ASM Refinement and Generalizations of Forward Simulation in Data Refinement: A Comparison

Gerhard Schellhorn

Institut für Informatik, Universität Augsburg, D-86135 Augsburg, Germany

Abstract

In [1], we have formalized Börger's refinement notion for Abstract State Machines (ASMs). The formalization was based on transition systems, and verification conditions were expressed in Dynamic Logic.

In this paper, the relation between ASM refinement and data refinement is explored. Data refinement expresses operations and verification conditions using relational calculus.

We show how to bridge the gap between the different notations, and that forward simulation in the behavioral approach to data refinement can be viewed as a specific instance of ASM refinement with 1:1 diagrams, where control structure is not refined.

We also prove that two recent generalizations of data refinement, weak refinement and coupled refinement can be derived from ASM refinement.

Key words: abstract state machines, refinement, data refinement

1 Introduction

Refinement is one of the fundamental concepts in the formal development of correct software systems from abstract specifications. Almost every specification language comes with an associated refinement concept.

Email address: `schellhorn@informatik.uni-augsburg.de` (Gerhard Schellhorn).

Based on the refinement concept introduced by Börger ([2], [3]) in the 90'ies into the framework of Gurevich's ASMs [4], a formalization and a correctness proof was given in [1]. A methodology for the use of the ASM refinement paradigm and numerous applications are described in [5].

As with many other refinement notions the refinement concept is based on the use of *commuting diagrams* to structure correctness proofs. The definitions given in [1] originated from formalizing the correctness proofs for the Prolog compiler given in [6], where it was necessary to allow more freedom for the possible shapes of commuting diagrams than previous approaches (e.g. [7], [8] and [9]) had allowed.

A state-based approach and commuting diagrams are also used in two other refinement notions: data refinement [10] and refinement of I/O automata [11]. To be compatible with these refinement notions, the definition of ASM refinement was based on a very simple notion of transition system, i.e. the semantics of ASM rules, instead of the concrete syntax of the rules.

Since one design goal of the framework was effective deduction support, verification conditions were expressed in Dynamic Logic. Dynamic Logic combines the imperative, operational style of specifying transitions as used in ASMs with (first-order or higher-order) predicate logic to specify properties. Since specification and verification using Dynamic Logic is supported by the KIV system [12], machine-supported correctness proofs for the theory could be done, and it could be successfully applied in the already mentioned verification of a Prolog compiler [13],[14].

This paper substantiates the claim stated in [5], that the ASM refinement framework is a suitable framework to study refinement of state-based systems. We show that ASM refinement can be instantiated to the framework of data refinement, as surveyed in [10]. More specifically we will prove that the notion of generalized forward simulation can be instantiated to the notion of forward simulation in the behavioral (also known as the blocking or guards) approach. The proofs require a precise definition how to translate the relational description of data types to equivalent operational ASM rules. Behavioral data refinement then is the instance of ASM refinement that requires a specific form of commuting diagrams (1:1 diagrams), and an additional constraint of conformality. We also show where the contract (also know as non-blocking or precondition) approach differs from ASM refinement.

Recently a number of generalizations have been defined for data refinement. Two important ones are weak refinement [15] which divides operations into external and internal ones (the internal ones being invisible), similar to internal and external actions as used in process algebra, and coupled refinement [16] where one operation can be implemented by a sequence of operations. We will show that both generalizations can be derived as instances of ASM refinement.

This paper is organized as follows: Section 2 gives an introduction to ASMs and Dynamic Logic. ASM refinement is described in Section 3. Four distinct correctness criteria (depending on whether termination must be preserved or not, and whether only finite or also infinite traces are of interest) using Dynamic Logic are defined. These two sections are shortened versions of [1].

Section 4 defines abstract data types and data refinement. Section 5 proves that the behavioral approach to data refinement is equivalent to the special case of ASM refinement with finite input and external control. We also show that forward simulation in data refinement is sufficient to guarantee ASM refinement for specific ASMs with infinite input and internal control. Section 6 analyzes the contract approach, which interprets the domain of an operation as a precondition (and not as a guard, as the behavioral approach does). The contract approach is more liberal than ASM refinement, since divergent runs may be implemented with terminating runs. Nevertheless, operations as used in the contract approach are possible in ASM refinement, and we give verification conditions for this case.

Section 7 shows, how ASM refinement generalizes (behavioral) data refinement, and under which circumstances the requirement of conformality can be dropped.

Sections 8 and 9 analyze how weak refinement and coupled refinement can be derived as instances of data refinement. Finally, Section 10 concludes.

2 Transition Systems and ASMs

The definition of ASM refinement is based on the following simple notion of a transition system:

Definition 1 (Transition system) *A transition system* $\mathsf{M} = (\mathsf{S}, \mathsf{IN}, \mathsf{RULE})$ *consists of a set* S *of states, a subset* $\mathsf{IN} \subseteq \mathsf{S}$ *of initial states and a transition relation* $\mathsf{RULE} \subseteq \mathsf{S} \times \mathsf{S}$

The definition of the semantics of ASMs as a transition system can be found in [4]. It uses algebras as states and transition relation RULE is defined using sets of updates. If extensions of ASM rules that may diverge are considered, an element $\perp$ is assumed to be in the set of states: e.g. for the recursive rules given in [17], state $\perp$ will be the reached if and only if no update set can be computed from RULE. Using the logic of [18], this is equivalent to $\neg\ \mathsf{def}(\mathsf{RULE})$. It is required that $\perp \notin \mathsf{IN}$ (divergence cannot be an initial state) and that $(\bot, \mathsf{s}) \notin \mathsf{RULE}$ holds for all states s (RULE cannot be applied after divergence). We will often be lax in distinguishing a transition system from an ASM, especially when we define rules operationally.

To reason over a transition system M elementary definitions from set and relation theory are used: for a set S, S^* and S^∞ denote the finite and infinite sequences over S, S^ω is the union of both. $\sigma \in \mathsf{S}^\omega$ is often called a stream over S, and we have the obvious definitions of $\mathsf{length}(\sigma) \in \mathbb{N} \cup \{\infty\}$ and $\sigma(\mathsf{n})$ for $\mathsf{n} \leq \mathsf{length}(\sigma)$.

For a relation $\mathsf{R} \subseteq \mathsf{S}_1 \times \mathsf{S}_2$, $\mathsf{dom}(\mathsf{R}) \subseteq \mathsf{S}_1$ is its domain, $\mathsf{ran}(\mathsf{R}) \subseteq \mathsf{S}_2$ is its range, and R^{-1} is the inverse relation that exchanges the two. For relations $\mathsf{R}_1 \subseteq \mathsf{S}_1 \times \mathsf{S}_2$ and $\mathsf{R}_2 \subseteq \mathsf{S}_2 \times \mathsf{S}_3$, $\mathsf{R}_1 \mathbin{⨾} \mathsf{R}_2$ is their composition. R^n is the n-fold composition of $\mathsf{R} \subseteq \mathsf{S} \times \mathsf{S}$, where $\mathsf{R}^0 = \mathsf{id}$ is the identity relation. R^* and R^+ are the union of all R^n for $\mathsf{n} \geq 0$ and $\mathsf{n} > 0$.

Definition 2 (Final states, traces, runs and I/O behavior)

- *A state* s *of* M *is* final, *if* RULE *is not applicable, i.e. if* $\mathsf{s} \notin \mathsf{dom}(\mathsf{RULE})$*. The set of final states is denoted as* OUT *(output states).*
- *A trace (or execution)* $\sigma \in \mathsf{S}^\omega$ *(written:* $\mathsf{trace}(\sigma)$*) is a finite or infinite sequence of states, such that* $\mathsf{RULE}(\sigma(\mathsf{i}), \sigma(\mathsf{i}+1))$ *holds for* $\mathsf{i} < \mathsf{length}(\sigma)$*. For a finite trace, the last state must satisfy* $\sigma(\mathsf{length}(\sigma)) \in \mathsf{OUT}$*.*
- *A* run *of* M *is a trace* σ *that starts with an initial state* $\sigma(0) \in \mathsf{IN}$*.*
- *The* partial *input/output behavior* $\mathsf{PIO}(\mathsf{M})$ *of* M *is the set of pairs* $(\mathsf{s}, \mathsf{s}_0) \in (\mathsf{IN} \setminus \{\bot\}) \times (\mathsf{OUT} \setminus \{\bot\})$*, such that* $\mathsf{RULE}^*(\mathsf{s}, \mathsf{s}_0)$*.*
- *The* total *input/output behavior* $\mathsf{TIO}(\mathsf{M}) \subseteq (\mathsf{IN} \setminus \{\bot\}) \times \mathsf{OUT})$ *extends the partial one by adding pairs* $(\mathsf{s}, \bot)$*, when* s *is an initial state with an infinite run*[1] *.*

To reason over executions of transition systems the temporal operators $\mathsf{AF}(\mathsf{s}, \mathsf{p})$ ("for all executions starting with s predicate p will eventually hold") and $\mathsf{EF}(\mathsf{s}, \mathsf{p})$ ("for some execution starting with s eventually p") are defined. Operators $\mathsf{AF}^+(\mathsf{s}, \mathsf{p})$ and $\mathsf{EF}^+(\mathsf{s}, \mathsf{p})$ are used if the number of steps must be positive. Formally:

$$\begin{aligned}
\mathsf{AF}(\mathsf{s},\mathsf{p}) :&\leftrightarrow \forall\, \sigma.\ \sigma(0) = \mathsf{s} \wedge \mathsf{trace}(\sigma) \rightarrow \exists\, \mathsf{n}.\ \mathsf{p}(\sigma(\mathsf{n}))\\
\mathsf{EF}(\mathsf{s},\mathsf{p}) :&\leftrightarrow \exists\, \sigma.\ \sigma(0) = \mathsf{s} \wedge \mathsf{trace}(\sigma) \wedge \exists\, \mathsf{n}.\ \mathsf{p}(\sigma(\mathsf{n}))\\
\mathsf{AF}^+(\mathsf{s},\mathsf{p}) :&\leftrightarrow \mathsf{s} \notin \mathsf{OUT} \wedge (\forall\, \mathsf{s}_0.\ \mathsf{RULE}(\mathsf{s},\mathsf{s}_0) \rightarrow \mathsf{AF}(\mathsf{s}_0,\mathsf{p}))\\
\mathsf{EF}^+(\mathsf{s},\mathsf{p}) :&\leftrightarrow \mathsf{s} \notin \mathsf{OUT} \wedge (\exists\, \mathsf{s}_0.\ \mathsf{RULE}(\mathsf{s},\mathsf{s}_0) \wedge \mathsf{EF}(\mathsf{s}_0,\mathsf{p}))
\end{aligned}$$

$\mathsf{EF}(\mathsf{s}, \mathsf{p})$ is equivalent to $\exists\, \mathsf{s}'.\ \mathsf{RULE}^*(\mathsf{s}, \mathsf{s}') \wedge \mathsf{p}(\mathsf{s}')$, while AF has no such definition in terms of the transitive closure of RULE.

[1] Note that $\bot$ is already in S, if partial rules are considered. Then nontermination may occur due to diverging rules as well as due to infinite runs.

3 ASM Refinement

In this section we summarize the results of [1] on ASM refinement. The definition refines an "abstract" transition system $\mathsf{AM} = (\mathsf{AS}, \mathsf{AIN}, \mathsf{ARULE})$ to a "concrete" transition system $\mathsf{CM} = (\mathsf{CS}, \mathsf{CIN}, \mathsf{CRULE})$. States that belong to the abstract and concrete system will be denoted as as and cs respectively. Other terminology will be prefixed or subscripted with "A" and "C" as necessary, to distinguish between notions of the abstract and concrete system. E.g. σ_{A} and AOUT will denote traces and final states of the abstract system.

There are four notions of refinement correctness. The first two notions consider only the input/output behavior of the transition systems. They use two relations $\mathsf{IR} \subseteq \mathsf{AIN} \times \mathsf{CIN}$ and $\mathsf{OR} \subseteq \mathsf{AOUT} \times \mathsf{COUT}$ to determine when initial and final states are considered to be equivalent.

Definition 3 (Preservation of partial correctness) *A refinement of* AM *to* CM *preserves partial correctness with respect to* $(\mathsf{IR}, \mathsf{OR})$*, if for every* finite *run* σ_{C} *of* CM *there exists a finite run* σ_{A} *such that the initial states are related by* IR *and the final states are related by* OR*.*

Informally, the definition says, that runs of CM do not yield other results than runs of AM. Terminating refined runs simulate terminating abstract runs via the input/output correspondence. For $\perp \notin \mathsf{S}$ preservation of partial correctness can be specified as $\mathsf{PIO(CM)} \subseteq \mathsf{IR}^{-1} \mathbin{⨾} \mathsf{PIO(AM)} \mathbin{⨾} \mathsf{OR}$ using partial I/O behavior. If $\mathsf{AS} = \mathsf{CS}$ and $\mathsf{IR} = \mathsf{OR} = \mathsf{id}_{\mathsf{AS}}$, then a refinement preserves partial correctness assertions (hence the name).

Preservation of partial correctness is a weak form of refinement, since finite runs may be implemented by diverging ones. Even if the refinement is complete, i.e. the reverse refinement is correct too, it is still possible to implement a system with one finite run by a system that has the same run, and in addition an infinite run from the same initial state. A stronger notion is:

Definition 4 (Preservation of total correctness) *A refinement from* AM *to* CM *preserves total correctness if it preserves partial correctness, and if for any infinite run* σ_{C} *there exists an infinite run* σ_{A} *of* AM *such that the initial states are related by* IR*.*

The stronger definition implies that terminating *as well as non-terminating* refined runs simulate an abstract run via the input/output correspondence. The inclusion $\mathsf{TIO(CM)} \subseteq \mathsf{IR}^{-1} \mathbin{⨾} \mathsf{TIO(AM)} \mathbin{⨾} (\mathsf{OR} \cup \{(\perp, \perp)\})$ holds, if preservation of total correctness can be proved. Again, if both IR and OR are identity, all total correctness assertions for the programs are preserved.

Both definitions are still too weak to study refinement of reactive systems, which should process the same inputs and give the same outputs on infinite runs. Therefore:

Definition 5 (Partial and total preservation of traces) *Given a relation* $\mathsf{IO} \subseteq \mathsf{AS} \times \mathsf{CS}$ *a refinement from* AM *to* CM *totally preserves traces with respect to* IO *if it preserves total correctness for* $\mathsf{IR} := \mathsf{IO} \cap (\mathsf{AIN} \times \mathsf{CIN})$ *and* $\mathsf{OR} := \mathsf{IO} \cap (\mathsf{AOUT} \times \mathsf{COUT})$*, and if for any infinite run* σ_{C} *there is an infinite run* σ_{A} *and two strictly monotone sequences* $\mathsf{i}_0 < \mathsf{i}_1 < \dots$ *and* $\mathsf{j}_0 < \mathsf{j}_1 < \dots$ *of natural numbers, such that for every* k $\mathsf{IO}(\sigma_{\mathsf{A}}(\mathsf{i}_{\mathsf{k}}), \sigma_{\mathsf{C}}(\mathsf{j}_{\mathsf{k}}))$ *holds. For a refinement to partially preserve traces, the refinement must preserve partial correctness and the sequence* $\mathsf{i}_0 \leq \mathsf{i}_1 \leq \dots$ *is only required to be monotone.*

Preservation of traces adds the requirement that an *infinite* refined run must simulate the corresponding abstract one by passing through infinitely many corresponding states. The correspondence is given abstractly by a relation IO. Typically IO will compare intermediate input and output of the ASMs, which is usually modeled by reading or writing external functions. We have refrained from giving a more specific definition, to allow maximal freedom in instantiating the relation IO. Possible concrete instances are: equality of the traces of variables (as in TLA [19]), equality of traces of actions as in refinement of IO automata (as in [11]), IO refinement using an element-wise comparison of inputs and outputs (relations $\mathsf{IT}*$ and $\mathsf{OT}*$ in [20]). Finally, nonatomic refinement of outputs is possible choosing a suitable relation IO: if e.g. the concrete data type represents a natural number from the abstract data type as a sequence of bytes, then an operation that outputs the natural number may be refined to a sequence of operations where each operation outputs one byte.

While most refinements in ASM refinement guarantee total preservation of traces, it is meaningful too to ignore termination and to consider only the input and output that happens during the run of an ASM. Then a run that immediately stops and therefore has no input and output is considered indistinguishable from a run that diverges without any input or output. Partial preservation of traces formalizes this view, which is also taken in refinement of I/O automata [11].

To prove refinement correctness we use commuting diagrams like the ones shown in Figure 1 and a *coupling invariant* INV between abstract and concrete states. In data refinement this relation is usually called a simulation. If we want to preserve traces with respect to IO, then INV must be stronger than IO: if e.g. IO says that both ASMs do the same output, INV may be a conjunct of IO and other properties necessary to prove invariance.

Unlike in data refinement, where diagrams have to match one rule application (1:1 diagram), ASM refinement diagrams may have any shape ("m:n

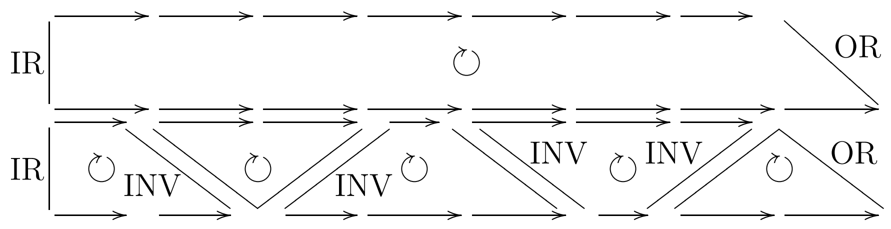

Fig. 1. Verification of a refinement using commuting diagrams

diagrams", where m abstract transitions match n concrete transitions). Even triangular shapes are allowed, but the case of an infinite consecutive sequence of 0:n diagrams must be prevented using some wellfounded predicate $<_{0n}$ on pairs of states. Similarly, infinite sequences of m:0 diagrams must also be ruled out using $<_{m0}$, if total correctness or total preservation of traces is to be achieved.

The verification condition propagates the invariant forwards through the traces (so it is a forward or downward simulation). Since it does not preserve INV in every step, we call INV a "generalized forward simulation". The dual notion, generalized backwards simulation can be defined (some results are in [1]) but is even weaker in our setting than it is in refinement of I/O automata (where it is already weaker than forward simulation) and we will not investigate it in this paper. The proof condition for a commuting diagram with INV to be a generalized forward simulation is as follows:

$$\begin{array}{ll} & \mathsf{INV}(\mathsf{as},\mathsf{cs}) \wedge \neg\,(\mathsf{as} \in \mathsf{AOUT} \wedge \mathsf{cs} \in \mathsf{COUT}) \\ \rightarrow & \quad \mathsf{EF}^+(\mathsf{as}, \lambda\,\mathsf{cs}'.\mathsf{INV}(\mathsf{as},\mathsf{cs}) \wedge (\mathsf{as},\mathsf{cs}') <_{\mathsf{m0}} (\mathsf{as},\mathsf{cs})) \\ & \vee\ \mathsf{AF}^+(\mathsf{cs}, \lambda\,\mathsf{cs}'.\quad \mathsf{EF}^+(\mathsf{as}, \lambda\,\mathsf{as}'.\mathsf{INV}(\mathsf{as}',\mathsf{cs}'))) \\ & \qquad\qquad\qquad\qquad \vee\ \mathsf{INV}(\mathsf{as},\mathsf{cs}') \wedge (\mathsf{as},\mathsf{cs}') <_{\mathsf{0n}} (\mathsf{as},\mathsf{cs})) \end{array} \qquad \text{(VC)}$$

Intuitively, the proof condition says: if states as and cs are related by INV and not both final, then it must be possible to add a commuting diagram, such that INV holds at the end: either this diagram may consist of abstract steps only to form a triangular m:0 diagram (first disjunct, $<_{m0}$ must decrease), or (second disjunct) it must finally be possible to complete a diagram for whatever concrete steps are chosen (the size of the diagram may depend on the choices). The number of abstract steps needed to complete the diagram may be positive, resulting in a m:n diagram where both $\mathsf{m},\mathsf{n} > 0$, or it may be zero and $<_{0n}$ must decrease. Given this condition it can be shown:

Theorem 1 (Generalized forward simulation) *A refinement from* AM *to* CM *preserves total correctness if*

- $\forall\ \mathsf{cs} \in \mathsf{CIN}.\ \exists\ \mathsf{as} \in \mathsf{AIN}.\ \mathsf{IR}(\mathsf{as},\mathsf{cs})$ *("initialize")*
- $\mathsf{IR}(\mathsf{as},\mathsf{cs}) \rightarrow \mathsf{AF}(\mathsf{cs}, \lambda\,\mathsf{cs}'.\mathsf{EF}(\mathsf{as}, \lambda\,\mathsf{as}'.\mathsf{INV}(\mathsf{as}',\mathsf{cs}')))$ *("establish invariant")*
- *verification condition (VC) holds ("keep invariant")*
- $\mathsf{as} \in \mathsf{AOUT} \wedge \mathsf{cs} \in \mathsf{COUT} \wedge \mathsf{INV}(\mathsf{as},\mathsf{cs}) \rightarrow \mathsf{OR}(\mathsf{as},\mathsf{cs})$ *("finalize")*

$$\begin{tikzcd}
\mathsf{as} \arrow[r, dotted, "\vdash \mathsf{ARULE}^+"] \arrow[d, dash, "\mathsf{INV}"'] & \vdash \mathsf{as}' \\
\mathsf{cs} \arrow[ur, dotted, dash, "\vdash \mathsf{INV}"'] &
\end{tikzcd}$$

$\vdash (\mathsf{as}', \mathsf{cs}) <_{\mathsf{0n}} (\mathsf{as}, \mathsf{cs})$

(A)

$$\begin{tikzcd}
\mathsf{as} \arrow[r, dotted, "\vdash \mathsf{ARULE}^+"] \arrow[d, dash, "\mathsf{INV}"'] & \vdash \mathsf{as}' \arrow[d, dotted, dash, "\vdash \mathsf{INV}"] \\
\mathsf{cs} \arrow[r, "\mathsf{CRULE}^+"'] & \mathsf{cs}'
\end{tikzcd}$$

(B)

$$\begin{tikzcd}
\mathsf{as} \arrow[r, dotted, "\vdash \mathsf{ARULE}"] \arrow[d, dash, "\mathsf{INV}"'] & \vdash \bot \arrow[d, dotted, dash, "\vdash \mathsf{INV}"] \\
\mathsf{cs} \arrow[r, "\mathsf{CRULE}"'] & \bot
\end{tikzcd}$$

(C)

$$\begin{tikzcd}
\mathsf{as} \arrow[d, dash, "\mathsf{INV}"'] \arrow[dr, dotted, dash, "\vdash \mathsf{INV}"] & \\
\mathsf{cs} \arrow[r, dotted, "\mathsf{CRULE}^+"'] & \mathsf{cs}'
\end{tikzcd}$$

$\vdash (\mathsf{as}, \mathsf{cs}') <_{\mathsf{m0}} (\mathsf{as}, \mathsf{cs})$

(D)

Fig. 2. Commuting diagrams in ASM refinement

It totally preserves traces, if additionally INV *implies* IO. *For preservation of partial correctness and partial preservation of traces, the condition that* $<_{\mathsf{0n}}$ *decreases for 0:n diagrams may be dropped in (VC).*

The "initialize" condition guarantees that every initial state cs has a corresponding initial state as with $\mathsf{IR}(\mathsf{as}, \mathsf{cs})$. For two such states "establish invariant" guarantees the existence of an initial commuting diagram that establishes INV (for every run starting from cs). In applications this condition can usually be strengthened to $\mathsf{IR} \subseteq \mathsf{INV}$ since INV is usually true for for as and cs already. The "finalize" condition establishes OR for final states.

The proof of the theorem in [1] intuitively follows the construction of commuting diagrams as shown in Figure 1.

For formal verification we express refinement correctness and the proof obligations for ASM rules in Dynamic Logic, which combines rules which are given operationally as abstract programs with predicate logic formulas. Dynamic Logic defines the operators $[\alpha]\ \varphi$, $\langle\alpha\rangle\ \varphi$, and $\langle\!|\alpha|\!\rangle\ \varphi$, where α is an abstract program and φ is a formula of Dynamic Logic again (so boxes and diamonds can be nested). The meaning of the three formulas is "any terminating run of α ends in a state where φ holds", "there is a terminating run of α which ends in a state where φ holds", "all runs of α terminate and end in a state where φ holds". When φ is a first-order formula $\langle\!|\alpha|\!\rangle\ \varphi$ ($[\alpha]\ \varphi$) is just another way to denote the weakest (liberal) precondition of α with respect to φ.

For the most common case of ASM rules that do not diverge (which formally means that $\bot$ is neither in CS nor in AS), expressing the verification conditions in Dynamic Logic is easy, since in this case:

$$\mathsf{EF}(\mathsf{st},\mathsf{p}) \leftrightarrow \langle \textbf{while} \neg\, \mathsf{p}(\mathsf{st}) \wedge \mathsf{st} \notin \mathsf{AOUT}\ \textbf{do}\ \mathsf{TRULE}\rangle\ \mathsf{p}(\mathsf{st})$$
$$\mathsf{AF}(\mathsf{st},\mathsf{p}) \leftrightarrow \langle\!| \textbf{while} \neg\, \mathsf{p}(\mathsf{st}) \wedge \mathsf{st} \notin \mathsf{AOUT}\ \textbf{do}\ \mathsf{TRULE} |\!\rangle\ \mathsf{p}(\mathsf{st})$$
$$\text{where } \mathsf{TRULE} := \textbf{choose}\ \mathsf{s}'\ \textbf{with}\ \mathsf{RULE}(\mathsf{st},\mathsf{s}')\ \textbf{in}\ \mathsf{st} := \mathsf{s}'$$

In the formula, st is a program variable, i.e. a 0-ary dynamic function. Note, that the nondeterministic choice done by the **choose** construct is just an artifact of converting the semantics RULE (a relation) of an ASM rule back to a program of Dynamic Logic. A syntactically given ASM rule just instantiates TRULE.

If rules may diverge, states as, cs etc. mentioned in (VC) may be $\perp$. To get verification conditions in Dynamic Logic which do not directly mention $\perp$, it is necessary to say how relations and predicates on the state (which are defined as formulas) behave with respect to $\perp$. For the relations IO, IR, OR and INV the extension which says that $\perp$ is related to $\perp$, but to no other state, is chosen, and $\perp \in \mathsf{OUT}$ is assumed. There is a weaker alternative here, following the contract approach of data refinement (see Section 4). We will look at this alternative in Section 6.

Using the embedding a general condition for commuting m:n diagrams can be derived. Its formal definition in Dynamic Logic is given in [1]. Figure 2 gives a pictorial description, which shows the four types of allowed diagrams. States and relations after a "$\vdash$" symbol, as well as dotted lines must be shown to exist, assuming the rest of the diagram is given. The diagrams do not show that the choice between the diagrams may depend on the next steps of CM, and that all except the first step of CM in the diagrams (B) and (D) must be steps that can not diverge.

4 Abstract Data Types and Data Refinement

In this section we will give the basic definitions of abstract data types and data refinement. For more on the history of data refinement, see the books [10] and [20]. We will use the notation of the latter book.

Definition 6 (Abstract data type)
An abstract data type $\mathsf{DT} = (\mathsf{G}, \mathsf{S}, \mathsf{INIT}, \{\mathsf{OP}_i\}_{i\in I}, \mathsf{FIN})$ *consists of*

- *a set of global states* G*,*
- *a set of (local) states* S
- *a total relation* $\mathsf{INIT} \subseteq \mathsf{G} \times \mathsf{S}$
- *relations ("operations")* $\mathsf{OP}_i \subseteq \mathsf{S} \times \mathsf{S}$ *where* i *is from some index set* I
- *a total relation* $\mathsf{FIN} \subseteq \mathsf{S} \times \mathsf{G}$

Relations INIT, $\mathsf{OP_i}$ and FIN, are assumed to be the semantics of operations (ultimately implemented by programs) just like we assumed the transition relation of an ASM to be the semantics of an ASM rule. Initialization and finalization are assumed to be total. While we have used ASM rules to specify operations, the operations of a data type are defined using formulas like $\mathsf{x} > 0 \wedge \mathsf{x}' = \mathsf{x} + \mathsf{y}$ that describe a relation on the variables. Unprimed variables denote the state before, primed variables the state after execution of the operation (e.g. in schemas of the Z specification language [21]). Such a formula can be viewed as a specification with pre- and postcondition: the part of the formula using unprimed variables that identifies the domain of the relation (here: the pairs (x, y) such that $\mathsf{x} > 0$) is the precondition, while the rest of the formula that restricts the values of primed variables is the postcondition. A technical difference between ASM rules and operations of a data type is the nature of the state: ASM rules use a first-order algebra as state, while data types use vector of variables. Obviously, a vector of variables can be represented as an algebra of program variables (i.e. 0-ary functions that can be modified). The reverse direction is possible too, by encoding modifiable functions as sets of argument-value pairs ("mappings" in Z).

To use a data type in a program (assumed to have global state G) one has to call the initialization operation INIT first. This will yield an initial local state in S which can subsequently be modified by the operations $\mathsf{OP_i}$ (and no other operations). Operation FIN is provided to finally extract a result to the global program state (the local state is destroyed with this operation). This means that the global state of a program will change according to the relation $\mathsf{INIT} \mathbin{;} \mathsf{OP_{is}} \mathbin{;} \mathsf{FIN}$, where $\mathsf{OP_{is}} := \mathsf{OP_{i_1}} \mathbin{;} \mathsf{OP_{i_2}} \mathbin{;} \ldots \mathbin{;} \mathsf{OP_{i_n}}$ for $\mathsf{is} = [\mathsf{i_1}, \ldots \mathsf{i_n}] \in \mathsf{I}^*$.

Refinement of an abstract data type ADT to a concrete data type CDT is possible only when the two data types are conformal: they must have the same global state G (indicating that they are used by the same program) and the same index set I. Refinement replaces all calls of abstract operations by calls of the corresponding concrete ones.

Definition 7 (Refinement of data types)
A (concrete) data type $\mathsf{CDT} = (\mathsf{G}, \mathsf{CS}, \mathsf{CINIT}, \{\mathsf{COP_i}\}_{\mathsf{i} \in \mathsf{I}}, \mathsf{CFIN})$ *refines an (abstract) data type* $\mathsf{ADT} = (\mathsf{G}, \mathsf{AS}, \mathsf{AINIT}, \{\mathsf{AOP_i}\}_{\mathsf{i} \in \mathsf{I}}, \mathsf{AFIN})$ *if for every finite sequence* $\mathsf{is} \in \mathsf{I}^*$

$$\mathsf{CINIT} \mathbin{;} \mathsf{COP_{is}} \mathbin{;} \mathsf{CFIN} \subseteq \mathsf{AINIT} \mathbin{;} \mathsf{AOP_{is}} \mathbin{;} \mathsf{AFIN} \tag{1}$$

It is not immediately obvious that the correctness condition as given implies that abstract operations can be replaced by concrete ones in any program (some information on this topic is given in [10]). In fact, ASM refinement can be viewed as the refinement of a program that uses the operations, as we will see in the next section. To verify data refinement, commuting diagrams as

$$\begin{tikzcd}
{} \arrow[r, "\mathsf{AINIT}"] \arrow[dr, "\mathsf{CINIT}"'] & {} \arrow[r, "\mathsf{AOP}_{i_1}"] \arrow[d, dash, "\mathsf{R}"'] & {} \arrow[d, dash, "\mathsf{R}"] & \cdots & {} \arrow[r, "\mathsf{AOP}_{i_n}"] \arrow[d, dash, "\mathsf{R}"'] & {} \arrow[r, "\mathsf{AFIN}"] \arrow[d, dash, "\mathsf{R}"] & {} \\
 & {} \arrow[r, "\mathsf{COP}_{i_1}"'] & {} & \cdots & {} \arrow[r, "\mathsf{COP}_{i_n}"'] & {} \arrow[ur, "\mathsf{CFIN}"'] &
\end{tikzcd}$$

Fig. 3. Verification of data refinement using commuting diagrams

shown in Figure 3 are used.

That the diagrams commute, can be shown using either forward or backward simulation. We will only consider forward simulation here. Assuming all relations $\mathsf{OP_i}$ and FIN are total, the following theorem can be proved:

Theorem 2 (Forward simulation for total operations [22])
If a relation $\mathsf{R} \subseteq \mathsf{AS} \times \mathsf{CS}$ *can be found, such that*

- $\mathsf{CINIT} \subseteq \mathsf{AINIT} \mathbin{⨾} \mathsf{R}$ *("initialization")*
- $\forall\, \mathsf{i} \in \mathsf{I}.\ \mathsf{R} \mathbin{⨾} \mathsf{COP_i} \subseteq \mathsf{AOP_i} \mathbin{⨾} \mathsf{R}$ *("correctness")*
- $\mathsf{R} \mathbin{⨾} \mathsf{CFIN} \subseteq \mathsf{AFIN}$ *("finalization")*

can be proved, then CDT *is a correct refinement of* ADT

For partial relations $\mathsf{OP_i} \subseteq \mathsf{S} \times \mathsf{S}$, two approaches are possible, which embed the relation into a total relation over $\mathsf{S}_\perp := \mathsf{S} \cup \{\perp\}$. The behavioral (also called the blocked or guards) approach assumes that if an input is in $\overline{\mathsf{dom}(\mathsf{OP_i})} := \mathsf{S}_\perp \setminus \mathsf{dom}(\mathsf{OP_i})$, then $\widehat{\mathsf{OP_i}}$ has $\perp$ as successor state:

$$\widehat{\mathsf{OP_i}} := \mathsf{OP_i} \cup (\overline{\mathsf{dom}(\mathsf{OP_i})} \times \{\perp\})$$

$\widehat{\mathsf{INIT}}$ and $\widehat{\mathsf{FIN}}$ are defined in the same way (G is also embedded into $\mathsf{G} \cup \{\perp\}$). $\widehat{\mathsf{R}}$ is defined to be $\mathsf{R} \cup \{(\perp, \perp)\}$. Since INIT is total, $\widehat{\mathsf{INIT}} = \mathsf{INIT} \cup \{(\perp, \perp)\}$.

The embedding can be interpreted in two ways: either $\perp$ represents divergence or it represents "$\mathsf{OP_i}$ is not applicable". The latter seems to be the preferred interpretation when setting up a correspondence to process algebra (for more information on this topic and possible interpretations of $\perp$ see [23]). Both interpretations use the same embedding, so they have the same correctness conditions. The two interpretations will lead to two different ASMs as described in Section 5.

By applying Theorem 2 and removing $\perp$ from the resulting conditions the embedding allows to prove:

Theorem 3 (Forward simulation in the behavioral approach [24])
If it can be proved that

- $\mathsf{CINIT} \subseteq \mathsf{AINIT} \mathbin{;} \mathsf{R}$ *("initialization")*
- $\forall\ \mathsf{i} \in \mathsf{I}.\ \mathsf{ran}(\mathsf{dom}(\mathsf{AOP_i}) \lhd \mathsf{R}) \subseteq \mathsf{dom}(\mathsf{COP_i})$ *("applicability")*
- $\forall\ \mathsf{i} \in \mathsf{I}.\ \mathsf{R} \mathbin{;} \mathsf{COP_i} \subseteq \mathsf{AOP_i} \mathbin{;} \mathsf{R}$ *("correctness")*
- $\mathsf{R} \mathbin{;} \mathsf{CFIN} \subseteq \mathsf{AFIN}$ *("finalization")*

then CDT *is a correct refinement of* ADT*.*

In the applicability condition the notation $\mathsf{M} \lhd \mathsf{R} := \mathsf{R} \cap (\mathsf{M} \times \mathsf{ran}(\mathsf{R}))$ denotes the restriction of R to the domain M.

The contract (also called the non-blocked or precondition) approach, assumes that for $\mathsf{s} \notin \mathsf{dom}(\mathsf{OP_i})$ the operation $\mathsf{OP_i}$ may diverge or choose a random next state (chaotic behavior). The definition for this case is

$$\widetilde{\mathsf{OP_i}} := \mathsf{OP_i} \cup (\overline{\mathsf{dom}(\mathsf{OP_i})} \times \mathsf{S}_\perp)$$

$\widetilde{\mathsf{INIT}}$ and $\widetilde{\mathsf{FIN}}$ are defined in the same way. The contract approach uses a different embedding of R than the one that is used in ASM refinement and in the behavioral approach: it defines $\widetilde{\mathsf{R}} := \mathsf{R} \cup (\{\perp\} \times \mathsf{S}_\perp)$. With this embedding it can be proved that:

Theorem 4 (Forward simulation in the contract approach [25])
CDT *is a correct refinement of* ADT *in the contract approach, if the same conditions as for the behavioral approach can be proved, except that the "correctness" condition is replaced by the weaker condition:*

$$\forall\ \mathsf{i} \in \mathsf{I}.\ (\mathsf{dom}(\mathsf{AOP_i}) \lhd \mathsf{R}) \mathbin{;} \mathsf{COP_i} \subseteq \mathsf{AOP_i} \mathbin{;} \mathsf{R} \quad \textit{("correctness")}$$

5 Behavioral Data Refinement and ASM refinement

At first glance, being able to use operations in an arbitrary context seems to be rather different from considering a concrete transition system, such as the one an ASM typically provides. Of course it is possible to view the full transition relation ARULE^* of an ASM as the only operation of an abstract data type (refinement replaces the whole ASM with CRULE^*), but this does not really give much insight.

It makes more sense to view an abstract data type as an instance of an ASM, where the operations $\mathsf{OP_i}$ correspond to ASM rules that modify the current state st (a dynamic function of arity 0). Three problems have to be solved to translate a data type to an ASM. First, a suitable ASM rule has to be defined that executes operations. Second, conformality of operations has to be enforced, and third, the difference in initialization and finalization has to be considered.

A solution for the first problem is easy: for total rules the corresponding ASM rule is simply

$$\mathsf{TRULE_i} := \textbf{ choose } \mathsf{s'} \in \mathsf{S} \textbf{ with } \mathsf{OP_i}(\mathsf{st}, \mathsf{s'}) \textbf{ in } \mathsf{st} := \mathsf{s'}$$

For the behavioral approach an operation $\mathsf{OP_i}$ is translated to

$$\mathsf{BRULE_i} := \textbf{ if } \mathsf{st} \in \mathsf{dom}(\mathsf{OP_i}) \textbf{ then } \mathsf{TRULE_i} \textbf{ else abort}$$

where **abort** is the always diverging program. The semantics of rules $\mathsf{TRULE_i}$ and $\mathsf{BRULE_i}$ coincides with the relations $\mathsf{OP_i}$ and $\widehat{\mathsf{OP}}_\mathsf{i}$.

A way to solve the second problem, is to enforce conformality by *completely removing* control over the selection of rules from the ASM, and to shift it to the environment. The resulting ASM just reacts to the environment, which provides a finite sequence of inputs $\mathsf{is} \in \mathsf{I}^*$ stored in a program variable il in the initial state of the ASM. For the behavioral approach there are then two possibilities to define an ASM, depending on the interpretation of $\perp$.

If $\perp$ is interpreted as "the operation cannot be applied outside of its domain" then the ASM must stop in this case:

```
BERULE(DT) := if $\mathsf{il} \neq [] \wedge \mathsf{st} \in \mathsf{dom}(\mathsf{OP_i})$
              then $\mathsf{BRULE_{head(il)}}$
                   $\mathsf{il} := \mathsf{tail}(\mathsf{il})$
```

(it does not matter whether $\mathsf{BRULE_{head(il)}}$ or $\mathsf{TRULE_{head(il)}}$ is used here). If, on the other hand, $\perp$ is interpreted as "if the operation is applied outside of its domain, it diverges" we get:

```
DERULE(DT) := if $\mathsf{il} \neq []$
              then $\mathsf{BRULE_{head(il)}}$
                   $\mathsf{il} := \mathsf{tail}(\mathsf{il})$
```

where head and tail are the usual operations on sequences.

Using these rules, ASMs can be defined as follows (the "B" and "D" stand for the blocking and diverging variant of the behavioral approach, the "E" for external control, the superscript "*" indicates the finite number of inputs; other variants will be defined later on).

$$\mathsf{BEM}^*(\mathsf{DT}) = (\mathsf{S} \times \mathsf{I}^*, \mathsf{ran}(\mathsf{INIT}) \times \mathsf{I}^*, \mathsf{BERULE}(\mathsf{DT}))$$
$$\mathsf{DEM}^*(\mathsf{DT}) = (\mathsf{S} \times \mathsf{I}^*, \mathsf{ran}(\mathsf{INIT}) \times \mathsf{I}^*, \mathsf{DERULE}(\mathsf{DT}))$$

Both ASMs start with initial states in $\mathsf{ran}(\mathsf{INIT})$. Computations of $\mathsf{DEM}^*(\mathsf{DT})$ end when the list of inputs is empty, and this ASM may diverge. $\mathsf{BEM}^*(\mathsf{DT})$ never diverges, but computations may stop with a nonempty list of inputs. Both ASMs are completely passive, they just react to buttons pushed, i.e. inputs provided by the environment.

Finally, when comparing refinements we have to consider the third problem, which is the difference in initialization and finalization: since ASMs are defined like automata using initial and final states, ASM refinement compares these states using two relations IR and OR. Data refinement is more complex, since it requires an additional set G, and initialization and finalization operations. Unfortunately there are cases where these two are incompatible: consider a correct data refinement and assume $\mathsf{CINIT}(\mathsf{g}, \mathsf{cs})$ and $(\mathsf{COP}_{\mathsf{is}} \mathbin{;} \mathsf{CFIN})(\mathsf{cs}, \mathsf{g}_\mathsf{k})$ for some is and both $\mathsf{k} = 1, 2$. Then there may be two different states as_1 and as_2 such that $\mathsf{AINIT}(\mathsf{g}, \mathsf{as}_\mathsf{k})$ and $(\mathsf{AOP}_{\mathsf{is}} \mathbin{;} \mathsf{AFIN})(\mathsf{as}_\mathsf{k}, \mathsf{g}_\mathsf{k})$ holds. This implies that in the ASM refinement $\mathsf{IR}(\mathsf{as}_\mathsf{k}, \mathsf{cs})$ should hold, to have a corresponding abstract run for both concrete runs starting with cs. But $\mathsf{IR}(\mathsf{as}_2, \mathsf{cs})$ also would imply that the run with result g_1 must have a corresponding run starting with as_2, which may not be the case. Therefore defining relations IR and OR such that the ASM refinement from $\mathsf{BEM}^*(\mathsf{ADT})$ to $\mathsf{BEM}^*(\mathsf{CDT})$ is correct is not possible in general.

There are two ways out of this problem: one is to extend the ASM rule to include the initialization and finalization operations. The other is to focus on refinements that can be verified using forward simulation.

The first solution defines $\mathsf{BINITRULE}$ and $\mathsf{BFINRULE}$ from INIT and FIN like $\mathsf{BRULE}_\mathsf{i}$ was defined from OP_i and has the following rule:

```
$\mathsf{EXTDERULE}(\mathsf{DT}) :=$ if $\neg\ \mathsf{final}$
                            then if $\mathsf{init}$
                                 then $\mathsf{BINITRULE}$
                                      $\mathsf{init} := \mathsf{false}$
                                 else if $\mathsf{il} \neq []$
                                      then $\mathsf{BRULE}_{\mathsf{head(il)}}$
                                           $\mathsf{il} := \mathsf{tail(il)}$
                                      else $\mathsf{BFINRULE}$
                                           $\mathsf{final} := \mathsf{true}$
```

The resulting ASM $\mathsf{EXTDEM}^*(\mathsf{DT})$ has a state consisting of four components $\mathsf{st} \in (\mathsf{G} \cup \mathsf{S})$, $\mathsf{il} \in \mathsf{I}^*$, $\mathsf{init} \in \mathsf{Bool}$ and $\mathsf{final} \in \mathsf{Bool}$. Initial states have $\mathsf{st} \in \mathsf{G}$, $\mathsf{init} = \mathsf{true}$ and $\mathsf{final} = \mathsf{false}$. The ASM diverges on attempts to apply an operation OP_i, when $\mathsf{st} \notin \mathsf{dom}(\mathsf{OP}_\mathsf{i})$. $\mathsf{EXTBEM}^*(\mathsf{DT})$ is defined like $\mathsf{EXTDEM}^*(\mathsf{DT})$, but with the additional check $\mathsf{st} \in \mathsf{dom}(\mathsf{OP}_{\mathsf{head(il)}})$ as for $\mathsf{BEM}^*(\mathsf{DT})$ above. This ASM stops computation, when $\mathsf{st} \notin \mathsf{dom}(\mathsf{OP}_{\mathsf{head(il)}})$. The extended ASMs allow to compare ASM refinement to data refinement in general as the following theorem holds:

Theorem 5 (Data refinement and ASM refinement)
The following three notions are equivalent:

- *The data refinement from* ADT *to* CDT *is correct in the behavioral approach*
- *The ASM refinement from* $\mathsf{EXTDEM}^*(\mathsf{ADT})$ *to* $\mathsf{EXTDEM}^*(\mathsf{CDT})$ *preserves total correctness for* IR *and* OR *both identity on* G.
- *The ASM refinement from* $\mathsf{EXTBEM}^*(\mathsf{ADT})$ *to* $\mathsf{EXTBEM}^*(\mathsf{CDT})$ *preserves total correctness for* IR *identity on* G *and*

$$\begin{aligned}&\mathsf{OR}(\mathsf{as}, \mathsf{il_A}, \mathsf{final_A}, \mathsf{cs}, \mathsf{il_C}, \mathsf{final_C}) := \\ &\qquad \mathsf{il_A} = \mathsf{il_C} \wedge \mathsf{final_A} \wedge \mathsf{final_C} \\ &\quad \wedge (\mathsf{il_A} = [] \rightarrow \mathsf{as} = \mathsf{cs} \in \mathsf{G}) \\ &\quad \wedge (\mathsf{il_A} \neq [] \rightarrow \mathsf{as} \notin \mathsf{dom}(\mathsf{AOP}_{\mathsf{head}(\mathsf{il_A})}) \wedge \mathsf{cs} \notin \mathsf{dom}(\mathsf{COP}_{\mathsf{head}(\mathsf{il_C})}))\end{aligned}$$

The proof is obvious by comparing definitions.

The theorem shows that interpreting $\perp$ as divergence in the behavioral approach is meaningful and even leads to a closer correspondence to ASM refinement than the (standard) blocking interpretation. The latter interpretation must explicitly mention runs which are aborted because of an inapplicable rule (last line in the definition of OR), while for the former $(\perp, \perp) \in \mathsf{OR}$ matches rules that diverge implicitly.

Adding initialization and finalization operations to the ASMs is necessary only if the data refinement can not be proved using forward simulation. Otherwise the simpler ASMs $\mathsf{DEM}^*(\mathsf{DT})$ (or $\mathsf{BEM}^*(\mathsf{DT})$) are sufficient:

Theorem 6 (Forward simulations in data and ASM refinement)

- *If the refinement from* ADT *to* CDT *can be verified using the forward simulation* R, *then the refinements from* $\mathsf{BEM}^*(\mathsf{ADT})$ *to* $\mathsf{BEM}^*(\mathsf{CDT})$ *and from* $\mathsf{DEM}^*(\mathsf{ADT})$ *to* $\mathsf{DEM}^*(\mathsf{CDT})$ *totally preserve traces for* $\mathsf{IO} := \mathsf{R}$. *This implies that they also preserve total correctness for* $\mathsf{IR} := \mathsf{R} \cap (\mathsf{AIN} \times \mathsf{CIN})$ *and* $\mathsf{OR} := \mathsf{R} \cap (\mathsf{AOUT} \times \mathsf{COUT})$.
- *If the ASM refinement from* $\mathsf{BEM}^*(\mathsf{ADT})$ *to* $\mathsf{BEM}^*(\mathsf{CDT})$ *(or the one from* $\mathsf{DEM}^*(\mathsf{ADT})$ *to* $\mathsf{DEM}^*(\mathsf{CDT})$*) can be verified using an invariant of the form*

$$\mathsf{INV}(\mathsf{as}, \mathsf{il_A}, \mathsf{cs}, \mathsf{il_C}) := \mathsf{R}(\mathsf{as}, \mathsf{cs}) \wedge \mathsf{il_A} = \mathsf{il_C} \tag{2}$$

  and 1:1 diagrams only, then the refinement of ADT *to* CDT *is correct and can be verified using the forward simulation* R.

The proof of Theorem 6 was formally done using KIV. For the first item it is shown, that the correctness conditions of forward simulation imply the conditions of ASM refinement when using the invariant as given in (2): the "initialize" condition follows from "initialization" and the fact that CINIT and

AINIT are total. “establish invariant” is trivial by definition. “correctness” and “applicability” are cases (B) with one rule application and (C) as given in Figure 2. “finalize” is trivial again. Note that since the ASMs do not use operations AFIN and CFIN, the “finalization” condition is not needed to establish the conditions of ASM refinement. It *is* needed for the proof of the second part of the theorem, i.e. that

$$\forall\ \mathsf{is} \in \mathsf{I}^*.\ \mathsf{CINIT} \mathbin{⨾} \mathsf{COP_{is}} \mathbin{⨾} \mathsf{CFIN} \subseteq \mathsf{AINIT} \mathbin{⨾} \mathsf{AOP_{is}} \mathbin{⨾} \mathsf{AFIN} \tag{3}$$

is implied by correctness of an ASM refinement with 1:1 diagrams. Again, due to the difference in initialization and finalization, there is a small gap between ASM refinement and data refinement: preservation of total correctness guarantees that $\mathsf{COP_{is}(cs, cs')}$ implies

$$\exists\ \mathsf{as, as'}.\ \mathsf{IR(as, cs)} \land \mathsf{AOP_{is}(as, as')} \land \mathsf{OR(as', cs')}$$

This is slightly weaker than the condition of data refinement: $\mathsf{COP_{is}(cs, cs')}$ and (3) implies

$$\forall\ \mathsf{as}.\ \mathsf{IR(as, cs)} \to \exists\ \mathsf{as'}.\ \mathsf{AOP_{is}(as, as')} \land \mathsf{OR(as', cs')}$$

i.e. it guarantees that *every* initial state as with $\mathsf{IR(as, cs)}$ has a corresponding abstract run to the concrete one from cs to $\mathsf{cs'}$ (and “initialization” implies there is at least one), while ASM refinement only requires that there is *some* initial state as with a corresponding run. Nevertheless inspection of the formal proof of Theorem 1 in [1] shows that the verification conditions of ASM refinement guarantees the stronger condition too: since the main lemma given as formula (16) on p. 28 already guarantees correspondence for *all* abstract runs, only the small part that proves the strengthened main theorem from this lemma has to be adapted.

Summarizing we have: *behavioral data refinement is refinement of ASMs with external control and finite sequences of operations.*

The assumption that the sequence $\mathsf{is} \in \mathsf{I}^*$ is finite has been made, to get the exact correspondence to data refinement. An infinite sequence $\mathsf{is} \in \mathsf{I}^\infty$ or both finite and infinite sequences $\mathsf{is} \in \mathsf{I}^\omega$ is possible too, resulting in ASMs $\mathsf{BEM^\infty(DT)}$, $\mathsf{DEM^\infty(DT)}$ and $\mathsf{BEM^\omega(DT)}$, $\mathsf{DEM^\omega(DT)}$. It is also possible to move control to the ASM itself. The simplest ASM in this case is e.g. $\mathsf{DIM^\infty(DT)}$ which has the rule

$$\mathsf{DIRULE^\infty(DT)} := \textbf{choose}\ \mathsf{i} \in \mathsf{I}\ \textbf{in}\ \mathsf{BRULE_i}$$

To block the application of rules outside their domain, there are again two possibilities: Either the ASM can choose $\mathsf{i} \in \mathsf{I}$ arbitrarily, and stop computation if for this choice $\mathsf{st} \notin \mathsf{dom(OP_i)}$ does not hold. Or the ASM can choose only such an $\mathsf{i} \in \mathsf{I}$ that satisfies $\mathsf{st} \notin \mathsf{dom}(\mathsf{OP_i})$ and stop only, if none is available. Both choices are possible. For the latter choice and a finite sequence of inputs $\mathsf{BIM}^*(\mathsf{DT})$ has the following rule:

```
$\mathsf{BIRULE}^*(\mathsf{DT}) :=$ if $\mathsf{cnt} \neq 0 \wedge \exists\, \mathsf{i}.\ \mathsf{st} \in \mathsf{dom}(\mathsf{OP_i})$
                    then choose $\mathsf{i} \in \mathsf{I}$ with $\mathsf{st} \in \mathsf{dom}(\mathsf{OP_i})$
                         in $\mathsf{BRULE_i}$
                            $\mathsf{cnt} := \mathsf{cnt} - 1$
```

The rule uses an additional state component cnt to limit the number of rule applications. Its initial value is an arbitrary natural number.

For all of these machines we can prove that data refinement using forward simulation implies ASM refinement:

Theorem 7 (Forward simulation (behavioral) implies ASM ref.) *If the refinement of* ADT *to* CDT *can be proven correct using forward simulation with* R *in either for the approach with total relations or with respect to the behavioral approach, then each refinement from* $\mathsf{YXM}^\alpha(\mathsf{ADT})$ *to* $\mathsf{YXM}^\alpha(\mathsf{CDT})$*, where* $\mathsf{Y} \in \{\mathsf{B}, \mathsf{D}\}$*,* $\mathsf{X} \in \{\mathsf{E}, \mathsf{I}\}$ *and* $\alpha \in \{*, \infty, \omega\}$ *is an ASM refinement that totally preserves traces with respect to* R*.*

The theorem is proved using

$$\mathsf{INV}(\mathsf{as}, \mathsf{gs_A}, \mathsf{cs}, \mathsf{gs_C}) := \mathsf{R}(\mathsf{as}, \mathsf{cs}) \wedge \mathsf{gs_A} = \mathsf{gs_C}$$

6 The Contract Approach and ASM refinement

While data refinement is an instance of ASM refinement in the behavioral approach, things are different for the contract approach, even though the interpretation of $\bot$ as divergence seems to be a closer match. There are two differences: the first shows up, when one tries to find an ASM rule that formalizes the chaotic behavior of operations outside their domain. The closest match is

$$\mathsf{CRULE_i} := \textbf{if } \mathsf{st} \in \mathsf{dom}(\mathsf{OP_i}) \textbf{ then } \mathsf{TRULE_i} \textbf{ else chaos}$$

where **chaos** changes the state randomly or diverges. Formally:

$$\textbf{chaos} := \textbf{choose } \mathsf{b} \in \mathsf{Bool}, \mathsf{s'} \in \mathsf{S} \textbf{ in if } \mathsf{b} \textbf{ then } \mathsf{st} := \mathsf{s'} \textbf{ else abort}$$

The semantics of $\mathsf{CRULE_i}$ is

$$\widetilde{\widetilde{\mathsf{OP}}}_\mathsf{i} := \mathsf{OP_i} \cup ((\mathsf{S} \setminus \mathsf{dom}(\mathsf{OP_i})) \times \mathsf{S}_\bot) \cup \{(\bot, \bot)\}$$

while $\widetilde{\mathsf{OP}}_\mathsf{i}$ additionally includes $\{\perp\} \times \mathsf{S}$. This semantics has the strange effect that an operation can recover from divergence to produce a final result in G. It seems, one motivation for this semantics is to allow the refinement relation to be the subset relation [2]. The refinement relation is also the source for the second difference: the contract approach may implement a diverging run by a terminating one.

The first difference is of technical nature. Using an ASM rule $\mathsf{CRULE}_\mathsf{i}$ with semantics $\widetilde{\widetilde{\widetilde{\mathsf{OP}}}}_\mathsf{i}$ can be justified by the following theorem, which uses a refinement relation $\subseteq^\perp$, that explicitly mentions the possibility of implementing a diverging by a terminating run:

Theorem 8 (Modified semantics for the contract approach)
If the verification conditions of the contract approach hold, then

$$\forall\ \mathsf{is} \in \mathsf{I}^*.\ \widetilde{\widetilde{\widetilde{\mathsf{CINIT}}}} \fatsemi \widetilde{\widetilde{\widetilde{\mathsf{COP}}}}_\mathsf{is} \fatsemi \widetilde{\widetilde{\widetilde{\mathsf{CFIN}}}} \subseteq^\perp \widetilde{\widetilde{\widetilde{\mathsf{AINIT}}}} \fatsemi \widetilde{\widetilde{\widetilde{\mathsf{AOP}}}}_\mathsf{is} \fatsemi \widetilde{\widetilde{\widetilde{\mathsf{AFIN}}}}$$

where

$$\mathsf{R}_1 \subseteq^\perp \mathsf{R}_2 :\ \leftrightarrow \forall\ \mathsf{g}, \mathsf{g}' \in \mathsf{G}_\perp.\ \mathsf{R}_1(\mathsf{g}, \mathsf{g}') \rightarrow \mathsf{R}_2(\mathsf{g}, \mathsf{g}') \vee \mathsf{R}_2(\mathsf{g}, \perp)$$

$\widetilde{\widetilde{\widetilde{\mathsf{CINIT}}}} \fatsemi \widetilde{\mathsf{R}} \subseteq \widetilde{\widetilde{\widetilde{\mathsf{AINIT}}}}$ *is equivalent to "initialization". $\widetilde{\mathsf{R}} \fatsemi \widetilde{\widetilde{\widetilde{\mathsf{COP}}}}_\mathsf{i} \subseteq \widetilde{\widetilde{\widetilde{\mathsf{COP}}}}_\mathsf{i} \fatsemi \widetilde{\mathsf{R}}$ is equivalent to the "correctness" and "applicability" condition. $\widetilde{\mathsf{R}} \fatsemi \widetilde{\widetilde{\widetilde{\mathsf{CFIN}}}} \subseteq^\perp \widetilde{\widetilde{\widetilde{\mathsf{AFIN}}}}$ is equivalent to "finalization".*

The formal proof in KIV for the first part is a simple induction over the length of is. The second part uses similar calculations than the ones of the original proof. The theorem shows that the use of a semantics, which recovers from $\perp$, is not mandatory in the contract approach.

The fact, that data refinement in the contract approach allows to implement a diverging run by a terminating one, is more difficult to accommodate. It would require a different embedding of the relations used in ASM refinement: IO, IR, OR and INV would have to relate $\perp$ on the abstract level to every state (including $\perp$) on the concrete level. Also, we would have to assume $\perp \notin \mathsf{OUT}$ and to include $\{(\perp, \perp)\} \in \mathsf{RULE}$.

We leave an investigation of the contract embedding for ASM refinement and the derivation of general commuting diagrams, similar to the ones given in Figure 2, as future work, but would like to remark that there are two restrictions implied by the contract approach: first, although the embedding is sufficient to guarantee that total correctness assertions are preserved (since *no* total

[2] Another reason is that the semantics of recursion together with (infinite) nondeterminism is simpler for a demonic semantics (see [10]) than in the general case as defined e.g. in [26].

correctness assertions hold for a diverging run), partial correctness assertions are preserved only for rules with a demonic semantics (i.e. when $(\mathsf{s}, \perp) \in \mathsf{RULE}$ implies $(\mathsf{s}, \mathsf{s}') \in \mathsf{RULE}$ for all s'; potential divergence is considered as bad as mandatory divergence). This would require to forbid an ASM rule like

choose $\mathsf{b} \in \mathsf{Bool}$ **in if** b **then skip else abort**

which may nondeterministically diverge (**abort**) or behave like identity (**skip**), and which has the semantics $\mathsf{id} \cup \mathsf{S} \times \{\perp\}$.

Second, the weaker embedding seems neither appropriate for compiler verification nor for reactive systems: in compiler verification, if an interpreter of the source code (i.e. the ASM of the abstract level) fails to deliver a result by diverging (or more general: by diverging or returning an error), it does not seem reasonable that the processor (i.e the ASM of the concrete level) should produce a correct result for the compiled program.

Similarly, for reactive systems, a run that diverges immediately should not be implemented with a (finite or infinite) trace of the system which performs input and output as expected.

Even though refinement of diverging with terminating runs is not possible in ASM refinement, rules with chaotic behavior outside their domain can be accommodated as the following theorem shows. The theorem uses CXM^α, which is the transition system with the same definition as DXM^α, except that $\mathsf{BRULE}_\mathsf{i}$ has been replaced by $\mathsf{CRULE}_\mathsf{i}$.

Theorem 9 (Forward Simulation (contract) and ASM refinement) *If the refinement of* ADT *to* CDT *can be verified using forward simulation with* R *using the contract approach, and if*

$\forall\, \mathsf{cs}.\ \exists\, \mathsf{as}.\ \mathsf{R}(\mathsf{as}, \mathsf{cs})$ *("full range")*

(or equivalently $\mathsf{ran}(\mathsf{R}) = \mathsf{CS}$*) holds, then all refinements from* $\mathsf{CXM}^\alpha(\mathsf{ADT})$ *to* $\mathsf{CXM}^\alpha(\mathsf{CDT})$ *where* $\mathsf{X} \in \{\mathsf{E}, \mathsf{I}\}$ *and* $\alpha \in \{*, \infty, \omega\}$ *totally preserve traces.*

Again, this theorem has been formally verified using KIV. The extra "full range" condition prevents implementing diverging runs by terminating ones. A diagram with $(\mathsf{as}, \mathsf{cs}) \in \mathsf{R}$, $\mathsf{as} \notin \mathsf{dom}(\mathsf{AOP}_\mathsf{i})$, $(\mathsf{cs}, \mathsf{cs}') \in \mathsf{COP}_\mathsf{i}$ now commutes by choosing an arbitrary as' with $\mathsf{R}(\mathsf{as}', \mathsf{cs}')$.

7 Generalizing Data Refinement to ASM Refinement

Data refinement considers ASMs where control must be purely external. But many of the case studies that were done in ASM refinement are not of this restricted kind, and from the examples we have looked at, neither are many examples done in specification languages such as B [27], Z [21] or VDM [28]. Typically, there may be input (in ASMs via external functions, in Z via input parameters) that directly or indirectly influences the transitions an ASM does, but the ASM decides itself which rule to apply next, not the environment.

An ASM which chooses the next rule application itself is typically the result of encoding a program that uses the operations of a data type as a transition system. Such encodings are common in temporal logic (e.g. in TLA [19], or in the Step prover [29]). They result in an interpreter, which typically has a program counter (or several, if parallel programs are considered) which controls execution. Abstractly, such an ASM has a current global state gst (a 0-ary dynamic function) with values in some set G in addition to the current state st of the data type. In addition to executing some $\mathsf{OP_i}$ that changes the local state st, the global state gst is modified executing an ASM rule CTRL that may read both st and gst. The choice which rule to apply next (or to stop) is controlled by a predicate p. Assuming initial states $\mathsf{IN} \subseteq \mathsf{S} \times \mathsf{G}$, this leads to the definition of an ASM

$$\mathsf{GM}(\mathsf{DT}, \mathsf{p}, \mathsf{G}, \mathsf{IN}, \mathsf{CTRL}) := (\mathsf{S} \times \mathsf{G}, \mathsf{IN}, \mathsf{GRULE}(\mathsf{DT}, \mathsf{p}, \mathsf{G}, \mathsf{CTRL}))$$

with the following rule:

$$\begin{aligned}\mathsf{GRULE}(\mathsf{DT}, \mathsf{p}, \mathsf{G}, \mathsf{CTRL}) = \ &\mathbf{if}\ \exists\ \mathsf{i}.\ \mathsf{p}(\mathsf{st}, \mathsf{gst}, \mathsf{i})\\ &\mathbf{then\ choose}\ \mathsf{i} \in \mathsf{I}\ \mathbf{with}\ \mathsf{p}(\mathsf{st}, \mathsf{gst}, \mathsf{i})\\ &\qquad \mathbf{in}\ \mathsf{CTRL}\\ &\qquad\quad \mathsf{TRULE_i}\end{aligned}$$

Note, that the predicate p may implement guards, which truly block rules from being executed, so we may have guards as well as preconditions in ASM rules. Therefore simulating the guards and preconditions of the B specification language [27] or the approach in [30] should be possible, but we leave a careful analysis of this topic to further research. Many variations of this ASM are possible: e.g. operations may read from or write to other streams stored in G than the sequence of indices that were present in the machines with external control (as is done in IO refinement in Z). Also INIT and FIN may be used as additional operations. Therefore this ASM is not a “most general” ASM. Nevertheless it has the property that it subsumes the ASMs of the previous section by suitable instantiation of p,G, IN and CTRL. E.g. $\mathsf{DEM}^*(\mathsf{DT})$ is the instance with

$$\begin{aligned}&\mathsf{G} := \mathsf{I}^*, \mathsf{p}(\mathsf{st}, \mathsf{il}, \mathsf{i}) := \mathsf{il} \neq [\,] \wedge \mathsf{i} = \mathsf{head}(\mathsf{il}) \text{ and}\\ &\mathsf{CTRL} = \mathbf{if}\ \mathsf{st} \in \mathsf{dom}(\mathsf{OP}_{\mathsf{head(il)}})\ \mathbf{then}\ \mathsf{il} := \mathsf{tail}(\mathsf{il})\ \mathbf{else\ abort}\end{aligned}$$

A refinement of $\mathsf{GM}(\mathsf{ADT}, \mathsf{p}, \mathsf{G}, \mathsf{IN}, \mathsf{CTRL})$ may consist in data refinement, and choose to replace the operations by those of CDT. It is correct then, if the data refinement is correct, and the possible rules that may be selected are reduced:

$$\mathsf{R}(\mathsf{as},\mathsf{cs}) \wedge \mathsf{p_C}(\mathsf{cs},\mathsf{g},\mathsf{i}) \rightarrow \mathsf{p_A}(\mathsf{as},\mathsf{g},\mathsf{i})$$

But now that the ASM has operations *and control* it is natural to consider not only (data) refinement of operations, but also refinement of the control structure CTRL. A simple example is moving control from or to the ASM:

Theorem 10 (Internal and external control) *Given a data type* DT*,* $\mathsf{XEM}^\alpha(\mathsf{DT})$ *is a correct and complete (i.e. the reverse refinement is also correct) refinement of* $\mathsf{XIM}^\alpha(\mathsf{DT})$*, for every* $\mathsf{X} \in \{\mathsf{B},\mathsf{D},\mathsf{C}\}$*,* $\alpha \in \{*,\infty,\omega\}$ *that totally preservers traces for* $\mathsf{R} := \mathsf{id_S}$*.*

The proof that the refinement is correct is a simple forward simulation as in data refinement. Note that the "total range" constraint of Theorem 8 is trivially satisfied for $\mathsf{R} = \mathsf{id}$ in the case $\mathsf{X} = \mathsf{C}$. For completeness, backward simulation proves the case where $\alpha = *$. The other two cases require to look at infinite traces σ_A of XIM. Now each two successive states are related by some $\mathsf{OP_i}$: we have $\mathsf{OP}_{\mathsf{i_k}}(\sigma(\mathsf{k}),\sigma(\mathsf{k}+1))$. Therefore, XEM will execute the same trace given the input sequence $\lambda\,\mathsf{k}.\mathsf{i_k}$, proving refinement. Many other examples of refinements of the control structure are found in compiler verification, e.g. refinement may do peephole optimization (see chapter 7 in [14]).

Recently generalizations of refinement that modify control have also been considered in data refinement, and we will study two such generalizations in the following two sections, weak refinement and coupled refinement. A fourth simple generalization is refinement still using arbitrary 1:1 diagrams, but dropping the requirement of conformality using a relation Π between the two index sets I and J. At least one (usually exactly one) abstract operation for every concrete one is required.

Theorem 11 (ASM refinement with 1:1 diagrams)

Given two data types $\mathsf{ADT} = (\mathsf{G},\mathsf{AS},\mathsf{AINIT},\{\mathsf{AOP_i}\}_{\mathsf{i}\in\mathsf{I}},\mathsf{AFIN})$ *and* $\mathsf{CDT} = (\mathsf{G},\mathsf{CS},\mathsf{CINIT},\{\mathsf{COP_j}\}_{\mathsf{j}\in\mathsf{J}},\mathsf{CFIN})$*, and a relation* $\Pi \subseteq \mathsf{I} \times \mathsf{J}$ *with* $\mathsf{ran}(\Pi) = \mathsf{J}$*, the ASM refinement from* $\mathsf{GM}(\mathsf{ADT},\mathsf{p_A},\mathsf{G},\mathsf{ran}(\mathsf{AINIT}) \times \mathsf{G},\mathsf{CTRL})$ *to* $\mathsf{GM}(\mathsf{CDT},\mathsf{p_C},\mathsf{G},\mathsf{ran}(\mathsf{CINIT}) \times \mathsf{G},\mathsf{CTRL})$ *totally preserves traces with respect to* R*, if*

- $\mathsf{CINIT} \subseteq \mathsf{AINIT} \mathbin{;} \mathsf{R}$ *("initialization")*
- $\forall\, \mathsf{g} \in \mathsf{G}.\ \forall\, \mathsf{i},\mathsf{j}.\quad \Pi(\mathsf{i},\mathsf{j}) \wedge \mathsf{p_C}(\mathsf{cs},\mathsf{g},\mathsf{j}) \wedge \mathsf{R}(\mathsf{as},\mathsf{cs}) \wedge \mathsf{COP_j}(\mathsf{cs},\mathsf{cs}')$
  $\rightarrow \exists\, \mathsf{as}'.\ \mathsf{AOP_i}(\mathsf{as},\mathsf{as}') \wedge \mathsf{R}(\mathsf{as}',\mathsf{cs}')$ *("correctness")*
- $\forall\, \mathsf{g} \in \mathsf{G}.\ \forall\, \mathsf{i} \in \mathsf{I}.\quad \mathsf{p_C}(\mathsf{cs},\mathsf{g},\mathsf{j}) \wedge \mathsf{R}(\mathsf{as},\mathsf{cs}) \wedge \mathsf{cs} \notin \mathsf{dom}(\mathsf{COP_j})$
  $\rightarrow \mathsf{as} \notin \mathsf{dom}(\mathsf{AOP_i})$ *("applicability")*
- $\forall\, \mathsf{g} \in \mathsf{G}.\ \forall\, \mathsf{i},\mathsf{j}.\ (\mathsf{i},\mathsf{j}) \in \Pi \wedge \mathsf{R}(\mathsf{as},\mathsf{cs}) \wedge \mathsf{p_C}(\mathsf{cs},\mathsf{g},\mathsf{j}) \rightarrow \mathsf{p_A}(\mathsf{as},\mathsf{g},\mathsf{i})$
  ("guard inclusion")

Note that for $\mathsf{p_A} = \mathsf{p_C} = \mathsf{true}$ the conditions are equal to the original conditions of behavioral data refinement, except that $\mathsf{COP_i}$ has been replaced by $\mathsf{COP_j}$. For data refinement [20] defines a similar notion called alphabet translation. For simplicity, an injective mapping $\pi : \mathsf{I} \to \mathsf{J}$ is assumed (if necessary, operations on the concrete level can be duplicated). Alphabet translation allows to add arbitrary extra operations outside $\mathsf{ran}(\pi)$, which is not allowed in ASM refinement, since it may add new behavior. Adding operations can be done, when the data type defines a constraint, which defines an upper bound on what new operations can do, as in the approach of [31]. The constraint is then viewed as an abstract operation that all new methods must implement.

A correct refinement with only 1:1 diagrams that totally preserves traces with respect to R still shares the important property with data refinement that it preserves invariants as well as both partial and total correctness assertions for every pair of corresponding operations:

Theorem 12 (Partial and total correctness for 1:1 diagrams)
Given a correct refinement as in the previous theorem, and a pre- and post-condition for $\mathsf{AOP_i}$*, then the refinement preserves partial and total correctness for every pair* $(\mathsf{i},\mathsf{j}) \in \Pi$ *and every pair of pre- postconditions* φ, ψ*:*

$$\begin{array}{l} \quad(\varphi_{\mathsf{as}}^{\mathsf{ast}} \to [\mathsf{ARULE_i}]\ \psi_{\mathsf{as}}^{\mathsf{ast}}) \\ \to (\exists\ \mathsf{as}.\ \mathsf{R}(\mathsf{as},\mathsf{cst}) \wedge \varphi) \to [\mathsf{CRULE_j}]\ (\exists\ \mathsf{as}.\ \mathsf{R}(\mathsf{as},\mathsf{cst}) \wedge \psi) \end{array} \tag{4}$$

$$\begin{array}{l} \quad(\varphi_{\mathsf{as}}^{\mathsf{ast}} \to \langle\!|\mathsf{ARULE_i}|\!\rangle\ \psi_{\mathsf{as}}^{\mathsf{ast}}) \\ \to (\exists\ \mathsf{as}.\ \mathsf{R}(\mathsf{as},\mathsf{cst}) \wedge \varphi) \to \langle\!|\mathsf{CRULE_j}|\!\rangle\ (\exists\ \mathsf{as}.\ \mathsf{R}(\mathsf{as},\mathsf{cst}) \wedge \psi) \end{array} \tag{5}$$

If $\varphi_{\mathsf{as}}^{\mathsf{ast}}$ *is an invariant of all abstract operations, then* $\exists\ \mathsf{as}.\ \mathsf{R}(\mathsf{as},\mathsf{cst}) \wedge \varphi$ *is an invariant of all concrete operations.*

Note that since ASM rules $\mathsf{ARULE_i}$ and $\mathsf{CRULE_j}$ modify the 0-ary dynamic functions ast and cst, which can not be quantified in first-order logic, variable as must be substituted by ast in φ and ψ as shown. Also remember, that $\varphi \to [\alpha]\ \psi$ in Formula (4) is equivalent to the Hoare triple $\{\varphi\}\alpha\{\psi\}$, and that $\varphi \to \langle\!|\alpha|\!\rangle\ \psi$ in (5) is the same as $\varphi \to \mathsf{wp}(\alpha, \psi)$ in Dijkstra's wp calculus. Implementation of operations given by pre- postcondition is the specific case, where 1:1 diagrams are useful. Other diagram types than 1:1 diagrams as used in ASM refinement do not preserve partial and total correctness assertions for operations any longer. Invariants are only preserved in a weaker sense (see Chapter 6.5 in [14]).

8 Weak Refinement as an Instance of ASM Refinement

Weak refinement [20] assumes that the operations of a data type are divided into external and internal operations. The index set is the disjoint union $\mathsf{I} \dot{\cup} \mathsf{E}$ and OP_i is external if $\mathsf{i} \in \mathsf{E}$. External operations are controlled by the environment. In between two external operations, a program over the data type may execute any number of internal operations. In a refinement between two data types, conformality must be kept between external operations, while the set of internal operations may be arbitrarily changed (using different sets of indices I and J, both disjoint from E).

Definition 8 (Weak data refinement)
Given data types $\mathsf{ADT} = (\mathsf{G}, \mathsf{AS}, \mathsf{AINIT}, \{\mathsf{AOP}_\mathsf{i}\}_{\mathsf{i} \in \mathsf{E} \dot{\cup} \mathsf{I}}, \mathsf{AFIN})$ *and* $\mathsf{CDT} = (\mathsf{G}, \mathsf{CS}, \mathsf{CINIT}, \{\mathsf{COP}_\mathsf{j}\}_{\mathsf{j} \in \mathsf{E} \dot{\cup} \mathsf{J}}, \mathsf{CFIN})$ *the refinement from* ADT *to* CDT *is a correct weak refinement, if*[3]

$$\begin{array}{ll}\forall\, \mathsf{js} \in \mathsf{J}^*, \mathsf{gs}, \mathsf{gs}'. & (\mathsf{CINIT} \fatsemi \mathsf{COP}_\mathsf{js} \fatsemi \mathsf{CFIN})(\mathsf{gs}, \mathsf{gs}') \\ & \rightarrow \exists\, \mathsf{is} \in \mathsf{I}^*.\ \mathsf{is}|_\mathsf{E} = \mathsf{js}|_\mathsf{E}\ \wedge\ (\mathsf{AINIT} \fatsemi \mathsf{AOP}_\mathsf{is} \fatsemi \mathsf{AFIN})(\mathsf{gs}, \mathsf{gs}')\end{array}$$

where $\mathsf{is}|_\mathsf{E}$ *and* $\mathsf{js}|_\mathsf{E}$ *are the restrictions of* is *and* js *to elements in* E.

Weak refinement can be proved using either a weak forward or a weak backward simulation. For a weak forward simulation R, the correctness criterion is:

Theorem 13 (Weak forward simulation) *A weak data refinement from* ADT *to* CDT *with total relations* AOP_i, COP_i *is correct, if a relation* R *and a wellfounded order* $<$ *on* CS *can be found such that*

- $\mathsf{CINIT} \fatsemi \mathsf{CIOPS} \subseteq \mathsf{AINIT} \fatsemi \mathsf{AIOPS} \fatsemi \mathsf{R}$ *("initialization")*
- $\forall\, \mathsf{i} \in \mathsf{E}. \quad \mathsf{R} \fatsemi \mathsf{CIOPS} \fatsemi \mathsf{COP}_\mathsf{i} \fatsemi \mathsf{CIOPS}$
  $\subseteq \mathsf{AIOPS} \fatsemi \mathsf{AOP}_\mathsf{i} \fatsemi \mathsf{AIOPS} \fatsemi \mathsf{R}$ *("correctness")*
- $\mathsf{R} \fatsemi \mathsf{CIOPS} \fatsemi \mathsf{CFIN} \subseteq \mathsf{AIOPS} \fatsemi \mathsf{AFIN}$ *("finalization")*
- $\forall\, \mathsf{j} \in \mathsf{J}.\ \mathsf{R}(\mathsf{as}, \mathsf{cs}) \wedge \mathsf{COP}_\mathsf{j}(\mathsf{cs}, \mathsf{cs}') \rightarrow \mathsf{cs}' < \mathsf{cs}$ *("decrease")*

where $\mathsf{CIOPS} := (\bigcup_{\mathsf{j} \in \mathsf{J}} \mathsf{COP}_\mathsf{j})^*$ *and* $\mathsf{AIOPS} := (\bigcup_{\mathsf{i} \in \mathsf{I}} \mathsf{AOP}_\mathsf{i})^*$ *are the reflexive, transitive closures of the internal operations.*

Two remarks should be made about the theorem: first, weak data refinement need not satisfy that for $\mathsf{j} \in \mathsf{J}$

$$\mathsf{R}(\mathsf{as}, \mathsf{cs}) \wedge \mathsf{COP}_\mathsf{j}(\mathsf{cs}, \mathsf{cs}') \rightarrow \exists\, \mathsf{as}'.\ \mathsf{AIOPS}(\mathsf{as}, \mathsf{as}') \wedge \mathsf{R}(\mathsf{as}', \mathsf{cs}') \tag{6}$$

[3] See [32] for an equivalent definition

$$\begin{tikzcd}
\cdot \arrow[r, "\mathsf{AINIT}"] \arrow[dr, "\mathsf{CINIT}"'] & \cdot \arrow[r, "\mathsf{AIOPS}"] & \cdot \arrow[d, dash, "\mathsf{R}"] \\
& \cdot \arrow[r, "\mathsf{CIOPS}"'] & \cdot
\end{tikzcd}
\qquad
\begin{tikzcd}
\cdot \arrow[r, "\mathsf{AIOPS}"] \arrow[d, dash, "\mathsf{R}"'] & \cdot \arrow[r, "\mathsf{AIOP_e}"] & \cdot \arrow[r, "\mathsf{AIOPS}"] & \cdot \arrow[d, dash, "\mathsf{R}"] \\
\cdot \arrow[r, "\mathsf{CIOPS}"'] & \cdot \arrow[r, "\mathsf{CIOP_e}"'] & \cdot \arrow[r, "\mathsf{CIOPS}"'] & \cdot
\end{tikzcd}
\qquad
\begin{tikzcd}
\cdot \arrow[r, "\mathsf{AIOPS}"] \arrow[d, dash, "\mathsf{R}"'] & \cdot \arrow[r, "\mathsf{AFIN}"] & \cdot \\
\cdot \arrow[r, "\mathsf{CIOPS}"'] & \cdot \arrow[ur, "\mathsf{CFIN}"'] &
\end{tikzcd}$$

Fig. 4. Verification of a weak refinement using commuting diagrams

as claimed in [20], since the abstract states needed for different numbers of internal steps may all be on *different* traces (assume AOP, that adds some random number n to the state, while CDT calls an incrementing COP n times). Second, the definition in [20] uses an expression E that must decrease, to define the well founded order. For the approach to be correct, it is necessary that this expression is defined over CS only, not over $\mathsf{CS} \times \mathsf{AS}$ (as the analogy to ASM refinement would suggest).

ASM refinement is almost an instance of weak refinement, where the ASM rule is viewed as an internal operation. The match is not exact for two reasons: ASM rules may express relations that are an arbitrary subset of $(\mathsf{S} \times \mathsf{S} \cup \{\perp\})$, and ASMs do not require that any state is “potentially final”. This holds for data types, since finalization is total, i.e. applicable in any state. An ASM decides internally when a run is finished. The fact that finalization may be called at any point during the computation prevents the use of m:n diagrams with n > 1.

Therefore the conditions for a correct weak refinement are stronger than those of ASM refinement: weak refinement assumes that *all* diagrams as shown in Figure 4 commute. This implies, that for every intermediate state cs during the run of the concrete machine, there exists an abstract state as where the same number of external operations has been executed, such that $\mathsf{R}(\mathsf{as}, \mathsf{cs})$. The weaker condition of ASM refinement requires only that the coupling invariant holds *at least once* between two external operations. It is easy to define refinements that satisfy the weaker condition of ASM refinement for a specific invariant, but not the condition of weak data refinement: almost all refinements in compiler verification are of this type. They refine an operation that interprets a certain source code operation (which typically is an internal operation, since most instructions of a programming language do not produce output or consume input) non-atomically by a sequence of a n assembler instructions. ASM refinement will verify the corresponding 1:n diagrams, using a coupling invariant that does *not* hold in the middle of these diagrams.

To show that weak refinement is an instance of ASM refinement, we consider an ASM $\mathsf{WM}(\mathsf{DT})$ (again this is an instance of the generic ASM GM) that has a finite *or* infinite sequence of external operations $\mathsf{es} \in \mathsf{E}^\omega$ stored in a program variable el. The ASM blocks operations which are not applicable. In each transition the ASM chooses nondeterministically either to stop, to execute the external operation $\mathsf{OP}_{\mathsf{head(el)}}$, or to perform an arbitrary internal operation. Since in general, weak refinement does not allow a definition of OR (the problematic case is, when for $\mathsf{R}(\mathsf{as},\mathsf{cs})$, $\mathsf{CFIN}(\mathsf{cs},\mathsf{gs}_1)$ and $\mathsf{CFIN}(\mathsf{cs},\mathsf{gs}_2)$ two *different* abstract states as_1 and as_2 are necessary such that $\mathsf{AIOPS}(\mathsf{as},\mathsf{as}_1)$, $\mathsf{AFIN}(\mathsf{as}_1,\mathsf{gs}_1)$, $\mathsf{AIOPS}(\mathsf{as},\mathsf{as}_2)$ and $\mathsf{AFIN}(\mathsf{as}_1,\mathsf{gs}_2)$) the ASM rule applies FIN with $\mathsf{BFINRULE}$ explicitly:

```
WRULE(DT) := if $\neg$ final then
             choose b $\in$ Bool, i $\in$ I $\cup$ E
             with b $\vee$ st $\in$ dom(OP_i) $\wedge$
                     (i $\in$ I $\vee$ el $\neq$ [] $\wedge$ i = head(el))
             in if b
                then BFINRULE
                       final := true
                else TRULE_i
                       if el $\neq$ [] $\wedge$ i = head(el) then el := tail(el)
```

Even for finite input $\mathsf{es} \in \mathsf{E}^*$, this ASM may have infinite runs that consist of internal operations only (livelock), a possibility implicitly assumed in [20]. Since data refinement considers finite runs only, neither the "no livelock" assumption for the abstract machine, nor the "decrease" condition, which ensures no livelock for the concrete machine is really needed in the correctness proof (it would not be needed if the ASM would execute a finite number of operations either). To show that the refinement preserves fairness is simple, since the coupling invariant will imply that $\mathsf{el}_\mathsf{A} = \mathsf{el}_\mathsf{C}$. So, if the concrete trace infinitely often removes elements from the stream el_C, the abstract trace must do the same to keep the equality.

Before we can verify that $\mathsf{WM}(\mathsf{CDT})$ refines $\mathsf{WM}(\mathsf{ADT})$, we have to consider the problem of initialization: weak data refinement does not always allow to define an equivalence relation IR on $\mathsf{AIN} \times \mathsf{CIN}$, since two runs of CDT that start with the same initial state cs (i.e. $\mathsf{CINIT}(\mathsf{gs},\mathsf{cs})$ for some gs) may have corresponding abstract runs that start with two different initial abstract states as_1 and as_2 (with $\mathsf{AINIT}(\mathsf{gs},\mathsf{as}_1)$ and $\mathsf{AINIT}(\mathsf{gs},\mathsf{as}_2)$). Since we have not seen such examples in practice, and condition (6) even implies impossibility of this case, we chose to define IR

$$\mathsf{IR}(\mathsf{as},\mathsf{cs}) :\leftrightarrow \exists\, \mathsf{gs}.\ \mathsf{CINIT}(\mathsf{gs},\mathsf{cs}) \wedge (\mathsf{AIOPS} \,⨾\, \mathsf{R})(\mathsf{as},\mathsf{cs}) \tag{7}$$

State as is the initial state that is used to complete the smallest initial diagram, where no concrete internal operations have been applied. Assuming

$$\mathsf{IR} \,⨾\, \mathsf{CIOPS} \subseteq \mathsf{AIOPS} \,⨾\, \mathsf{R} \tag{8}$$

we can verify:

Theorem 14 (Weak forward simulation implies ASM refinement) *If the refinement from* ADT *to* CDT *is a correct weak data refinement, and if* IR *as defined in (7) satisfies (8), then the refinement of* $\mathsf{WM(ADT)}$ *to* $\mathsf{WM(CDT)}$ *is a correct ASM refinement that totally preserves traces for* R *and total correctness for*

$$\mathsf{OR}(\mathsf{as}, \mathsf{el_A}, \mathsf{final_A}, \mathsf{cs}, \mathsf{el_C}, \mathsf{final_C}) := \mathsf{as} = \mathsf{cs} \in \mathsf{G} \wedge \mathsf{final_A} \wedge \mathsf{final_C} \wedge \mathsf{el_A} = \mathsf{el_C}$$

The operations $\mathsf{OP_i}$ *may all be partial in this refinement.*

Note that even though partial relations are used in the theorem, only the conditions of Theorem 13 are needed, an "applicability" condition as in the behavioral approach is not necessary for the correctness proof. The reason is that the ASM has no blocking behavior: if neither an internal nor the next external operation is applicable, the ASM applies the finalization operation.

The direct proof with KIV that we did is technically rather involved, the simple attempt with

$$\mathsf{INV}(\mathsf{as}, \mathsf{el_A}, \mathsf{final_A}, \mathsf{cs}, \mathsf{el_C}, \mathsf{final_C}) := \mathsf{R}(\mathsf{as}, \mathsf{cs}) \wedge \mathsf{el_A} = \mathsf{el_C} \wedge \mathsf{final_A} = \mathsf{final_C}$$

fails. The problem lies in the definition of the $<$ predicate on concrete states: consider a run σ_C starting with cs, and assume $\mathsf{R}(\mathsf{as}, \mathsf{cs})$ holds. To guarantee absence of livelock it must be proved that the states on the trace decrease, while internal operations are applied: $\sigma_\mathsf{C}(0) > \sigma_\mathsf{C}(1) > \sigma_\mathsf{C}(2) > \ldots$ implies that ultimately an external operation must be applied. Now $\sigma_\mathsf{C}(0) > \sigma_\mathsf{C}(1)$ follows from "decrease", since we have a corresponding abstract state as. But to show, that $\sigma_\mathsf{C}(1) > \sigma_\mathsf{C}(2)$ a state state as' is necessary with $\mathsf{R}(\mathsf{as}', \sigma_\mathsf{C}(1))$. Now, since condition (6) does not hold in general, $\mathsf{R}(\mathsf{as}, \mathsf{cs})$ is not sufficient to guarantee the existence of this state on the trace starting with as.

Therefore we relate each state cs with a state as, that occurred one diagram earlier:

$$\begin{aligned}
&\mathsf{INV}(\mathsf{as}, \mathsf{el_A}, \mathsf{final_A}, \mathsf{cs}, \mathsf{el_C}, \mathsf{final_C}) :\leftrightarrow \\
&\qquad\qquad \neg\, \mathsf{final_A} \wedge \neg\, \mathsf{final_C} \\
&\qquad\quad \wedge (\exists\, \mathsf{i} \in \mathsf{E}.\ \mathsf{el_A} = [\mathsf{i}] + \mathsf{el_C} \wedge (\mathsf{R} \mathbin{⨾} \mathsf{CIOPS} \mathbin{⨾} \mathsf{COP_i} \mathbin{⨾} \mathsf{CIOPS})(\mathsf{as}, \mathsf{cs})) \\
&\qquad \vee\, \mathsf{OR}(\mathsf{as}, \mathsf{el_A}, \mathsf{final_A}, \mathsf{cs}, \mathsf{el_C}, \mathsf{final_C})
\end{aligned}$$

The disjunction with OR is needed for those runs, that do not apply any external operations. Now, executing an internal step in cs adds a 0:1 diagram. To show that "decrease" implies that the resulting state cs' is less than cs (using the given order $<$ as $<_{\mathsf{0n}}$), a state as' with $\mathsf{R}(\mathsf{as}', \mathsf{cs})$ is found via completing one commuting diagram of weak data refinement. But note, that this diagram and the resulting state as' will *not* be used to construct the corresponding trace. Again, when (6) is false execution of more internal operations may require to choose a different abstract run. as' is only used in the other case, when

$$\begin{tikzcd}
\mathsf{as} \arrow[r, "\mathsf{AIOPS}"] \arrow[d, dash, "\mathsf{id}"] & {} \arrow[r, "\mathsf{AOP_e}"] & {} \arrow[r, "\mathsf{AIOPS}"] & {} \arrow[d, dash, "\mathsf{id}"] \\
\mathsf{as}^* \arrow[rrr, "\mathsf{AOP^*_e}"] \arrow[d, dash, "\mathsf{R}"] & & & {} \arrow[d, dash, "\mathsf{R}"] \\
\mathsf{cs}^* \arrow[rrr, "\mathsf{COP^*_e}"] \arrow[d, dash, "\mathsf{INV}"'] \arrow[dr, dash, "\mathsf{INV}"] & & & {} \arrow[d, dash, "\mathsf{INV}"] \arrow[dl, dash, "\mathsf{INV}"'] \\
\mathsf{cs} \arrow[r, "\mathsf{CIOPS}"'] & {} \arrow[r, "\mathsf{COP_e}"'] & {} \arrow[r, "\mathsf{CIOPS}"'] & {}
\end{tikzcd}$$

Fig. 5. Three ASM refinements that simulate weak data refinement

CDT executes the next *external* step from cs to cs' to complete a n:1 diagram. Then as' will be the corresponding state for cs'. Again, as' is then one external operation behind cs'. The details of this proof are quite elaborate, and it is the only one we found so far that requires the general "init" condition of ASM refinement (instead of just $\mathsf{IR} \subseteq \mathsf{INV}$).

The refinement can also be verified another way that is even sketched in [20]: weak refinement can be viewed as ordinary data refinement (with external operations only) of an abstract data type $\mathsf{ADT}*$ with operations

$$\mathsf{AOP^*_e} := \mathsf{AIOPS} \mathbin{;} \mathsf{AOP_e} \mathbin{;} \mathsf{AIOPS} \tag{9}$$

to an analogously defined (conformal) data type $\mathsf{CDT}*$. We can formalize this, by looking at the corresponding ASMs $\mathsf{BEM}^\omega(\mathsf{ADT}*)$ and $\mathsf{BEM}^\omega(\mathsf{CDT}*)$ [4]. This means the verification can be done as a composition of the three ASM refinements from $\mathsf{WM}(\mathsf{ADT})$ via $\mathsf{BEM}^\omega(\mathsf{ADT}*)$ and $\mathsf{BEM}^\omega(\mathsf{CDT}*)$ to $\mathsf{WM}(\mathsf{CDT}*)$, as shown in Figure 5.

The first refinement is a trivial ASM refinement with n:1 diagrams and identity as coupling invariant. The second refinement is standard data refinement. The only difficult refinement is the one from $\mathsf{BEM}^\omega(\mathsf{CDT}*)$ to $\mathsf{WM}(\mathsf{CDT}*)$. The coupling invariant for this case is

$$\mathsf{INV}(\mathsf{cs}^*, \mathsf{cs}) :\leftrightarrow \mathsf{CIOPS}(\mathsf{cs}^*, \mathsf{cs}) \wedge \mathsf{reachable}(\mathsf{cs})$$

i.e. $\mathsf{WM}(\mathsf{CDT}*)$ may have done some more internal operations. We get an m:1 diagram for the case, where CDT executes an external operation, and 0:1

[4] The ASMs used here differ slightly from the ones of Section 5: like WM they must apply the finalization operation at the end of each run.

diagrams, when CDT executes an internal operation. The reachability condition $\mathsf{reachable}(\text{cs})$, defined as $\exists\ \mathsf{gs} \in \mathsf{G}, \mathsf{js} \in (\mathsf{E} \cup \mathsf{J})^*.\ (\mathsf{CINIT} \mathbin{;} \mathsf{COP}_{\mathsf{js}})(\mathsf{gs}, \mathsf{cs})$ is needed to prove, that infinitely many 0:1 diagrams are impossible. "decrease" is applicable, since the lemma

$$\mathsf{reachable}(\mathsf{cs}) \rightarrow \exists\ \mathsf{as}.\ \mathsf{R}(\mathsf{as}, \mathsf{cs})$$

can be proved by wellfounded induction on the number of steps executed by CDT. Note that in these refinements, no embedding is used. All operations may be applied only, when the current state is in their domain.

Using the behavioral or contract embedding is possible for weak refinement, but the embedding should not be used for internal operations, as [20] motivates from a process algebraic view. A suitable embedding for an internal operation IOP is therefore

$$\widehat{\widehat{\mathsf{IOP}}} = \mathsf{IOP} \cup \{(\bot, \bot)\} \tag{10}$$

With this embedding an internal operation is never applied outside its domain, and programs that diverge (or block) may be continued with internal operations (still remaining in that state). This embedding commutes with forming finite sequences, i.e.

$$\mathsf{C}\widehat{\widehat{\mathsf{IOP}}}\mathsf{S} = (\textstyle\bigcup_{\mathsf{j} \in \mathsf{J}} \widehat{\widehat{\mathsf{COP}}}_{\mathsf{j}})^*$$

holds. We use this embedding for internal operations in both the contract as well as the behavioral approach of refinement. External operations EOP still use $\widehat{\mathsf{EOP}}$ and $\widetilde{\mathsf{EOP}}$ as defined in Section 4.

To define the correctness conditions we need the notion of the strong domain $\mathsf{sdom}(\mathsf{COP})$ of an external operation COP. The strong domain consists of those states cs, that do not allow to choose any sequence of internal operations, such that COP diverges (or blocks) in the resulting state cs'. Formally:

$$\mathsf{cs} \in \mathsf{sdom}(\mathsf{COP}) :\leftrightarrow \neg \exists\ \mathsf{cs}'.\ \mathsf{CIOPS}(\mathsf{cs}, \mathsf{cs}') \wedge \mathsf{cs}' \notin \mathsf{dom}(\mathsf{COP})$$

$\mathsf{sdom}(\mathsf{COP})$ is always a subset of $\mathsf{dom}(\mathsf{CIOPS} \mathbin{;} \mathsf{COP})$, in general a proper subset. With this definition we get:

Theorem 15 (Weak forward simulation (behavioral and contract)) *A weak data refinement from* ADT *to* CDT *is correct when using the behavioral embedding if a relation* R *and a wellfounded order* $<$ *on* CS *can be found such that*

- $\mathsf{CINIT} \mathbin{;} \mathsf{CIOPS} \subseteq \mathsf{AINIT} \mathbin{;} \mathsf{AIOPS} \mathbin{;} \mathsf{R}$ *("initialization")*

- $\forall\, \mathsf{i} \in \mathsf{E}.\quad (\mathsf{R} \fatsemi \mathsf{CIOPS} \fatsemi \mathsf{COP_i} \fatsemi \mathsf{CIOPS})$
  $\subseteq \mathsf{AIOPS} \fatsemi \mathsf{AOP_i} \fatsemi \mathsf{AIOPS} \fatsemi \mathsf{R}$ *("correctness")*
- $\forall\, \mathsf{i} \in \mathsf{E}.\ \mathsf{ran}(\mathsf{sdom}(\mathsf{AOP_i}) \lhd \mathsf{R}) \subseteq \mathsf{sdom}(\mathsf{COP_i})$ *("applicability")*
- $\mathsf{R} \fatsemi \mathsf{CIOPS} \fatsemi \mathsf{CFIN} \subseteq \mathsf{AIOPS} \fatsemi \mathsf{AFIN}$ *("finalization")*
- $\forall\, \mathsf{j} \in \mathsf{J}.\ \mathsf{R}(\mathsf{as}, \mathsf{cs}) \wedge \mathsf{COP_j}(\mathsf{cs}, \mathsf{cs}') \rightarrow \mathsf{cs}' < \mathsf{cs}$ *("decrease")*

For the contract approach the "correctness" condition can be weakened to

$$\begin{aligned}\forall\, \mathsf{i} \in \mathsf{E}.\quad & (\mathsf{sdom}(\mathsf{AOP_i}) \lhd \mathsf{R}) \fatsemi \mathsf{CIOPS} \fatsemi \mathsf{COP_i} \fatsemi \mathsf{CIOPS} \\ & \subseteq \mathsf{AIOPS} \fatsemi \mathsf{AOP_i} \fatsemi \mathsf{AIOPS} \fatsemi \mathsf{R}\end{aligned}$$

The theorem was proved formally using KIV. The proof is done as usual by removing $\perp$ from the conditions resulting from the embedding. Again the "decrease" condition is not necessary in the proof (it is even incompatible with the embedding (10), since $\perp < \perp$ would violate wellfoundedness). Both the "correctness" as well as the "applicability" condition given here differ from the ones given in [20] for the contract approach [5], which are:

$$\begin{aligned}\forall\, \mathsf{i} \in \mathsf{I}.\quad & (\mathsf{dom}(\mathsf{AIOPS} \fatsemi \mathsf{AOP_i}) \lhd \mathsf{R}) \fatsemi \mathsf{CIOPS} \fatsemi \mathsf{COP_i} \fatsemi \mathsf{CIOPS} \\ & \subseteq \mathsf{AIOPS} \fatsemi \mathsf{AOP_i} \fatsemi \mathsf{AIOPS} \fatsemi \mathsf{R}\ (\text{“correctness”}) \\ \forall\, \mathsf{i} \in \mathsf{I}.\ & \mathsf{ran}(\mathsf{dom}(\mathsf{AIOPS} \fatsemi \mathsf{AOP_i}) \lhd \mathsf{R}) \\ & \subseteq \mathsf{dom}(\mathsf{CIOPS} \fatsemi \mathsf{COP_i})\ (\text{“applicability”})\end{aligned} \tag{11}$$

These conditions are the ones that would result from a refinement of $\widetilde{\mathsf{AOP}}^*_{\mathsf{e}}$ ($\mathsf{AOP}^*_{\mathsf{e}}$ as defined in (9)) to $\widetilde{\mathsf{COP}}^*_{\mathsf{e}}$. But the semantics of programs is different, since the equations

$$\begin{aligned}\widetilde{\mathsf{AOP}}^*_{\mathsf{e}} &= \mathsf{A}\widehat{\widehat{\mathsf{IOP}}}\mathsf{S} \fatsemi \widetilde{\mathsf{AOP}}_{\mathsf{e}} \fatsemi \mathsf{A}\widehat{\widehat{\mathsf{IOP}}}\mathsf{S} \\ \widehat{\mathsf{AOP}}^*_{\mathsf{e}} &= \mathsf{A}\widehat{\widehat{\mathsf{IOP}}}\mathsf{S} \fatsemi \widehat{\mathsf{AOP}}_{\mathsf{e}} \fatsemi \mathsf{A}\widehat{\widehat{\mathsf{IOP}}}\mathsf{S}\end{aligned}$$

do not hold in general, and the right hand side is actually used (first line for the contract, second line for the behavioral approach). The difference between the conditions allows to define the following counter example:

Example 1 $\mathsf{ADT} := (\mathsf{G}, \mathsf{AS}, \mathsf{INIT}, \{\mathsf{AOP_1}\}, \mathsf{AFIN})$,
$\mathsf{CDT} = (\mathsf{G}, \mathsf{CS}, \mathsf{INIT}, \{\mathsf{COP_1}, \mathsf{COP_2}\}, \mathsf{CFIN})$,
$\mathsf{AOP_1}, \mathsf{COP_1}$ *are external,* $\mathsf{COP_2}$ *is internal.*
$\mathsf{G} = \{*\}$, $\mathsf{AS} = \{1, 2\}$, $\mathsf{INIT} = \{(*, 1)\}$, $\mathsf{AOP_1} = \{(1, 2)\}$,
$\mathsf{CS} = \{1, 1\mathsf{b}, 2\}$, $\mathsf{COP_2} = \{(1, 1\mathsf{b})\}$, $\mathsf{COP_1} = \{(1\mathsf{b}, 2)\}$.

For both the behavioral as well as the contract embedding the concrete data

[5] the conditions are given for Z refinement, which (when input and output is not considered) is basically the contract approach using G = {*}. For this case the finalization operation must be $\mathsf{FIN} = \mathsf{S} \times \mathsf{G}$, and the "finalization" condition is trivial.

type has a run to $\bot$ *by applying* COP_1 *immediately. The abstract data type has no corresponding run, so the weak refinement is not correct. Conditions (11) are true for* $\mathsf{R} = \{(1,1),(1,1b),(2,2)\}$*. The problem is that the "applicability" condition is satisfied, since* $1 \in \mathsf{dom}(\mathsf{AIOPS} \mathbin{⨟} \mathsf{AOP}_1)$*. The "applicability" condition of the theorem prohibits the example above, since* $1 \notin \mathsf{sdom}(\mathsf{COP}_1)$ *.*

The "correctness" condition of the theorem is strictly weaker than the one in (11). It allows to verify the following weak refinement for the contract approach:

Example 2 $\mathsf{ADT} = (\mathsf{G},\mathsf{S},\mathsf{INIT},\{\mathsf{OP}_1,\mathsf{AOP}_2\},\mathsf{FIN})$,
$\mathsf{ADT} = (\mathsf{G},\mathsf{S},\mathsf{INIT},\{\mathsf{OP}_1,\mathsf{COP}_2\},\mathsf{FIN})$,
where OP_1 *is internal,* AOP_2 *and* COP_2 *are external.*
$\mathsf{G} = \{*\}$, $\mathsf{S} = \{1,2,3,4\}$, $\mathsf{INIT} = \{(*,1)\}$,
$\mathsf{OP}_1 = \{(1,2),(1,3)\}$, $\mathsf{AOP}_2 = \{(3,4)\}$, $\mathsf{AOP}_2 = \{(3,4),(2,4)\}$.

The refinement is a correct weak refinement, since the additional run of CDT *through* $1,2,4$ *refines the diverging run* $1,2,\bot$*. The "correctness" condition of (11) is violated:* $1 \in \mathsf{dom}(\mathsf{AIOPS} \mathbin{⨟} \mathsf{AOP}_2)$ *implies that the pair* $(1,4)$ *violates the subset condition. Since* $1 \notin \mathsf{sdom}(\mathsf{AOP}_2)$*, the "correctness" condition of the theorem is satisfied.*

An ASM which corresponds to weak behavioral refinement can be defined too. ASM $\mathsf{WBM(DT)}$ with rule $\mathsf{WBRULE(DT)}$ chooses randomly between external and applicable internal operations. Flag b is true when an internal operation is chosen (otherwise the index i is ignored):

```
WBRULE(DT) := if $\neg$ final then
        choose $b \in Bool, i \in I$ with $b \rightarrow st \in dom(OP_i)$
        in if b then $TRULE_i$
              else if $el \neq []$
                    then if $st \notin dom(OP_{head(il)})$ then final := true
                              $BRULE_{head(il)}$
                              el := tail(el)
                     else BFINRULE
                              final := true
```

Again, even if finite input $\mathsf{el} \in \mathsf{E}^*$ is considered only, this ASM has infinite runs, so verification of the ASM refinement really needs the "decrease" condition. Replacing $\mathsf{BRULE}_{\mathsf{head(il)}}$ by $\mathsf{DRULE}_{\mathsf{head(il)}}$ gives $\mathsf{WDM(DT)}$. Since this ASM diverges when $\mathsf{st} \notin \mathsf{dom}(\mathsf{OP}_{\mathsf{head(il)}})$, the assignment $\mathsf{final} := \mathsf{true}$ for this case can be removed from the code.

Theorem 16 (Weak forward simulation (beh.) implies ASM ref.) *If the refinement from* ADT *to* CDT *is a correct weak data refinement for the behavioral approach, and if* IR *as defined in (7) satisfies (8), then both the refinements of* $\mathsf{WBM(ADT)}$ *to* $\mathsf{WBM(CDT)}$ *and of* $\mathsf{WDM(ADT)}$ *to* $\mathsf{WDM(CDT)}$

are correct ASM refinements that totally preserves traces for R *and total correctness for*

$$\begin{aligned}\mathsf{OR} := \quad & \mathsf{el_A} = \mathsf{el_C} \wedge \mathsf{final_A} \wedge \mathsf{final_C} \wedge (\mathsf{el_A} = [] \rightarrow \mathsf{as} = \mathsf{cs} \in \mathsf{G}) \\ & \wedge (\mathsf{el_A} \neq [] \rightarrow \mathsf{cs} \notin \mathsf{dom}(\mathsf{COP_{head(el)}}) \wedge \mathsf{as} \notin \mathsf{dom}(\mathsf{AOP_{head(el)}}))\end{aligned}$$

in the first case, and for $\mathsf{OR} := \mathsf{as} = \mathsf{cs} \in \mathsf{G} \wedge \mathsf{final_A} \wedge \mathsf{final_C} \wedge \mathsf{el_A} = \mathsf{el_C} = []$ *in the second.*

Relation OR distinguishes two cases for final states: either both ASMs have regularly terminated after executing all required external operations (case $\mathsf{el_A} = []$), and the finalization operations have computed the same result. Or both ASMs have stopped in states when an attempt was made to apply an external operation outside of its domain (case $\mathsf{el_A} \neq []$). For the refinement from $\mathsf{WDM(ADT)}$ to $\mathsf{WDM(CDT)}$ the second second case is not needed in OR, since it is implied by the fact that diverging runs must implement diverging runs. The refinement proofs for WBM and WDM in KIV are very similar to the ones for WM, except that a few extra cases must be considered for blocking and divergence.

9 Coupled Refinement implies ASM Refinement

Coupled refinement, defined in [16], is a restricted case of non-atomic refinement, defined between two data types $\mathsf{ADT} = (\mathsf{G}, \mathsf{AS}, \mathsf{AINIT}, \{\mathsf{AOP_i}\}_{\mathsf{i} \in \mathsf{I}}, \mathsf{AFIN})$ and $\mathsf{CDT} = (\mathsf{G}, \mathsf{CS}, \mathsf{CINIT}, \{\mathsf{COP_i}\}_{\mathsf{j} \in \mathsf{J}}, \mathsf{CFIN})$. The general idea of non-atomic refinement is to implement each operation $\mathsf{AOP_i}$ of an abstract data type by a sequence $\mathsf{COP}_{\rho(\mathsf{i})}$. The indices are given by a total function $\rho : \mathsf{I} \rightarrow \mathsf{J}^+$ ($\rho(\mathsf{i}) = []$ is not allowed in coupled refinement).

Non-atomic refinement replaces the correctness condition of ordinary data refinement with the requirement, that all 1:n diagrams with $\mathsf{AOP_i}$ and $\mathsf{COP}_{\rho(\mathsf{i})}$ commute. To enforce, that the concrete data type executes only groups of operations with indices $\rho(\mathsf{i})$ the behavioral approach is used: if e.g. $\rho(1) = [1, 2, 3]$ then a run starting with $\mathsf{COP_2}$ or $\mathsf{COP_3}$ is prevented, by assuming initial states are not in $\mathsf{dom}(\mathsf{COP_2})$ or $\mathsf{dom}(\mathsf{COP_3})$. Also, all states after execution of $\mathsf{COP_1}$ should be in $\mathsf{dom}(\mathsf{COP_2})$, but not in $\mathsf{dom}(\mathsf{COP_3})$. If several abstract operations are refined, the sequences of implementing operations are required to be disjoint, i.e. $\rho(\mathsf{i}) \cap \rho(\mathsf{j}) = \emptyset$ for $\mathsf{i} \neq \mathsf{j}$. Coupled refinement always allows to execute a sequence of operations $\mathsf{COP}_{\rho(\mathsf{i_1})} \fatsemi \mathsf{COP}_{\rho(\mathsf{i_2})} \fatsemi \ldots \fatsemi \mathsf{COP}_{\rho(\mathsf{i_k})}$ (thereby constructing commuting 1:n diagrams). But it also *may* allow to interleave diagrams: if e.g. $\rho(1) = [1, 2, 3]$ and $\rho(2) = [4, 5]$ then a run executing operations with indices $[1, 2, 4, 3, 5]$ is acceptable, if and only if the state reached after executing $\mathsf{COP_1} \fatsemi \mathsf{COP_2}$ is in $\mathsf{dom}(\mathsf{COP_4})$. In this case, execution of $\mathsf{COP_4}$ must preserve the applicability of COP_3. Autoconcurrency, i.e. interleaving a diagram with itself is not allowed, e.g. the sequence $[1, 1]$ is forbidden.

To define verification conditions, different invariants R^{js} for $\mathsf{js} \in \mathsf{IL}(\rho)$, a subset of J^* are defined. The idea is, that initially $\mathsf{R}^{[]} := \mathsf{R}$ should hold. Executing COP_4 gives a 0:1 diagram, where $\mathsf{R}^{[4]}$ is required at the end. Subsequent execution of COP_5 together with AOP_2 gives a 1:1 diagram which should lead back to $\mathsf{R}^{[]}$. If the two diagrams of the example may be interleaved, then instead of COP_5, a 0:1 diagram may be added by applying COP_1, leading to $\mathsf{R}^{[4,1]}$. Executing COP_5 together with AOP_2 then leads to $\mathsf{R}^{[1]}$. This means that $\mathsf{IL}(\rho)$ has to consist of all sequences of events, that are interleavings of proper prefixes of the sequences $\rho(\mathsf{i})$. For the example where $\rho(1) = [1, 2, 3]$ and $\rho(2) = [4, 5]$ we have

$$\mathsf{IL}(\rho) = \{[], [1], [4], [1, 2], [1, 4], [4, 1], [4, 1, 2], [1, 4, 2], [1, 2, 4]\}$$

For $\mathsf{js} \in \mathsf{IL}(\rho)$ we define $\mathsf{open}(\mathsf{js})$ to be those indices j_1, that open a diagram: there must be an i with $\rho(\mathsf{i}) = [\mathsf{j}_1, \ldots \mathsf{j}_n]$ such that $\mathsf{js} + [\mathsf{j}_1]$ is still in $\mathsf{IL}(\rho)$. In our example $\mathsf{open}([]) = \{1, 4\}$, $\mathsf{open}([4]) = \{1\}$ and $\mathsf{open}([1, 4]) = \{\}$. $\mathsf{cont}(\mathsf{js})$ contains those indices that continue (but do not close) an already open diagram: j is in $\mathsf{cont}(\mathsf{js})$, if it is not in $\mathsf{open}(\mathsf{js})$ and if $\mathsf{js} + [\mathsf{j}]$ is still in $\mathsf{IL}(\rho)$. In the example $\mathsf{cont}([]) = \mathsf{cont}([4]) = \{\}$, and $\mathsf{cont}([1]) = \{2\}$. For $\mathsf{j} \in (\mathsf{open}(\mathsf{js}) \cup \mathsf{cont}(\mathsf{js}))$ we set $\mathsf{js} \oplus \mathsf{j} := \mathsf{js} + [\mathsf{j}]$. $\mathsf{close}(\mathsf{js})$ is the set of pairs $(\mathsf{i}, \mathsf{j}_n)$, such that j_n is a possible next step after js that completes a diagram with the abstract operation i. For $\rho(\mathsf{i}) = [\mathsf{j}_1, \ldots \mathsf{j}_n]$, this means that $(\mathsf{i}, \mathsf{j}_n)$ is in $\mathsf{close}(\mathsf{js})$, if $[\mathsf{j}_1, \ldots \mathsf{j}_{n-1}]$ is homoeomorphically embedded in js. For the example this means $\mathsf{close}([1]) = \emptyset$, $\mathsf{close}([1, 4]) = \{(2, 5)\}$ and $\mathsf{close}([4, 1, 2]) = \{(2, 5), (1, 3)\}$. For $(\mathsf{i}, \mathsf{j}_n) \in \mathsf{close}(\mathsf{js})$ $\mathsf{js} \oplus \mathsf{j}_n$ is the result of removing the subsequence $[\mathsf{j}_1, \ldots \mathsf{j}_{n-1}]$ from js. For the example the definition of $\oplus$ implies $[1] \oplus 4 = [1, 4]$ (a step in open), $[1, 4] \oplus 2 = [1, 4, 2]$ (a step in cont), $[1, 2, 4] \oplus 5 = [1, 2]$ and $[1, 2] \oplus 3 = []$ (two steps in close). Finally we set

$$\mathsf{next}(\mathsf{js}) = \mathsf{open}(\mathsf{js}) \cup \mathsf{cont}(\mathsf{js}) \cup \{\mathsf{j} : (\mathsf{i}, \mathsf{j}) \in \mathsf{close}(\mathsf{js})\}$$

to be all operations that can be applied next. With these definitions the correctness conditions for coupled refinement are:

Definition 9 (Correctness conditions for coupled refinement) *A coupled refinement is correct, if in addition to the simulation* R *additional simulations* R^{js} *can be found for every* $\mathsf{js} \in \mathsf{IL}(\rho)$ *such that*

- $\mathsf{CINIT} \subseteq \mathsf{AINIT} \mathbin{⨾} \mathsf{R}$ *("initialization")*
- $\mathsf{R} \mathbin{⨾} \mathsf{CFIN} \subseteq \mathsf{AFIN}$ *("finalization")*
- $\mathsf{R} = \mathsf{R}^{[]}$ *("coupling")*
- $\mathsf{R}^{\mathsf{js}}(\mathsf{as}, \mathsf{cs}) \wedge \mathsf{j} \in \mathsf{open}(\mathsf{js}) \wedge \mathsf{cs} \in \mathsf{dom}(\mathsf{COP}_\mathsf{j}) \wedge \mathsf{COP}_\mathsf{j}(\mathsf{cs}, \mathsf{cs}') \rightarrow \mathsf{R}^{\mathsf{js} \oplus \mathsf{j}}(\mathsf{as}, \mathsf{cs}')$ *("open diagram")*

- $\mathsf{R}^{\mathsf{js}}(\mathsf{as},\mathsf{cs}) \wedge \mathsf{j} \in \mathsf{cont}(\mathsf{js}) \rightarrow \mathsf{cs} \in \mathsf{dom}(\mathsf{COP}_\mathsf{j})$ *("continue applicable")*
- $\mathsf{R}^{\mathsf{js}}(\mathsf{as},\mathsf{cs}) \wedge \mathsf{j} \in \mathsf{cont}(\mathsf{js}) \wedge \mathsf{COP}_\mathsf{j}(\mathsf{cs},\mathsf{cs}') \rightarrow \mathsf{R}^{\mathsf{js}\oplus\mathsf{j}}(\mathsf{as},\mathsf{cs}')$ *("continue diagram")*
- $\mathsf{R}^{\mathsf{js}}(\mathsf{as},\mathsf{cs}) \wedge (\mathsf{i},\mathsf{j}) \in \mathsf{close}(\mathsf{js}) \rightarrow \mathsf{cs} \in \mathsf{dom}(\mathsf{COP}_\mathsf{j})$ *("close applicable")*
- $\mathsf{R}^{\mathsf{js}}(\mathsf{as},\mathsf{cs}) \wedge (\mathsf{i},\mathsf{j}) \in \mathsf{close}(\mathsf{js}) \rightarrow \mathsf{R}^{\mathsf{js}} \mathbin{;} \mathsf{COP}_\mathsf{j}(\mathsf{cs},\mathsf{cs}') \subseteq \mathsf{AOP}_\mathsf{i} \mathbin{;} \mathsf{R}^{\mathsf{js}\oplus\mathsf{j}}$ *("close diagram")*
- $\mathsf{R}^{\mathsf{js}}(\mathsf{as},\mathsf{cs}) \wedge \mathsf{j} \notin \mathsf{next}(\mathsf{js}) \rightarrow \mathsf{cs} \notin \mathsf{dom}(\mathsf{COP}_\mathsf{j})$ *("not applicable")*

The idea of the relation $\mathsf{R}^{\mathsf{js}}(\mathsf{as},\mathsf{cs})$ is to say: "Before operations $\mathsf{COP}_{\mathsf{js}}$ were executed, the concrete states were corresponding via R". This is equivalent to $(\mathsf{R} \mathbin{;} \mathsf{COP}_{\mathsf{js}})(\mathsf{as},\mathsf{cs})$ (note that for the case, when the sequence $\mathsf{COP}_{\mathsf{js}}$ is not applicable, $\mathsf{R} \mathbin{;} \mathsf{COP}_{\mathsf{js}}$ is false). The "open diagram" condition does not enforce, that $\mathsf{cs} \in \mathsf{dom}(\mathsf{COP}_\mathsf{j})$ for $\mathsf{j} \in \mathsf{open}(\mathsf{js})$, so an operation that opens a new diagram may or may not be applicable in a state with R^{js}. On the other hand, "continue applicable" and "close applicable" ensure, that it is always possible, to continue and close any diagram, that is already open.

To view this refinement as an ASM refinement, it is not possible to allow blocking or divergence as in the behavioral or contract approach. Otherwise the concrete data type may try to apply COP_2 in an initial state, leading to $\perp$, and the abstract data type may not have a corresponding run. Instead, $\mathsf{st} \notin \mathsf{dom}(\mathsf{OP})$ is interpreted as the rule being not applicable, like for internal operations of weak refinement. Therefore the ASMs $\mathsf{BIM}^\alpha(\mathsf{DT})$ are a suitable choice for the interpretation of coupled refinement. The most interesting ASM among these is $\mathsf{BIM}^\infty(\mathsf{DT})$ since it always has a chance to finish open diagrams. Its rule is

$$\begin{aligned}\mathsf{BIRULE}^\infty(\mathsf{DT}) := {} & \textbf{if } \exists\, \mathsf{i}.\ \mathsf{st} \in \mathsf{dom}(\mathsf{OP}_\mathsf{i}) \\ & \textbf{then choose } \mathsf{i} \in \mathsf{I} \textbf{ with } \mathsf{st} \in \mathsf{dom}(\mathsf{OP}_\mathsf{i}) \textbf{ in } \mathsf{BRULE}_\mathsf{i}\end{aligned}$$

For these ASMs it can be proved:

Theorem 17 (Coupled Refinement implies ASM refinement)
If the refinement from ADT to CDT is a coupled refinement, then the ASM refinements of $\mathsf{BIM}^\alpha(\mathsf{ADT})$ to $\mathsf{BIM}^\alpha(\mathsf{CDT})$, where $\alpha \in \{*, \infty, \omega\}$, totally preserve traces with respect to the relation

$$\mathsf{INV}(\mathsf{as},\mathsf{cs}) := \exists\, \mathsf{js} \in \mathsf{IL}(\rho).\ (\mathsf{R} \mathbin{;} \mathsf{COP}_{\mathsf{js}})(\mathsf{as},\mathsf{cs})$$

For $\alpha = \infty$ the refinement preserves total correctness for the stronger relations

$$\mathsf{IR} := \mathsf{R} \cap (\mathsf{AIN} \times \mathsf{CIN}),\ \mathsf{OR} := \mathsf{R} \cap (\mathsf{AOUT} \times \mathsf{COUT})$$

The proof is rather simple, since ASM refinement allows the 1:1 diagrams in "close diagram" as well as the 0:1 diagrams in the conditions "continue diagram" and "close diagram". The number of subsequent 0:1 diagrams is finite, since the sequence of executed concrete operations increases, and is bounded by the longest sequence in the finite set $\mathsf{IL}(\rho)$. The fact that diagrams may be interleaved prevents the use of larger diagrams. If the set $\mathsf{IL}(\rho)$ is defined to contain prefixes of $\rho(\mathsf{i})$ for *one* i only, the refinement of $\mathsf{AOP_i}$ to $\mathsf{COP}_{\rho(\mathsf{i})}$ can be verified as one $1 : \mathsf{length(js)}$ diagram. The refinement of $\mathsf{BIM}^\infty(\mathsf{ADT})$ to $\mathsf{BIM}^\infty(\mathsf{CDT})$ allows to set $\mathsf{OR} := \mathsf{R} \cap (\mathsf{AOUT} \times \mathsf{COUT})$ instead of $\mathsf{OR} := \mathsf{INV} \cap (\mathsf{AOUT} \times \mathsf{COUT})$, since the only potentially final states are those where $\mathsf{R}^{[]}$ holds. If $\mathsf{js} \neq []$, then at least one of $\mathsf{cont(js)}$ or $\mathsf{close(js)}$ is nonempty, so an operation must be applicable according to "continue applicable" and "close applicable".

The proof of the theorem shows, that the ASM refinement is still correct, if the conditions "continue applicable" and "close applicable" are weakened to

$$\mathsf{R^{js}(as, cs)} \rightarrow \exists\ \mathsf{j}.\ (\mathsf{j} \in \mathsf{cont(js)} \vee \exists\ \mathsf{i}.\ (\mathsf{i},\mathsf{j}) \in \mathsf{close(js)}) \wedge \mathsf{cs} \in \mathsf{dom(COP_j)}$$
("applicable")

It is sufficient, that *some* operation continuing one of the currently open diagrams is applicable instead of *all* such operations. For the example, this means that after $\mathsf{COP_1} \mathbin{;} \mathsf{COP_4}$ have been executed, it is sufficient, that one of $\mathsf{COP_2}$ or $\mathsf{COP_5}$ is applicable, instead of both. This still allows to find some continuation that closes all open diagrams.

The theorem shows that a coupled refinement can be viewed as an ASM refinement. But coupled refinement implies a slightly stronger condition, than the ASM refinement: coupled refinement implies, that from each pair of states with $\mathsf{R^{js}(as, cs)}$ states $\mathsf{as'}$ and $\mathsf{cs'}$ can be reached by completing all open diagrams, such that $\mathsf{R(as', cs')}$ holds again. The "close diagram" condition is responsible for this property. ASM refinement only proves the weaker condition

$$\mathsf{R^{js}(as, cs)} \wedge (\mathsf{i},\mathsf{j}) \in \mathsf{finals(js)} \rightarrow \mathsf{R^{js}} \mathbin{;} \mathsf{COP_j} \subseteq \mathsf{AOP_i} \mathbin{;} (\exists\ \mathsf{js'}.\ \mathsf{R^{js'}}) \qquad (12)$$

Now in applications, as the ones given in [16] and [33] the invariant R guarantees that $\mathsf{js'}$ in (12) can be chosen to be $\mathsf{js} \oplus \mathsf{j}$, but theoretically, the ASM refinement can choose $\mathsf{R^{js'}}$ with $\mathsf{js'} \neq \mathsf{js} \oplus \mathsf{j}$. To ensure that an ASM refinement with an arbitrary R implies the "close diagram" condition it is necessary to strengthen INV. One attempt, that can be shown to be an ASM refinement, is to add the condition "executing any $\mathsf{COP_j}$ will lead to a state where exactly the operations in $\mathsf{nexts(js \oplus j)}$ are applicable". This refinement guarantees that $\mathsf{js'}$ is a permutation of $\mathsf{js} \oplus \mathsf{j}$, so it implies that all open diagrams can be closed. Still, when e.g. $\rho(1) = [1, 2]$, $\rho(2) = [3, 4]$ and $\rho(3) = [5, 6]$, this coupling invariant allows, that executing $\mathsf{COP_2}$ and $\mathsf{AOP_1}$ in a state with $\mathsf{R}^{[1,3,5]}$ leads to $\mathsf{R}^{[5,3]}$ instead of $\mathsf{R}^{[3,5]}$. It seems, there is no other choice for a stronger coupling invariant that implies $\mathsf{js'} = \mathsf{js} \oplus \mathsf{j}$ in general, than to add the "close diagram" condition itself to INV.

$$\begin{aligned}
&\mathsf{INV}(\mathsf{as},\mathsf{cs}) := \\
&\quad \exists\ \mathsf{js} \in \mathsf{IL}(\rho).\quad (\mathsf{R} \circ\!\!\!\!\, ; \mathsf{COP}_{\mathsf{js}})(\mathsf{as},\mathsf{cs}) \\
&\qquad\qquad\qquad \wedge\ \forall\ \mathsf{js},\mathsf{cs},\mathsf{cs}',\mathsf{i},\mathsf{j}. \\
&\qquad\qquad\qquad\qquad (\mathsf{i},\mathsf{j}) \in \mathsf{finals}(\mathsf{js}) \\
&\qquad\qquad\qquad\quad \rightarrow \mathsf{R}^{\mathsf{js}} \circ\!\!\!\!\, ; \mathsf{COP}_{\mathsf{j}} \subseteq \mathsf{AOP}_{\mathsf{i}} \circ\!\!\!\!\, ; \mathsf{R}^{\mathsf{js}\oplus\mathsf{j}}
\end{aligned} \tag{13}$$

Now, already in the proof of the "establish invariant" condition of ASM refinement (the stronger version $\mathsf{IR} \subseteq \mathsf{INV}$ is sufficient) the "close diagram" condition has to be proved. It is then used in the 1:1 diagram to prove that js' can indeed be chosen as $\mathsf{js} \oplus \mathsf{j}$. Each of the 0:1 diagrams also repeats the proof of "close diagram". Note that this (admittedly ugly) trick is of theoretical interest only: it shows, that a special case of ASM refinement is definable that *exactly* has the verification conditions of coupled refinement.

Finally, it should be noted, that ASM refinement supports the general case of a relation on the index set (p.289 of [20]), or a mapping that needs to map one abstract operation to a bounded number of operations as in the example in [33], where setting the minutes on a clock to some value n between 0 and 59 is implemented by pushing a button, that increments from 0 n times. The relation may even depend on the size of data structures stored in the current states. Such cases, where the sizes m and n of m:n diagrams were dependent on the current states, were present in the verification of the Prolog compiler, that was cited in the introduction. The reason, that ASM refinement can handle such cases, is the fact, that ASM refinement is *not* required to keep a predefined relation between the indices of operations. As can be seen from our informal definitions of $\mathsf{IL}(\rho)$, cont etc. above, the request, that a refinement must keep a specific relation between indices, complicates the definition of verification conditions enormously. Fixing a concrete relation runs the risk of disallowing meaningful refinements. So although the explanatory value of commuting diagrams which show a specific correspondence is not doubted, it is easier not to prescribe a specific relation in the definition of refinement correctness.

10 Conclusion

In this paper we have analyzed the relation between ASM refinement and data refinement. The analysis had to bridge the gap between two rather distinct specification styles: ASMs have algebras as states and specify transitions operationally using function updates. Like TLA refinement [19] and refinement of IO automata [11] the semantics of ASMs is based on a notion of transition system. The environment is included in the state as so-called external functions.

Data refinement, on the other hand specifies operations using relations. The environment is modeled separately using a global state together with operations for initialization and finalization. Partial operations can be interpreted either as preconditions (contract approach) or as guards (behavioral approach). Data refinement considers finite sequences of operations only.

Despite all these technical differences, both formalisms have in common, that their refinement notions are based on commuting diagrams. We could prove that by mapping abstract data types to ASMs with external control and finite input, behavioral data refinement becomes a specific instance of ASM refinement.

Verification of ASM refinement is based on generalized forward simulation which uses arbitrary shaped m:n diagrams mapping m abstract operations to n concrete ones. The idea of data refinement is to implement each abstract operation by a corresponding concrete one. The resulting 1:1 diagrams can be verified using forward or backward simulation.

Our analysis of verification conditions has focused on proofs using forward simulation, since generalized backward simulation in ASM refinement (like in refinement of IO automata) is weaker than generalized forward simulation due to the presence of infinite runs. Again we could show, that the verification conditions of forward simulation in the behavioral approach to data refinement are instances of those of generalized forward simulation in ASM refinement.

The result is not, what we initially expected, since the interpretation of $\perp$ as divergence in ASM does not coincide with the usual interpretation of $\perp$ as “not applicable” in behavioral data refinement. Nevertheless the proofs show, that both interpretation are possible for the behavioral approach, the interpretation as divergence even leads to a simpler ASM refinement.

On the other hand, the contract approach, which uses $\perp$ to denote divergence (or more general: undefinedness) and therefore seemed closer to ASM refinement, turned out not to be an exact instance of ASM refinement for two reasons. First, its semantics is different from the semantics of ASM rules. Second, the contract approach allows to implement divergent runs with terminating ones, which is not allowed in ASM refinement. The first problem can be solved by defining the contract approach for operations with a semantics that does not allow to “recover from bottom”. The second problem would require a modified, more liberal ASM refinement theory, which we leave to further work.

Data refinement is an important special case of ASM refinement, since it proves stronger conditions than ASM refinement: pre- and postconditions as well as invariants are preserved by data refinement, while in general ASM refinement only has a weak preservation theorem for invariants.

While traditional data refinement replaces one abstract by one concrete operation, a number of generalizations have been defined recently, that have more liberal refinement conditions. We analyzed two of them, forward simulation in weak refinement which adds internal operations to a data type, and coupled data refinement, which allows to implement one abstract operation by a fixed number of concrete ones. We found, that both are basically instances of ASM refinement too (modulo small differences in initialization). The definition of weak refinement is almost as general as ASM refinement (ASM rules are more expressive than the pre- postcondition specifications used for operations), but its verification conditions are more restrictive than those of ASM refinement.

Summarizing, our results confirm the view expressed in [5], that ASM refinement is a suitable framework for studying refinement of state-based systems in general. The research done in this paper has not been exhaustive: backwards simulation has not been considered, as well as instances of data refinement, that use explicit input/output or preconditions and guards. Further research is also necessary to integrate fairness conditions, as used in temporal logic approaches to the refinement of state-based systems.

Acknowledgments

I would like to thank Heike Wehrheim for clarifying some of the details of coupled refinement and for pointing to errors. I would like to thank John Derrick and Eerke Boiten for explaining some of the issues involving the rôle of the bottom element. Their comments have lead to significant corrections in the view of the contract approach and of weak refinement, and to a lot of other improvements. Thanks also to Egon Börger for his encouragement and helpful review. Last but not least I would like to thank an anonymous referee for the enormous work he put into his thorough review. His comments and corrections were of invaluable help in improving the paper.

References

[1] G. Schellhorn, Verification of ASM Refinements Using Generalized Forward Simulation, Journal of Universal Computer Science (J.UCS) 7 (11) (2001) 952–979, available at `http://hyperg.iicm.tu-graz.ac.at/jucs/`.

[2] E. Börger, A Logical Operational Semantics for Full Prolog. Part I: Selection Core and Control, in: E. Börger, H. Kleine Büning, M. M. Richter, W. Schönfeld

(Eds.), CSL'89. 3rd Workshop on Computer Science Logic, Vol. 440 of LNCS, Springer, 1990, pp. 36–64.

[3] E. Börger, A Logical Operational Semantics of Full Prolog. Part II: Built-in Predicates for Database Manipulation, in: B. Rovan (Ed.), Mathematical Foundations of Computer Science, Vol. 452 of LNCS, Springer, 1990, pp. 1–14.

[4] M. Gurevich, Evolving algebras 1993: Lipari guide, in: E. Börger (Ed.), Specification and Validation Methods, Oxford University Press, 1995, pp. 9 – 36.

[5] E. Börger, The ASM Refinement Method, Formal Aspects of Computing 15, pp. 237–257, 2003.

[6] E. Börger, D. Rosenzweig, The WAM—definition and compiler correctness, in: C. Beierle, L. Plümer (Eds.), Logic Programming: Formal Methods and Practical Applications, Vol. 11 of Studies in Computer Science and Artificial Intelligence, North-Holland, Amsterdam, 1995, pp. 20–90.

[7] J. Moore, PITON: A Verified Assembly Level Language, Technical report 22, Computational Logic Inc., available at the URL: http://www.cli.com (1988).

[8] D. Cyrluk, Microprocessor Verification in PVS: A Methodology and Simple Example, Tech. Rep. SRI-CSL-93-12, Computer Science Laboratory, SRI International (Dec. 1993).

[9] A. Dold, A Formal Representation of Abstract State Machines using PVS, Verifix-Report Ulm 6.2, Universität Ulm, (revised version) (1998).

[10] W. de Roever, K. Engelhardt, Data Refinement: Model-Oriented Proof Methods and their Comparison, Vol. 47 of Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, New York, NY, 1998.

[11] N. Lynch, F. Vaandrager, Forward and Backward Simulations – Part I: Untimed systems, Information and Computation 121(2) (1995) 214–233, also: Technical Memo MIT/LCS/TM-486.b, Laboratory for Computer Science, MIT.

[12] W. Reif, G. Schellhorn, K. Stenzel, M. Balser, Structured specifications and interactive proofs with KIV, in: W. Bibel, P. Schmitt (Eds.), Automated Deduction—A Basis for Applications, Kluwer Academic Publishers, Dordrecht, 1998, pp. 13 – 39.

[13] G. Schellhorn, W. Ahrendt, The WAM Case Study: Verifying Compiler Correctness for Prolog with KIV, in: W. Bibel, P. Schmitt (Eds.), Automated Deduction—A Basis for Applications, Kluwer Academic Publishers, Dordrecht, 1998, pp. 165 – 194.

[14] G. Schellhorn, Verification of Abstract State Machines, Ph.D. thesis, Universität Ulm, Fakultät für Informatik, (available at `www.informatik.uni-augsburg.de/swt/fmg/papers/`) (1999).

[15] J. Derrick, E. A. Boiten, H. Bowman, M. Steen, Weak Refinement in Z, in: J. Bowen, M. Hinchey, D.Till (Eds.), ZUM '97: The Z Formal Specification Notation, Vol. 1212, Springer LNCS, 1997, pp. 369–388.

[16] J. Derrick, H. Wehrheim, Using Coupled Simulations in Non-atomic Refinement, in: D. Bert, J. Bowen, S. King, M. Walden (Eds.), ZB 2003: Formal Specification and Development in Z and B, Vol. 2651, Springer LNCS, 2003, pp. 127–147.

[17] E. Börger, J. Schmid, Composition and submachine concepts for sequential ASMs, in: P. Clote, H. Schwichtenberg (Eds.), Computer Science Logic (Proceedings of CSL 2000), Vol. 1862 of Lecture Notes in Computer Science, Springer-Verlag, 2000, pp. 41–60.

[18] R. F. Stärk, S. Nanchen, A Complete Logic for Abstract State Machines, Journal of Universal Computer Science (J.UCS) Abstract State Machines 2001: Theory and Applications.

[19] L. Lamport, The temporal logic of actions, ACM Transactions on Programming Languages and Systems 16 (3).

[20] J. Derrick, E. Boiten, Refinement in Z and in Object-Z : Foundations and Advanced Applications, FACIT, Springer, 2001.

[21] J. M. Spivey, The Z Notation: A Reference Manual, 2nd Edition, Prentice Hall International Series in Computer Science, 1992.

[22] He Jifeng, C. A. R. Hoare, J. W. Sanders, Data refinement refined, in: B. Robinet, R. Wilhelm (Eds.), Proc. ESOP 86, Vol. 213 of Lecture Notes in Computer Science, Springer-Verlag, 1986, pp. 187–196.

[23] E. Boiten, W. de Roever, Getting to the Bottom of Relational Refinement: Relations and Correctness, Partial and Total (Extended Abstract), in: R. Berghammer, B. Möller (Eds.), 7th International Seminar on Relational Methods in Computer Science (RelMCS 7), University of Kiel, 2003, pp. 82–88.

[24] C. Bolton, J. Davies, J. Woodcock, On the refinement and simulation of data types and processes, in: K. Araki, A. Galloway, K. Taguchi (Eds.), Proceedings of the International conference of Integrated Formal Methods (IFM), Springer, 1999, pp. 273–292.

[25] J. C. P. Woodcock, J. Davies, Using Z: Specification, Proof and Refinement, Prentice Hall International Series in Computer Science, 1996.

[26] J. deBakker, Mathematical Theory of Program Correctness, International Series in Computer Science, Prentice-Hall, 1980.

[27] J.-R. Abrial, The B-Book: Assigning Programs to Meanings, Cambridge University Press, 1996.

[28] C. B. Jones, Systematic Software Development using VDM, 2nd Edition, Prentice Hall, 1990.

[29] N. Bjørner, A. Brown, M. Colon, B. Finkbeiner, Z. Manna, H. Sipma, T. Uribe, Verifying temporal properties of reactive systems: A step tutorial, in: Formal Methods in System Design, 2000, pp. 227–270.

[30] R. Miarka, E. Boiten, J. Derrick, Guards, Preconditions, and Refinement in Z, in: J. Bowen, S. Dunne, A. Galloway, S. King (Eds.), ZB2000: Formal Specification and Development in Z and B, Vol. 1878, Springer LNCS, 2000, pp. 286–303.

[31] B. H. Liskov, J. M. Wing, A Behavioral Notion of Subtyping, ACM Transactions on Programming Languages and Systems.

[32] J. Derrick, E. Boiten, Errata for [20], `http://www.cs.kent.ac.uk/people/staff/jd1/books/refine/root.html` (2002).

[33] J. Derrick, E. Boiten, Recent Advances in Refinement, in: E. Börger, A. Gargantini, E.Riccobene (Eds.), ZB2000: Formal Specification and Development in Z and B, Vol. 2589, Springer LNCS, 2003, pp. 33–56.