentice-Hall, first edition, 1986.

[Jon89a] H.B.M. Jonkers. Description algebra. In M. Wirsing and J.A. Bergstra, editors, *Algebraic Methods: Theory, Tools and Applications*, pages 283–305. Springer Verlag, LNCS 394, 1989.

[Jon89b] H.B.M. Jonkers. An introduction to COLD-K. In M. Wirsing and J.A. Bergstra, editors, *Algebraic Methods: Theory, Tools and Applications*, pages 139–205. Springer Verlag, LNCS 394, 1989.

[KR89] C.P.J. Koymans and G.R. Renardel de Lavalette. The logic MPL_{ω}. In M. Wirsing and J.A. Bergstra, editors, *Algebraic Methods: Theory, Tools and Applications*, pages 247–282. Springer Verlag, LNCS 394, 1989.

[Lar92] P.G. Larsen. The dynamic semantics of the BSI/VDM specification language. Technical report, IFAD, February 1992.

[LPZ85] O. Lichtenstein, A. Pnueli, and L. Zuck. The glory of the past. In R. Parikh, editor, *Proceedings Logics of Programs 1985*, pages 196–218. Springer Verlag, LNCS 193, 1985.

[Mid87] C.A. Middelburg. Syntax and semantics of VVSL. Working Paper VIP.T.D.KM9, VIP, October 1987.

[Mid89a] C.A. Middelburg. Formalization of an abstract interface to a concurrent access handler using VVSL. Report 572 RNL/89, PTT Research Neher Laboratories, July 1989.

[Mid89b] C.A. Middelburg. Formalization of RDM concepts and an abstract RDBMS interface using VVSL. Report 290 RNL/89, PTT Research Neher Laboratories, May 1989.

[Mid89c] C.A. Middelburg. Logical semantics of flat VVSL. Report 954 RNL/89, PTT Research Neher Laboratories, December 1989.

[Mid89d] C.A. Middelburg. VVSL: A language for structured VDM specifications. *Formal Aspects of Computing*, 1(1):115–135, 1989.

[Mid90] C.A. Middelburg. Semantics of VVSL's structuring language. Report 329 RNL/90, PTT Research Neher Laboratories, May 1990.

[Mon85] B.Q. Monahan. A semantic definition of the STC VDM reference language. Technical report, STC IDEC Ltd, 1985.

[MP89] Z. Manna and A. Pnueli. The anchored version of the temporal framework. In J.W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, pages 201–284. Springer Verlag, LNCS 354, 1989.

[Oli88] H.E. Oliver. *Formal Specification Methods for Reusable Software Components.* PhD thesis, University College of Wales, Aberystwyth, 1988.

[PCT86] ESPRIT. *PCTE Functional Specifications*, 4th edition, June 1986.

[Ren89] G.R. Renardel de Lavalette. Modularisation, parameterisation, interpolation. *Journal of Information Processing and Cybernetics EIK*, 25:283–292, 1989.

[San84] D.T. Sannella. A set-theoretic semantics for Clear. *Acta Informatica*, 21:443–472, 1984.

[Spi88] J.M. Spivey. *Understanding Z.* Cambridge University Press, Cambridge Tracts in Theoretical Computer Science 3, 1988.

[ST85] D. Sannella and A. Tarlecki. Building specifications in an arbitrary institution. In G. Kahn, D.B. MacQueen, and G. Plotkin, editors, *Proceedings Symposium on Semantics of Data Types*, pages 337–356. Springer Verlag, LNCS 173, 1985.

[ST88] D. Sannella and A. Tarlecki. Towards formal development of programs from algebraic specifications: Implementations revisited. *Acta Informatica*, 25:233–281, 1988.

[Sta88] E.W. Stark. Proving entailment between conceptual state specifications. *Theoretical Computer Science*, 56:135–154, 1988.

[VIP88a] VIP Project Team. Kernel interface: Final specification. Report VIP.T.E.8.2, VIP, December 1988. Available from PTT Research.

[VIP88b] VIP Project Team. Man machine interface: Final specification. Report VIP.T.E.8.3, VIP, December 1988. Available from PTT Research.

[Win87] J.M. Wing. Writing Larch interface language specifications. *ACM Transactions on Programming Languages and Systems*, 9(1):1–24, 1987.

[Wir86] M. Wirsing. Structured algebraic specifications: A kernel language. *Theoretical Computer Science*, 42(2):123–249, 1986.

Appendix

Short Survey of Current Trends in Modular Structuring

A common assumption in most current theoretical work on modularization and parameterization, e.g. in [ST85], [BHK90], [Ren89], and [Jon89a, Fei89], is that a specification language has building blocks of structured specifications, which correspond to *theory presentations* in the language of an underlying logic. This assumption is trivially met by most existing languages for structured property-oriented specifications like Clear [BG80], the Larch Shared Language [GH86] and ASL [Wir86]. It is also met by existing languages for structured model-oriented (including state-based) specifications like Z [Spi88], the Larch/CLU Interface Language [Win87] and COLD-K [FJKR87, Part II].

Various potential meanings of a specification are considered. The most interesting and important ones are theory presentations, *theories* and *model classes*. Usually, language constructs for building large structured specifications from smaller ones, correspond to operations on theory presentations, theories or model classes. In [Wir86], which contains a definition and an analysis of ASL, is shown that a model based on theory presentations is very useful and that there is a strong connection between this model and the model based on model classes which is regarded as the standard model of ASL. In [FJKR87], the modularization constructs of COLD-K are given a meaning using a model based on theory presentation, called Class Algebra. In [Spi88], the modularization constructs of Z are given a meaning using a model based on model classes. In [BG80], the modularization constructs of Clear are given a meaning using a model based on theories. The Larch Shared Language is an exceptional specification language in this respect. In [GH86], its modularization constructs correspond to purely syntactic manipulations on specification texts.

In the models of modular specifications of Clear and COLD-K, the *origins* of names are taken into account in the treatment of name clashes in the composition of theories and theory presentations respectively. Usually it is assumed that in case of name clashes the same name is intended to denote the same thing.

The primitive operations of models of modular specifications which are proposed in connection with particular work on modularization, generally include operations for *combination*, *hiding* and *renaming*. Hiding and renaming are sometimes combined and called *deriving*. Combination provides for an import mechanism and hiding provides for an export mechanism. Renaming provides for control of name clashes. Roughly speaking, there are no essential differences between the various models after restriction to these operations, except the minor differences which are inherent in the choice to base the model on theory presentations, theories or model classes, and the differences due to the different treatment of name clashes. Major differences, if present, are in additional operations, e.g. operations for restrictions on allowable models that are not expressible in the underlying logic. These remarks do not apply to work in which modularization is tightly coupled to term-generated or initial models by restricting to them in advance, such as in the work presented in [EM85]. If the origin of names is not taken into account in the treatment of name clashes, then the operations for combination, hiding and renaming turn out to be mathematically simple. Otherwise, some complexity seems unavoidable. [BHK90] gives equational axioms of models of modular specifications, called module algebras, which provide operations for combination, hiding and renaming.

Semantically, parameterized specifications are usually viewed as functions on theory presentations, theories or model classes. Syntactically, a version of typed lambda calculus with parameter restriction, like the $\lambda\pi$-calculus introduced in [Fei89], is often used for parameterization of structured specifications and application of parameterized specifications. A lambda calculus based approach to parameterization is also pursued in the theoretical work presented in [ST88] and [Ren89]. Such an approach is also used to provide for a parameterization mechanism in various existing languages for structured specifications including ASL [Wir86] and COLD-K [FJKR87]. In [BG80] a different approach is used for Clear; parameterized specifications are viewed as morphisms in a category (of 'based theories'). However, the definition of application in [San84] (giving a set theoretic semantics for Clear) seems close to a construction that simulates a variant of β-conversion.