Consistency Enforcement for Static First-Order Invariants in Sequential Abstract State Machines

Klaus-Dieter Schewe(✉)

Zhejiang University, UIUC Institute, Haining, China
kdschewe@acm.org

Abstract. Given a program specification P and a first-order static invariant I the problem of consistency enforcement is to determine a modified program specification P_I that is consistent with respect to I, i.e. whenever I holds in a state S it also holds in the successor states determined by P_I, and at the same time only minimally deviates from P. We formalise this problem by the notion of the *greatest consistent specialisation* (GCS) adapting and generalising this 20 year old concept to sequential Abstract State Machines (ASMs) with emphasis on bounded parallelism. In a state satisfying I such that P is repairable the notion of consistent specialisation will require an enlargement of the update set, which defines a partial order with respect to which a GCS is defined. We show that GCSs are compositional in two respects: (1) the GCS of an ASM with a complex rule can be obtained from the GCSs of the involved assignments, and (2) the GCS with respect to a set of invariants can be built using the GCSs for the individual invariants in the set.

Keywords: Consistency enforcement · Static invariant · Consistent specialisation · Abstract State Machine · Compositionality

1 Introduction

State-based formal methods for the development of software systems place emphasis on correctness proofs, most of which are concerned with *consistency* and *refinement*. For the former ones we consider static invariants I that are to hold in every state S of a program specification P. Such invariants are expressed as logical formulae, most importantly using first-order logic (though not restricted to this), and consistency verification splits the problem into showing that consistency holds for the initial state(s) and is preserved by state transitions defined by P, i.e. whenever I holds in a state S it also holds in the successor state(s) determined by P. If such a proof fails, P has to be modified, but there is very little methodological support for this.

A remedy to this lack of methodology is provided by *consistency enforcement*, which has already been studied in the 1990s, in particular in the field of databases [13]. The emphasis was mainly on the use of rule triggering systems,

© Springer Nature Switzerland AG 2019
Y. Ait-Ameur and S. Qin (Eds.): ICFEM 2019, LNCS 11852, pp. 219–235, 2019.
https://doi.org/10.1007/978-3-030-32409-4_14

but in view of many problems associated with the active database approach (see [14]) the interest has somehow died out. Unfortunately, this also stopped further investigation of alternative approaches in rigorous state-based methods such as *greatest consistent specialisations* (GCSs, see [15]), which were grounded in Dijkstra's guarded commands with predicate transformer semantics [5] in an extended form [12] (also used at that time in the upcoming B method [1]).

In a nutshell, the problem of consistency enforcement is to determine a modified program specification P_I that is consistent with respect to I, i.e. whenever I holds in a state S it also holds in the successor states determined by P_I, and at the same time only minimally deviates from P, where "minimal deviation" is formalised by a specialisation order, i.e. P_I is maximal with respect to this order among all consistent specialisations of P with respect to I. In the theory of GCSs it could be shown that compositionally with respect to sets of invariants holds, i.e. a GCS for the conjunction of invariants in a set can be essentially built by taking the GCSs with respect to the individual invariants in arbitrary order. It could further be shown that compositionality with respect to the composition of P can also be achieved.

Nonetheless, the research remained uncompleted and still not fully satisfactory. The notion of state space was adopted from B, and thus refers to a finite set of state variables. When these are bound to bulk data such as sets or relations, the notion of specialisation becomes too restricted as already observed in [11]. Furthermore, the compositional GCS construction does not take parallelism into account, not even bounded parallelism, and the handling of GCSs for assignments has not been addressed.

In this paper, we pick up the thread from the 20 year old research and investigate consistency enforcement in the context of Abstract State Machines (ASMs, [3]). In doing so we emphasise Tarski structures as states with much more fine-grained locations, as well as non-determinism and parallelism, but we still restrict our investigation to bounded parallelism as in sequential ASMs [10]. We also restrict static invariants to formulae in first-order logic. As predicate transformers are not suited to capture parallelism, we redefine the notion of GCS—this will be done in Sect. 2 dedicated to preliminaries—but for simple state variables and specifications without parallelism the new definition will cover the old one. We will show that it is still possible to characterise specialisation using the logic of non-deterministic ASMs [8,9]. With this we are able to prove again compositionality with respect to the structure of ASM rules—this gives the main content of Sect. 3 dedicated to compositionality with respect to rules. This compositionality result allows us to concentrate on GCSs for assignments.

In Sect. 4 we address compositionality with respect to sets of invariants. It is easy to see that with the fine-grained notion of location compositionality for sets of invariants as in the guarded-command-based GCS theory cannot be achieved. However, as only assignments have to be taken into consideration, we can exploit *critical paths* as in [14], where they were used to show the limitations of rule triggering systems for consistency enforcement. Here we will use them, again in connection with *local stratification* to obtain a canonical form for the

GCS of an assignment with respect to a set of invariants. For the latter ones we exploit a representation in clausal form, where skolemised variables give rise to **choose**-rules. We conclude with a brief summery and outlook in Sect. 5.

2 Consistent Specialisations

As stated above consistency enforcement starts from a program specification P and a static invariant I. For the former one we will concentrate on specifications of *sequential algorithms*, and we know from [10] that these are captured by sequential ASMs, so our specifications will be based on ASMs (see [3] for a detailed introduction).

Thus, let Σ denote a *signature*, i.e. a finite set of function symbols. Each $f \in \Sigma$ has a fixed arity $ar_f \in \mathbb{N}$ (including 0). A *structure* S over Σ is given by a base set B and an interpretation of the function symbols, i.e. for $ar_f = n$ the structure contains a function $f_S : B^n \to B \cup \{\bot\}$, where $\bot$ denotes a value outside B representing undefinedness, by means of which partial functions are captured. An *isomorphism* between Σ-structures S and S' is a bijection $\sigma : B \to B'$ between the corresponding base sets of S and S', respectively (extended by $\sigma(\bot) = \bot$) such that $f_{S'}(\sigma(v_1), \ldots, \sigma(v_n)) = \sigma(f_S(v_1, \ldots, v_n))$.

Usually, we assume an implicit background, i.e. fixed constant values and operations on them, e.g. truth values and their junctors, or natural numbers. The background together with Σ allows us to define terms and formulae (using equality as predicate symbol) in the usual way. Their interpretation in a structure S is then defined as usual assuming that the constants are interpreted by themselves and the operations on them have a fixed interpretation, with their fixed interpretation in a structure S.

A (static) *invariant* over Σ is a closed first-order formula I in a logic with equality and only the function symbols in Σ (and the background). A structure S *satisfies* the invariant I (denoted as $S \models I$) iff the interpretation of I in S yields **true**.

2.1 Sequential Abstract State Machines

ASMs provide means for the specification of computations on isomorphism classes of structures. Thus, an ASM is defined by a signature Σ and an ASM closed rule r. The signature defines a *set $\mathcal{S}$ of states*, out of which a subset $\mathcal{I}$ is declared as *set of initial states*. Both $\mathcal{S}$ and $\mathcal{I}$ must be closed under isomorphisms.

Each state defines a set of locations. A *location* ℓ of S is given by a function symbol $f \in \Sigma$ (say of arity n) and a tuple $(v_1, \ldots, v_n) \in B^n$, where B is the base set of S. We write $val_S(\ell) = v_0$ for a location $\ell = (f, (v_1, \ldots, v_n))$ iff $f_S(v_1, \ldots, v_n) = v_0$ holds.

An *update* is a pair (ℓ, v_0) comprising a location ℓ and a value $v_0 \in B \cup \{\bot\}$. An *update set* is a finite set Δ of updates. Δ is called *clash-free* iff there cannot be two updates $(\ell, v), (\ell, v') \in \Delta$ with the same location ℓ and $v \neq v'$. If Δ is a clash-free update set on state S, then there is a well-defined *successor state*

$S' = S + \Delta$ with $val_{S'}(\ell) = v_0$ for $(\ell, v_0) \in \Delta$ and $val_{S'}(\ell) = val_S(\ell)$ otherwise. For completeness we further let $S + \Delta = S$ in case Δ is not clash-free.

The rule gives rise to state transitions. Sequential *ASM rules* r are defined as follows:

- Each *assignment* $f(t_1, \ldots, t_n) := t_0$ with a function symbol $f \in \Sigma$ of arity n and terms t_i for $i = 0, \ldots, n$ is an ASM rule.
- If φ is a Boolean term (i.e. it evaluates to a truth value) and r_1 and r_2 are ASM rules, then also **if** φ **then** r_1 **else** r_2 **endif** is an ASM rule (branching).
- If $r_1, \ldots, r_k$ are ASM rules, then also **par** $r_1 \ldots r_k$ **endpar** is an ASM rule (bounded parallelism).
- If x is a variable, $\varphi(x)$ is a Boolean term with free x and $r(x)$ is an ASM rule with free x, then also **choose** x **with** $\varphi(x)$ **do** r(x) **enddo** is an ASM rule (choice).

For completeness we also permit rules of the form **let** $x = t$ **in** $r(x)$, **skip** and **fail**. The former one is just a shortcut for **choose** x **with** $x = t$ **do** r(x) **enddo** emphasising that the "choice" is deterministic, **skip** can be seen as a shortcut for some $f(t_1, \ldots, t_n) := f(t_1, \ldots, t_n)$, i.e. a rule that does not change the state, and **fail** represents **choose** x **with** $x \neq x$ **do** r(x) **enddo**, i.e. it is a rule that is always undefined.

We use parentheses freely as well as the usual abbreviations for branching rules. We also mention the unbounded parallelism rule **forall** x **with** $\varphi(x)$ **do** r(x) **enddo**, which is permitted in general in ASMs, but not in sequential ASMs.

Given a state S a rule r together with a valuation ζ of its free variables yields a set of update sets, which we denote as $\boldsymbol{\Delta}_r(S)$:

- For an assignment rule $f(t_1, \ldots, t_n) := t_0$ there is exactly one update set, so we have $\boldsymbol{\Delta}_r(S) = \{\{((f, (val_{S,\zeta}(t_1), \ldots, val_{S,\zeta}(t_n))), val_{S,\zeta}(t_0))\}\}$.
- For a branching rule **if** φ **then** r_1 **else** r_2 **endif** we have $\boldsymbol{\Delta}_r(S) = \boldsymbol{\Delta}_{r_1}(S)$ for $val_{S,\zeta}(\varphi) = \mathbf{true}$, and $\boldsymbol{\Delta}_r(S) = \boldsymbol{\Delta}_{r_2}(S)$ for $val_{S,\zeta}(\varphi) = \mathbf{false}$.
- For a bounded parallel rule **par** $r_1 \ldots r_k$ **endpar** we have $\boldsymbol{\Delta}_r(S) = \{\Delta_1 \cup \cdots \cup \Delta_k \mid \Delta_i \in \boldsymbol{\Delta}_{r_i}(S) \text{ for } i = 1, \ldots, k\}$.
- For a choice rule **choose** x **with** $\varphi(x)$ **do** r(x) **enddo** we have $\boldsymbol{\Delta}_r(S) = \bigcup_{val_{S,\zeta}(\varphi(x))=\mathbf{true}} \boldsymbol{\Delta}_{r(x)}(S)$.

Then we also have $\boldsymbol{\Delta}_{\mathbf{skip}}(S) = \{\emptyset\}$ and $\boldsymbol{\Delta}_{\mathbf{fail}}(S) = \emptyset$.

Definition 1. An ASM is *consistent* with respect to an invariant I iff every initial state $S_0 \in \mathcal{I}$ satisfies I, and for every state $S \in \mathcal{S}$ satisfying I also every successor state $S' = S + \Delta$ with $\Delta \in \boldsymbol{\Delta}_r(S)$ satisfies I.

2.2 Greatest Consistent Specialisations

Recall from the introduction that the idea is to replace a specification P (i.e. a sequential ASM) by a modified specification P_I that is consistent with respect to a given invariant I subject to some minimality condition. Let us tacitly assume

that the given initial states S_0 always satisfy $S_0 \models I$, so we can concentrate on the state transitions. That is, we have to modify the rule r of a given sequential ASM. Therefore, in the following we always consider such a rule r.

Update sets in a state S represent the intention behind the specification. It therefore appears natural to request that the modification should preserve the updates, i.e. update sets should be (minimally) enlarged. The question is whether this is always possible, and the next example shows that this is not the case (see also [14]).

Example 1. Consider an ASM with a rule **par** $p(a) := \mathbf{true}$ $q(a) := \mathbf{false}$ **endpar** and an invariant $I \equiv \forall x.p(x) \Rightarrow q(x)$. Assume a is a constant evaluated to itself, so the only update set in an arbitrary state S will be $\Delta = \{((p,(a)),\mathbf{true}),((q,(a)),\mathbf{false})\}$. Hence in every successor state $S' = S+\Delta$ we will have $val_{S'}(p(a)) = \mathbf{true}$ and $val_{S'}(q(a)) = \mathbf{false}$, which violates I, and no enlargement of Δ can repair this. So the specification has to be considered to be non-repairable.

Example 1 shows that there are situations, where consistency enforcement by means of enlarging update sets is not possible, so we need a notion of a repairable update set. For this we consider a clash-free update set $\Delta \in \boldsymbol{\Delta}_r(S)$ as a structure, i.e. we let $val_\Delta(\ell) = v$ for $(\ell, v) \in \Delta$ and $val_\Delta(\ell) = \bot$ otherwise.

Definition 2. An update set $\Delta \in \boldsymbol{\Delta}_r(S)$ is *non-repairable* with respect to an invariant I iff $\Delta \models \neg I$ holds. An ASM is *repairable* in state S with respect to I iff there is at least one repairable update set $\Delta \in \boldsymbol{\Delta}_r(S)$.

Now we can approach a definition of consistent specialisation.

Definition 3. Consider an ASM $\mathcal{A}$ with a rule r and an invariant I. An ASM $\mathcal{A}_I$ with rule r_I is a *consistent specialisation* of $\mathcal{A}$ iff for all states S the following holds:

(i) If $S \models I$, then
  (a) for all repairable $\Delta \in \boldsymbol{\Delta}_r(S)$ there exists some $\Delta' \in \boldsymbol{\Delta}_{r_I}(S)$ with $\Delta \subseteq \Delta'$, and
  (b) for all $\Delta' \in \boldsymbol{\Delta}_{r_I}(S)$ there exists a repairable $\Delta \in \boldsymbol{\Delta}_r(S)$ with $\Delta \subseteq \Delta'$.
(ii) If $S \not\models I$, then $\boldsymbol{\Delta}_{r_I}(S) = \emptyset$.
(iii) If $S \models I$, then for $\Delta' \in \boldsymbol{\Delta}_{r_I}(S)$ we have $S + \Delta' \models I$.

Note that (i) and (iii) in Definition 3 together imply that if $\boldsymbol{\Delta}_r(S)$ only contains non-repairable update sets, then $\boldsymbol{\Delta}_{r_I}(S)$ will be empty.

We can define a partial order $\leq$ on consistent specialisations of $\mathcal{A}$ with respect to an invariant I. If $\mathcal{A}_1$ and $\mathcal{A}_2$ are two consistent specialisations of $\mathcal{A}$ with rules r_1 and r_2, respectively, then we have $\mathcal{A}_1 \leq \mathcal{A}_2$ iff for all states S and all $\Delta_1 \in \boldsymbol{\Delta}_{r_1}(S)$ there exists a $\Delta_2 \in \boldsymbol{\Delta}_{r_2}(S)$ with $\Delta_2 \subseteq \Delta_1$.

Clearly, if an update set is repairable, then the greatest consistent specialisation will contain all minimal extensions (exploit unbounded choice for this), which justifies the following definition.

Definition 4. Consider an ASM $\mathcal{A}$ with a rule r and an invariant I. An ASM $\mathcal{A}_I$ with rule r_I is the *greatest consistent specialisation* (GCS) of $\mathcal{A}$ iff it is a consistent specialisation and maximal with respect to the partial order $\leq$.

Remarks. Note that our definition of GCS is an ASM $\mathcal{A}_I$ over the same signature as the given ASM $\mathcal{A}$. Thus, states of $\mathcal{A}_I$ are also states of $\mathcal{A}$ and vice versa. Furthermore, the definition is based on the assumption that the invariant is confirmed to be valid, which is different from work as e.g. in [16], where invariants are allowed to be weakened to achieve consistency. GCS are also "universal" in the sense that all possible ways of minimal repair (as defined in Definition 3) are taken into account leaving decisions, which of these might be preferred to a human-driven refinement process. This differs from work of others, e.g. [4], where machine learning methods are applied to detect the "best" repair.

Example 2. Consider a simple invariant $I \equiv f \neq \bot \Rightarrow g(f) \neq \bot$, so the signature Σ contains at least function symbols f and g of arity 0 and 1, respectively. Take the rule r_1 defined by $f := a$ with a constant a. Then the GCS will be defined by a rule

if $f = \bot \vee g(f) \neq \bot$
then choose y **do par** $f := a$ $g(a) := y$ **endpar enddo**
else fail endif

This is easily verified using directly Definition 3. Note that the outermost branching rule and the **fail** is only needed to capture the case of the rule being applied in a state S not satisfying I.

Next take a rule r_2 defined by $g(a) := b$ with another constant b. Then the GCS is defined by

if $f = \bot \vee g(f) \neq \bot$ **then** $g(a) := b$ **else fail endif**

Clearly, the GCS of **par** r_1 r_2 **endpar** is

if $f = \bot \vee g(f) \neq \bot$ **then par** $f := a$ $g(a) := b$ **endpar else fail endif**

This is different from **par** r_{1I} r_{2I} **endpar**.

Example 3. Let us consider another simple, yet slightly more complicated example with an invariant $I \equiv f \neq \bot \Rightarrow \exists y.g(f, y) = f$. Let r_1 be as in Example 2 and let r_2 be defined as $g(a, b) := a$. Then the GCS of r_1 is defined by

if $f = \bot \vee \forall y.g(f, y) \neq f$
then choose y **do par** $f := a$ $g(a, y) := a$ **endpar enddo**
else fail endif

For r_2 the GCS will be defined by the rule

if $f = \bot \vee \forall y.g(f, y) \neq f$ **then** $g(a, b) := a$ **else fail endif**

Clearly, the GCS of **par** r_1 r_2 **endpar** is

if $f = \bot \vee \forall y.g(f, y) \neq f$
then par $f := a$ $g(a, y) := a$ **endpar**
else fail endif

Again, this is different from **par** r_{1I} r_{2I} **endpar**.

Remark. In principle, the new definition of a GCS follows the idea of the definition given in [15], but as we want to deal with parallelism, the use of predicate transformers is excluded. In addition, there are a few more subtle differences. Definition 4 is based on update sets, which are grounded on the fine-tuned notion of location in ASMs. For instance, every tuple in a relation defines a location, whereas in the "old" work the whole relation would be considered as a single value with the consequence that preserving "effects" (now updates) is much more restrictive. We will see that this has consequences on compositionality with respect to sets of invariants, but in Sect. 4 we will be able to achieve results that are even more convincing than those for the "old" notion of GCSs.

3 Compositionality with Respect to Rule Composition

The key result in [15] states that the GCS can be built by first replacing elementary commands, i.e. assignments and **skip**, by their respective GCSs, and then adding preconditions to ensure that the result is indeed a specialisation. This reduces GCS construction to the case of assignments. The most difficult parts of the decisive "upper bound theorem" are concerned with sequences and recursion. In our modernised theory we can dispense with recursion, as the semantics of ASMs includes the iteration of the rule, so we have a finer-grained notion of consistency. Nonetheless, an extension to recursive ASMs (see [2]) is likewise possible. As we deal with sequential algorithms there is also no need to bother about sequences, as they are not needed (see the proof of the sequential ASM thesis in [10] and the corresponding behavioural theory of unbounded parallel algorithms in [7]; sequences can be easily expressed using bounded parallelism and branching). However, we now have to deal at least with bounded parallelism, to which the theory in [15] does not apply.

In this section we will nonetheless show that compositionality holds for sequential ASM rules, and the most difficult part of the proof will concern the case of the bounded parallel rule. This will reduce GCS construction again to assignments, which we will handle in the next section.

3.1 Branching and Choice

First consider the cases of branching and choice rules.

Proposition 1. *Let $\mathcal{A}$ be an ASM with a rule of the form* ***if*** φ ***then*** r_1 ***else*** r_2 ***endif****. Then the rule of its GCS $\mathcal{A}_I$ with respect to an invariant I can be written as* ***if*** φ ***then*** r_{1I} ***else*** r_{2I} ***endif****, where r_{iI} is the rule of the GCS of the ASM defined by r_i.*

Due to space and time restrictions we omit the proof here. We only have to check the conditions of Definitions 3 and 4 using the definition of update sets yielded by branching rules. This is rather straightforward.

Proposition 2. *Let $\mathcal{A}$ be an ASM with a rule of the form* ***choose*** x ***with*** $\varphi(x)$ ***do*** $r(x)$ ***enddo****. Then the rule of its GCS $\mathcal{A}_I$ with respect to an invariant I can be written as* ***choose*** x ***with*** $\varphi(x)$ ***do*** $r_I(x)$ ***enddo****, where $r_I(x)$ is the rule of the GCS of the ASM defined by $r(x)$.*

Same as for the proof of Proposition 1 the proof of Proposition 2 is rather straightforward based on the definition of update sets yielded by a choice rule and Definitions 3 and 4.

3.2 Bounded Parallelism

Next consider the case of a bounded parallel rule. As Examples 2 and 3 already show, we cannot obtain a simple compositional result as in Proposition 1 or 2. If we build the GCSs of the different parallel branches of an unbounded parallel rule separately and recombine them using unbounded parallelism, then we may have additional branches that subsume those in the real GCS. There may also be **fail** branches. Both have to be excluded.

In order to do this we need to exploit the logic of non-deterministic ASMs [8,9], which is defined as a fragment of second-order logic, in which quantification over update sets is permitted. However, as we are not yet dealing with unbounded parallelism, we can simplify the logic discarding multiset functions and thus also update multisets. We can also dispense with the difficult handling of meta-finite structures.

Terms of the logic are the terms defined by the given ASM. *Formulae* of the logic are built inductively using the following rules:

- If s and t are terms, then $s = t$ is a formula.
- If $t_1, \ldots, t_r$ are terms and X is a second-order variable of arity r, then $X(t_1, \ldots, t_r)$ is a formula.
- If r is a rule and X is a second-order variable of arity 3, then $\mathrm{upd}(r, X)$ is a formula.
- If φ and ψ are formulae, then also $\neg\varphi$, $\varphi \vee \psi$ are formulae.
- If φ is a formula, x is a first-order variable and X is a second-order variable, then also $\forall x.\varphi$ and $\forall X.\varphi$ are formulae.

– If φ is a formula and X is a second-order variable of arity 3, then $[X]\varphi$ is a formula.

Additional junctors and quantifiers such as $\wedge$, $\rightarrow$ and $\exists$ are defined as shortcuts in the usual way. We also use $\langle X\rangle\varphi$ as shortcut for $\neg[X]\neg\varphi$ as common in dynamic logic.

In [8] a Henkin semantics for the logic was defined, in which the interpretation of second-order quantifiers is part of the specification of a structure.

Definition 5. A *Henkin prestructure* over signature Σ is a state S with non-empty base set B extended by sets of n-ary relations $D_n \subseteq \mathcal{P}(B^n)$ for each $n \geq 1$.

Variable assignments ζ for a Henkin prestructure S are defined as usual. We have $\zeta(x) \in B$ for first-order variables x, and $\zeta(X) \in D_n$ for second-order variables X of arity n. Given a variable assignment, formulae can be interpreted in a Henkin prestructure. As we want to obtain update sets, we introduce new constant symbols c_f for each function symbol $f \in \Sigma$. For a second-order variable X of arity 3 we write $val_{S,\zeta}(X) \in \Delta(r, S, \zeta)$, meaning that there is a set $\Delta \in \Delta(r, S, \zeta)$ such that $(f, a_0, a_1) \in U$ iff $(c_f^S, a_0, a_1) \in val_{S,\zeta}(X)$.

If φ is a formula, then the truth value of φ on S under ζ is determined as follows:

– If φ is of the form $s = t$, then $[\![\varphi]\!]_{S,\zeta} = \mathbf{true}$ iff $val_{S,\zeta}(s) = val_{S,\zeta}(t)$.
– If φ is of the form $X(t_1, \ldots, t_r)$, then $[\![\varphi]\!]_{S,\zeta} = \mathbf{true}$ iff $(val_{S,\zeta}(t_1), \ldots, val_{S,\zeta}(t_n)) \in val_{S,\zeta}(X)$.
– If φ is of the form $\mathrm{upd}(r, X)$, then $[\![\varphi]\!]_{S,\zeta} = \mathbf{true}$ iff $val_{S,\zeta}(X) \in \Delta(r, S, \zeta)$.
– If φ is of the form $\neg\psi$, then $[\![\varphi]\!]_{S,\zeta} = \mathbf{true}$ iff $[\![\psi]\!]_{S,\zeta} = \mathbf{false}$.
– If φ is of the form $(\alpha \vee \psi)$, then $[\![\varphi]\!]_{S,\zeta} = \mathbf{false}$ iff $[\![\alpha]\!]_{S,\zeta} = [\![\psi]\!]_{S,\zeta} = \mathbf{false}$.
– If φ is of the form $\forall x.\psi$, then $[\![\varphi]\!]_{S,\zeta} = \mathbf{true}$ iff $[\![\psi]\!]_{S,\zeta[x\mapsto a]} = \mathbf{true}$ for all $a \in B$.
– If φ is of the form $\forall X.\psi$, where X is a second-order variable of arity n, then $[\![\varphi]\!]_{S,\zeta} = \mathbf{true}$ iff $[\![\psi]\!]_{S,\zeta[X\mapsto R]} = \mathbf{true}$ for all $R \in D_n$.
– If φ is of the form $[X]\psi$, then $[\![\varphi]\!]_{S,\zeta} = \mathbf{false}$ iff $\zeta(X)$ represents a clash-free update set Δ such that $[\![\psi]\!]_{S+\Delta,\zeta} = \mathbf{false}$ holds.

We can use the logic to define $\mathrm{con}(r, X)$ to express that X represents one of the possible update sets generated by the rule r, and that X is clash-free. This can be expressed in the logic by the formula $\mathrm{con}(r, X) \equiv \mathrm{upd}(r, X) \wedge \mathrm{conUSet}(X)$, where

$$\mathrm{conUSet}(X) \equiv \bigwedge_{f\in\Sigma} \forall x, y, z.X(c_f, x, y) \wedge X(c_f, x, z) \rightarrow y = z.$$

Furthermore, in accordance with [15, Prop.7] we can use $\chi(r, r')$ to express that r' subsumes r. This can be expressed in the logic by

$$\chi(r, r') \equiv \{L' \mapsto L\}.[r'']\langle r\rangle(L = L'),$$

where L represents the common locations used by rules r and r', L' is a disjoint copy of L, and r'' results from r' by replacing all locations in L by the corresponding ones in L'.

Proposition 3. *Let $\mathcal{A}$ be an ASM with a rule of the form* ***par*** $r_1 \dots r_k$ ***endpar****. Then the rule of its GCS $\mathcal{A}_I$ with respect to an invariant I can be written as*

$$\textbf{\textit{choose}}\ \boldsymbol{y}_1, \dots, \boldsymbol{y}_k\ \textit{with} \bigwedge_{1 \le i \le k} \psi_i(\boldsymbol{y}_i) \wedge \exists X_i.con(r'_i(\boldsymbol{y}_i), X_i) \quad \textbf{\textit{do par}} \dots \hat{r}_i(\boldsymbol{y}_i) \dots \ \textbf{\textit{endpar enddo}},$$

where $\hat{r}_i(\boldsymbol{y}_i)$ is defined to be the rule

$$\textbf{\textit{if}} \bigwedge_{j \neq i} \neg\chi(r'_j(\boldsymbol{y}_j), r'_i(\boldsymbol{y}_i))\ \textbf{\textit{then}}\ r'_i(\boldsymbol{y}_i)\ \textbf{\textit{else fail endif}}$$

using the GCSs with the rules r_{iI} of the form

$$\textbf{\textit{choose}}\ \boldsymbol{y}_i\ \textbf{\textit{with}}\ \psi_i(\boldsymbol{y}_i)\ \textbf{\textit{do}}\ r'_i(\boldsymbol{y}_i)\ \textbf{\textit{enddo}}.$$

For the proof, which we have to omit due to space and time restrictions, we look again directly at Definitions 3 and 4. It is easy to see that the parallel composition of GCSs r_{iI} defines a consistent specialisation, but in general not the GCS. Conditions $\text{con}(r'_i(\boldsymbol{y}_i), X_i)$ remove branches with clashes in update sets, and conditions $\neg\chi(r'_j(\boldsymbol{y}_j), r'_i(\boldsymbol{y}_i))$ discard parallel branches yielding update sets that contain update sets from other branches.

Note that in Proposition 3 we assume that the GCSs r_{iI} of the rules r_i have a specific form. This is in accordance with Propositions 1, 2 and (inductively) 3 as well as with the key result (see Theorem 1 in Sect. 4), on GCSs of assignments.

Example 4. Let us reconsider Examples 2 and 3. In the former case the parallel composition of r_{1I} and r_{2I} leads to the rule

if $f = \bot \vee g(f) \neq \bot$
then choose y **do par** $f := a$ $g(a) := b$ $g(a) := y$ **endpar enddo else fail endif**

Each parallel branch with $y \neq b$ defines only update sets that are not clash-free. The requirement in Proposition 3 that there must exist a clash-free update set removes these branches, so only the correct GCS remains.

In Example 3 the parallel composition of r_{1I} and r_{2I} leads to the rule

if $f = \bot \vee \forall y.g(f, y) \neq f$
then choose y **do par** $f := a$ $g(a, b) := a$ $g(a, y) := a$ **endpar enddo**
else fail endif

Here a parallel branch with $y \neq b$ leads to update sets with an additional update $((g, (a, y)), a)$, hence $\chi(r'(b), r'(y))$ holds (using $r'(y)$ as name for the rules inside the **par**-block). Then the condition in Proposition 3 excludes such branches, and we obtain the correct GCS.

4 Compositionality with Respect to Sets of Invariants

Propositions 1, 2 and 3 allow us to build the GCS in a compositional way, provided the GCSs of assignments are known. We will address assignments in this section. We will combine this directly with the treatment of sets of invariants. Clearly, consistency with respect to such a set is equivalent to consistency with respect to the conjunction of the invariants in the set, so the results of the previous section can be preserved.

In [15, Prop.12] it could be shown that with the coarse notion of GCS defined there compositionality with respect to conjunctions can be easily obtained. In other words, a GCS with respect to a set of invariants can be built step-by-step using the invariants in the set in arbitrary order. This does no longer hold in our modernised and fine-grained case, as the following example shows.

Example 5. Consider two invariants $I_1 \equiv \forall x.p(x) \Rightarrow q(x)$ and $I_2 \equiv \forall x.p(x) \Rightarrow p(f(x))$. Taking an ASM with a rule $p(a) := \mathit{true}$, its GCS with respect to I_1 is defined by the rule **par** $p(a) := \mathit{true}$ $q(a) := \mathit{true}$ **endpar** on states satisfying I_1. Building for this the GCS with respect to I_2 gives a rule, which on states satisfying $I_1 \wedge I_2$ takes the form **par** $p(a) := \mathit{true}$ $q(a) := \mathit{true}$ $p(f(a)) := \mathit{true}$ **endpar**, which is not the GCS with respect to $I_1 \wedge I_2$.

The reason for the discrepancy between Example 5 and the "old" theory in [15] is again due to the changed treatment of locations. Using the theory in [15] p and q would be state variables bound to unary relations, i.e. sets, and a GCS for an assignment such as $p(a) := \mathit{true}$—in fact an insertion—cannot make further changes to p, and only use preconditions to enforce I_2.

Fortunately, there is away to deal simultaneously with several invariants mutually influencing each other. This is inspired by the handling of rule triggering systems in [14]. We will adopt this theory here exploiting the fact that it is only required for consistency enforcement for a single assignment. In fact, single assignments are almost always repairable, unless the invariant contains a single literal.

4.1 Clausal Form and Atomic Repairs

We assume that invariants I are first-order formulae, so we can write them in *prenex normal form*

$$\forall \boldsymbol{x}_1 \exists \boldsymbol{y}_1 \forall \boldsymbol{x}_2 \exists \boldsymbol{y}_2 \ldots . \varphi(\boldsymbol{x}_1, \boldsymbol{y}_1, \boldsymbol{x}_2, \boldsymbol{y}_2, \ldots).$$

Then the existentially quantified variables can be replaced by Skolem functions $y_{1j} = sk_{1j}(\boldsymbol{x}_1)$, $y_{2j} = sk_{2j}(\boldsymbol{x}_1, sk_{11}(\boldsymbol{x}_1), \ldots, sk_{1n_1}(\boldsymbol{x}_1), \boldsymbol{x}_2)$, etc.

We can further assume that the quantifier-free formula $\varphi(\boldsymbol{x}_1, \boldsymbol{y}_1, \boldsymbol{x}_2, \boldsymbol{y}_2, \ldots)$ is written in conjunctive normal form, so it gives rise to a set of clauses, i.e. disjunctions of literals $\neg L_1 \vee \neg L_2 \vee \cdots \vee \neg L_k \vee L_{k+1} \vee \cdots \vee L_{k+\ell}$ with the variables x_{ij} and y_{ij} appearing in the atoms. Furthermore, atoms are simply equations.

A violation of an invariant I is always linked to a violation of one of its clauses, so we concentrate on the clauses. As in the relational case treated in [14] we may define an *atomic repair* by means of a trigger. In case a positive literal t_1 or a negative literal $\neg t_2$ is violated by an assignment changing the value of either t_1 or t_2 and this leads to the whole clause to become false, we can use another assignment making a positive literal $t_1' \neq t_2'$ to become **true** or a negative literal $\neg t_1' \neq \neg t_2'$ become **false**. Any variable appearing in t_1 or t_2 must also be bound to the same value in the triggered update, and all other variables must be selected by embedding the assignment into a **choose**-rule. Any such possibility defines an *atomic repair rule* comprising an *event* E, i.e. an assignment rule leading to an invariant violation, a *clause* I that is violated, and a *repair* R, which is an assignment embedded in a choice-rule. In analogy to rule triggering system we write **on** E **if** $\neg I$ **do** R for such an atomic repair rule.

Example 6. Let us consider a simple example adopted from [14, Ex.5] with three clauses:

$$I_1 \equiv \neg p(x) \vee \neg r(x) \vee q(x) \quad I_2 \equiv \neg q(x) \vee p(x) \quad I_3 \equiv \neg p(x) \vee r(x)$$

Then we obtain the following ten atomic repair rules:

$$\begin{aligned}
R_1 &: \textbf{on } p(x) := \textbf{true } \textbf{if } \neg I_1 \textbf{ do } q(x) := \textbf{true} \\
R_2 &: \textbf{on } q(x) := \textbf{false } \textbf{if } \neg I_1 \textbf{ do } p(x) := \textbf{false} \\
R_3 &: \textbf{on } r(x) := \textbf{true } \textbf{if } \neg I_1 \textbf{ do } q(x) := \textbf{true} \\
R_4 &: \textbf{on } q(x) := \textbf{false } \textbf{if } \neg I_1 \textbf{ do } r(x) := \textbf{false} \\
R_5 &: \textbf{on } q(x) := \textbf{true } \textbf{if } \neg I_2 \textbf{ do } p(x) := \textbf{true} \\
R_6 &: \textbf{on } p(x) := \textbf{false } \textbf{if } \neg I_2 \textbf{ do } q(x) := \textbf{false} \\
R_7 &: \textbf{on } p(x) := \textbf{true } \textbf{if } \neg I_3 \textbf{ do } r(x) := \textbf{true} \\
R_8 &: \textbf{on } r(x) := \textbf{false } \textbf{if } \neg I_3 \textbf{ do } p(x) := \textbf{false} \\
R_9 &: \textbf{on } p(x) := \textbf{true } \textbf{if } \neg I_1 \textbf{ do } r(x) := \textbf{false} \\
R_{10} &: \textbf{on } r(x) := \textbf{true } \textbf{if } \neg I_1 \textbf{ do } p(x) := \textbf{false}
\end{aligned}$$

It is easy to see that all possible atomic repair rules—assuming a single assignment causing a violation—can be derived from a clause, the **on**-part refers to the violating assignment, the **if**-part is the negation of the clause, and the **do**-part is an assignment corresponding to another literal in the clause, by means of which the violation would be disabled.

A set of atomic repair rules is said to be *complete* for an assignment rule r iff for every possible violation of an invariant clause I defined by a set of invariants there is at least one atomic repair rule with event r and clause I, and the same holds for all assignment rules appearing as repair in at least one atomic repair rule. A complete set of repair rules for r defines a sequential ASM rule, which we call a *complete repair* and denote as r_{rep}.

4.2 Critical Paths

We will now provide the means to reduce a system of atomic repair rules in such a way that we can obtain a GCS. For this we adapt the notions of *rule graph* and *critical trigger path*.

Definition 6. Let Σ be a signature and $\mathcal{R}$ a set of atomic repair rules on Σ. Then the associated *rule graph* (V, E) is defined as follows:

- The set V of vertices is the disjoint union of Σ and $\mathcal{R}$. We then talk of Σ-vertices and $\mathcal{R}$-vertices, respectively.
- If $R \in \mathcal{R}$ has event E affecting a location of $p \in \Sigma$ and a repair on $q \in \Sigma$, then we have an edge from p to R labelled by $+$ or $-$ depending on E leading to a violation of a negative or a positive literal, respectively, and an edge from R to q analogously labelled by $+$ or $-$ depending on whether the repair assignment on a q-location refers to a positive or negative literal.

Definition 7. Let (V, E) be the rule graph associated with a system $\mathcal{R}$ of atomic repair rules. A *trigger path* is a sequence $v_0, e_1, v_1', e_1', \ldots, e_\ell', v_\ell$ of vertices and edges such that $v_i \in \Sigma$ for all $i = 0, \ldots, \ell$, $v_i' \in \mathcal{R}$ holds for all $i = 1, \ldots, \ell$, e_i is an edge from v_{i-1} to v_i', and e_i' is an edge from v_i' to v_i with the same label as e_{i+1}.

For a trigger path assign to each vertex $v_i \in \Sigma$ a formula φ_i that is the negation of a clause such that the following conditions hold:

(i) φ_i implies the negation of the clause associated with v_{i+1}';
(ii) the application of the repair in v_{i+1}' will lead to a state satisfying φ_{i+1}.

Definition 8. A trigger path $v_0, e_1, v_1', e_1', \ldots, e_\ell', v_\ell$ is *critical* iff it has maximum length and $\models \neg(\varphi_0 \vee \varphi_\ell)$ holds.

Intuitively, a critical trigger path corresponds to a sequence of applications of atomic repair rules, each initialised by a violation of a clause, which finally lead to a state, where the intended update is undone. Thus, such trigger paths cannot define an extension of a repairable update set.

Proposition 4. *Let S be a consistent state with respect to a given set of invariants, let r be an assignment affecting a location with function symbol $p \in \Sigma$, and assume $S + \Delta_r(S) \models \varphi_0$, where φ_0 is a conjunction of literals. Then for a complete repair r_{rep} the sequence $r; r_{rep}$ is a consistent specialisation with respect to the conjunction of the given invariants iff there is no critical trigger path starting with a vertex p labelled by φ_0.*

The proof, which we cannot present here, basically follows the arguments used in the proof of [14, Prop.4]. If there were such a critical trigger path, it would lead to a state, in which one of the updates in $\Delta_r(S)$ would be discarded, which is contained in the definition of a critical trigger path. Conversely, if there is no such trigger path, r_{rep} will lead to a state satisfying φ_ℓ.

4.3 Locally Stratified Sets of Invariants

We now look for sufficient and necessary conditions on the set of clauses derived from a set of invariants that will allow us to obtain a complete repair satisfying the conditions in Proposition 4, i.e. the absence of critical paths. The intuition behind this procedure is that non-critical trigger paths give rise to cumulative updates, by means of which a repairable update set can be extended to achieve consistency, whereas critical trigger paths would undo some of the given updates. A complete repair then defines a consistent specialisation as shown by Proposition 4.

Definition 9. Let C be a set of clauses on Σ derived from a set of invariants. Then C is called *stratified* iff there is a partition $C = C_1 \cup \ldots \cup C_n$ with pairwise disjoint sets of clauses C_i called *strata* such that the following conditions are satisfied:

(i) If L is a negative (or positive, respectively) literal of some clause $c \in C_i$, then all clauses $c' \in C$ containing a positive (or negative, respectively) literal L' such that L and L' are unifiable also lie in stratum C_i.
(ii) All clauses c, c' containing unifiable literals L and L' either both positive or both negative must lie in different strata.

Stratified sets of clauses give rise to complete repairs without critical trigger paths.

Proposition 5. *Let C be a set of clauses on Σ derived from a set of invariants, and assume that C is stratified. Then there exists a complete repair r_{rep} such that the sequence $r; r_{rep}$ is a consistent specialisation with respect to the conjunction of the given invariants.*

For the proof, which we omit again due to space and time restrictions, we construct the set of all atomic repair rules from C. Then assume the existence of a critical trigger path initiated by r, i.e. r affects a location with function symbol $p \in \Sigma$, $S + \Delta_r(S) \models \varphi_0$ holds for a conjunction of literals, and the critical trigger path starting with a vertex p labelled by φ_0. Then φ_0 and φ_ℓ must contain a literal and its negation, respectively, and the corresponding rules must involve clauses in different strata, which leads to a contradiction.

Stratified sets of invariant clauses are sufficient for the construction of consistent specialisation, and stratification can be checked effectively and efficiently (see Algorithm 8 and Propositions 9 and 10 in [14]), but stratification is no necessary condition. We will now look at the weaker notion of local stratification, which will give us also a necessary condition.

Definition 10. Let C be a set of clauses on Σ derived from a set of invariants. A *labelled subsystem* consists of a literal L (the *label*), a subset $C' = \{c \in C \mid \rho_L(c) \text{ is defined}\}$, and a set of clauses $C'' = \{\rho_L(c) \mid c \in C'\}$ such that each clause $c \in C'$ can be written as the disjunction $\rho_L(c) \vee c'$ with $\models c' \Rightarrow L$.

Here $\rho_L(c)$ is defined iff the negation $\sim L$ does not occur in the clause c. Then $\rho_L(c)$ results from c by omission of the literal L, if the result contains at least two literals. Otherwise $\rho_L(c)$ is simply c. We call c' the *label part* and $\rho_L(c)$ the *label-free part* of the clause c. If L is understood from the context, we drop the subscript and write ρ instead of ρ_L.

A labelled subsystem (C', C'', L) is called *stratified* iff the set C'' is stratified in the sense of Definition 9 or locally stratified as defined below.

Definition 11. Let C be a set of clauses on Σ derived from a set of invariants. Then C is called *locally stratified* iff $C = C'_1 \cup \cdots \cup C'_n$ with stratified labelled subsystems (C'_i, C''_i, L_i) $(i = 1, \ldots, n)$ such that for each clause $c \in C'_i$ and each literal L occurring in its label part with respect to C_i there exists another j with $c \in C'_j$ and L occurring in its label-free part of c with respect to C_j.

Proposition 6. *Let C be a set of clauses on Σ derived from a set of invariants. Then there exists a complete repair r_{rep} such that the sequence $r; r_{rep}$ is a consistent specialisation with respect to the conjunction of the given invariants iff C is locally stratified.*

For the proof of sufficiency we can proceed analogously to the proof of Proposition 5, i.e. we take the local strata to define sets of atomic repair rules and build the union of these. Then the assumption of a critical trigger path leads again to a contradiction to the set of clauses being locally stratified.

Conversely, if we have a consistent specialisation $r; r_{rep}$ Proposition 4 implies the absence of critical trigger paths. From this it is possible to construct a local stratification (see also the proof of [14, Thm.12].

Example 7. Reconsider the invariants in Example 6. It is easy to see that this set is locally stratified leading to the atomic repair rules in the example without R_9 and R_{10}.

Finally, we can obtain the GCS of an assignment r by a choice between all possible complete repairs defined by local stratifications of the given set of invariants.

Theorem 1. *Let C be a locally stratified set of clauses on Σ derived from a set of invariants. Then the GCS of an assignment r with respect to the set of invariants is defined by the sequence $r; rep$, where rep is defined by the choice among all complete repairs r_{rep} defined by different local stratifications of C.*

5 Conclusion

In this paper we picked up the 20 year old theory of greatest consistent specialisation for consistency enforcement with respect to static invariants. We generalised the definition in the context of sequential Abstract State Machines with finer grained locations and bounded parallelism. Then we obtained generalised compositionality results with respect to the composition of ASM rules

and sets of invariants. The new theory supports the systematic *construction* of consistent specifications, which is not bound to ASMs.

However, we still excluded unbounded parallelism from our investigation. Extending the theory in this direction is an open, non-trivial task for continued research. We also emphasised only invariants expressed in first-order logic as well as only static invariants, though this covers the vast majority of specifications using state-based rigorous methods. Nonetheless, extensions to more complex invariants as well as a theory for transition or general dynamic invariants would make sense. For instance, in [6] the importance of higher-order logic constructs in formal methods was emphasised. Furthermore, in a database context many classes of static constraints have been studied [17]. These give rise to important classes of static invariants that could be used to derive a catalogue of GCSs for them, and this could be further extended to classes of invariants in other contexts giving even more support for the construction of consistent specifications.

References

1. Abrial, J.-R.: The B-Book - Assigning Programs to Meanings. Cambridge University Press, Cambridge (2005)
2. Börger, E., Schewe, K.-D.: A behavioural theory of recursive algorithms (2019). Submitted for publication
3. Börger, E., Stärk, R.: Abstract State Machines. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-642-18216-7
4. Cai, C., Sun, J., Dobbie, G.: B-repair: repairing B-models using machine learning. In: 23rd International Conference on Engineering of Complex Computer Systems (ICECCS 2018), pp. 31–40. IEEE Computer Society (2018)
5. Dijkstra, E.W., Scholten, C.S.: Predicate Calculus and Program Semantics. Texts and Monographs in Computer Science. Springer, New York (1990). https://doi.org/10.1007/978-1-4612-3228-5
6. Ferrarotti, F., González, S., Schewe, K.-D., Turull-Torres, J.M.: Systematic refinement of abstract state machines with higher-order logic. In: Butler, M., Raschke, A., Hoang, T.S., Reichl, K. (eds.) ABZ 2018. LNCS, vol. 10817, pp. 204–218. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-91271-4_14
7. Ferrarotti, F., Schewe, K.-D., Tec, L., Wang, Q.: A new thesis concerning synchronised parallel computing - simplified parallel ASM thesis. Theor. Comput. Sci. **649**, 25–53 (2016)
8. Ferrarotti, F., Schewe, K.-D., Tec, L., Wang, Q.: A complete logic for database abstract state machines. Log. J. IGPL **25**(5), 700–740 (2017)
9. Ferrarotti, F., Schewe, K.-D., Tec, L., Wang, Q.: A unifying logic for nondeterministic, parallel and concurrent abstract state machines. Ann. Math. Artif. Intell. **83**(3–4), 321–349 (2018)
10. Gurevich, Y.: Sequential abstract state machines capture sequential algorithms. ACM Trans. Comput. Logic **1**(1), 77–111 (2000)
11. Link, S., Schewe, K.-D.: Towards an arithmetic theory of consistency enforcement based on preservation of delta-constraints. Electr. Notes Theor. Comput. Sci. **61**, 64–83 (2002)
12. Nelson, G.: A generalization of Dijkstra's calculus. ACM Trans. Program. Lang. Syst. **11**(4), 517–561 (1989)

13. Schewe, K.-D.: Consistency enforcement in Entity-Relationship and object-oriented models. Data Knowl. Eng. **28**(1), 121–140 (1998)
14. Schewe, K.-D., Thalheim, B.: Limitations of rule triggering systems for integrity maintenance in the context of transition specifications. Acta Cybern. **13**(3), 277–304 (1998)
15. Schewe, K.-D., Thalheim, B.: Towards a theory of consistency enforcement. Acta Inf. **36**(2), 97–141 (1999)
16. Schmidt, J., Krings, S., Leuschel, M.: Repair and generation of formal models using synthesis. In: Furia, C.A., Winter, K. (eds.) IFM 2018. LNCS, vol. 11023, pp. 346–366. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-98938-9_20
17. Thalheim, B.: Dependencies in Relational Databases. Springer, Wiesbaden (1991). https://doi.org/10.1007/978-3-663-12018-6