A Hoare-Style Verification Calculus for Control State ASMs

Werner Gabrisch and Wolf Zimmermann
Martin-Luther-Universität Halle-Wittenberg, Institut für Informatik,
06099 Halle/Saale, Germany, gabrisch,zimmer@informatik.uni-halle.de

ABSTRACT
We present a Hoare-style calculus for control-state Abstract State Machines (ASM) such that verification of control-state ASMs is possible. In particular, a Hoare-Triple \( \{ \varphi \}, A \{ \psi \} \) for an ASM \( A \) means that if an initial state \( \exists \) satisfies the precondition \( \varphi \) and a final state \( \exists' \) is reached by \( A \), then the final state satisfies the postcondition \( \psi \). While it is straightforward to generalize the assignment axiom of the Hoare-Calculus to a single state transition, the composition of Hoare-Triples is challenging since typical programming language concepts are not present in ASMs.

1. INTRODUCTION
Abstract State Machines (ASM) are well-suited to specify state transition systems. Thus, they are frequently used to specify programming language semantics, machine semantics, technical systems, and algorithms. For example, the semantics of a program (in a programming language or machine language) or the semantics of an algorithm is given by an ASM, i.e., an initial state and a set of state transition rules specifying state transitions. However, from a more abstract viewpoint, programs or algorithms are specified on a more abstract level by giving preconditions and postconditions. For higher programming languages the Hoare-Calculus can be used for verifying programs [12]. However, its correctness w.r.t. the programming language semantics needs to be proven. For any new programming language, a new calculus needs to be designed and its correctness needs to be proven again. Furthermore, it is the machine program that is running on a computer. Thus, the compiler must also be verified [8, 14]. An alternative approach would be to verify directly on the level of ASMs that if the initial states satisfy a precondition \( \varphi \), then the final states satisfy a postcondition \( \psi \) provided they are reached. The advantage is that the correctness of the calculus needs to be proven only on the level of ASMs and – with the numerous works on ASMs programming language semantics (cf. e.g. [11, 2, 3, 13, 19, 6, 1, 18]) – a verification of the functionality of a program is possible independently of the particular programming language semantics or machine semantics, respectively.

In this paper we show for a special class of ASMs, the control-state ASMs, a verification approach such that the satisfaction of postconditions on final states can be verified provided that initial states satisfy a given precondition. As in traditional Hoare calculus, only partial correctness is being considered. In contrast to refinement-based approaches, we assume that an ASM is given (and not being developed). For the above applications of ASMs, this is not a real restriction as the works on programming language semantics or machine semantics are defined by control-state ASMs (control-states are nodes in the abstract syntax tree or the addresses of machine instructions, respectively). Furthermore, refinement approaches cannot be applied to bottom-up development or only partially be applied to middle-out development approaches. In particular the latter is frequently used in software engineering approaches.

Section 2 introduces ASMs. We consider here many-sorted ASMs and assume that the initial states and the static parts are specified algebraically. Section 3 presents the verification calculus for control-state ASMs. It is based on assertions that are satisfied whenever the ASM is in a certain control-state. However, it might be difficult to find such assertions. Section 4 shows a more constructive approach. It is based on standard compiler technology. Section 5 discusses related work and Section 6 concludes our work.

2. ABSTRACT STATE MACHINES
We first define the notion of Abstract State Machines. For more details, we refer to the Lipari-Guide [9], the ASM 1997 Guide [10] and to [4]. We present here a many-sorted version of ASMs. An Abstract State Machine (ASM) is a tuple \((\Sigma, \Phi_{\text{init}}, \text{Trans})\) where \(\Sigma\) is a (many-sorted) signature, \(\Phi_{\text{init}}\) is a set of \(\Sigma\)-formulas (the initial conditions), and \(\text{Trans}\) is a finite set of transition rules. The set of states is the set \(\text{Alg}(\Sigma)\) of \(\Sigma\)-algebras; \([\cdot]\) denotes the interpretation function of symbols of \(\Sigma\) and \(A_q\) denotes the interpretation of sort \(A\) of \(\Sigma\) (which is a set, the carrier set) in \(\Sigma\)-algebra \(q\). A state \(q\) is initial iff \(q\) is a model of \(\Phi_{\text{init}}\) in the sense of logic, denoted as \(q \models \Phi_{\text{init}}\).

Notation: Sorts are denoted by capital letters, function symbols always start with a lower-case letter. The symbol \(f : T_1 \times \ldots \times T_n \rightarrow T\) denotes a function symbol representing a total function. \(t'(t'/t)\) denotes the term \(t\) where each occurrence of subterm \(t'\) is substituted by term \(t''\). This can be extended as usual to \(\Sigma\)-formulas (here, only terms without bound variables are substituted by ground-terms).
An update $f(t_1, \ldots, t_n) := t$ is a pair of $\Sigma$-terms $f(t_1, \ldots, t_n)$ and $t$ of the same sort. A set $U$ of updates is inconsistent in a state $q$ iff there are two updates $f(t_1, \ldots, t_n) := t, f(u_1, \ldots, u_m) := u \in U$ such that $[t]_q = [u_1]_q, \ldots, [t]_q = [u_m]_q$ but $[t]_q \neq [u]_q$. The successor state $q'$ of a state $q$ w.r.t. to a consistent set of updates $U$ is the $\Sigma$-algebra $q' = (A, \cdot, \lambda)$ where the carrier sets are the same as those of $q$ (i.e., $A_q = A_{q'}$) and the interpretation is defined by

$$[f]_{q'}(a_1, \ldots, a_n) = \left\{ \begin{array}{l}
[i]_q \text{ if } a_1 = [t_1]_q, \ldots, a_n = [t_n]_q, \text{ such that } q = q' \\
[i]_q, \ldots, q = q, \text{ otherwise}
\end{array} \right. $$

A transition rule has the form $\varphi \text{ then } U \triangleright q$ Updates where $\varphi$ is a formula of many-sorted first-order logic (the guard) and $U$ is a set of updates. A transition rule fires in state $q$ iff $q \models \varphi$. Let $q$ be a state and if $\varphi_1 \text{ then } U_1, \ldots, \varphi_k \text{ then } U_k$ be the transition rules in state $q$ such that $q = q_1 \wedge \ldots \wedge q_k$. If $U \equiv U_1 \cup \ldots \cup U_k$ is consistent, the successor state $q'$ of $q$ is the successor state of $q$. If $q$ w.r.t. $U$. A state $q$ is final iff no transition rule fires in state $q$.

A run of an ASM is a finite or infinite sequence of states $q_1, q_2, q_3, \ldots$, for each reachable state $q_i$ in the infinite case, $q_i$ is the initial state and either the run is infinite or its last state is final. A state $q$ is reachable iff there is a finite run $q_0, q_1, \ldots, q_k$ where $q_0$ is initial. In the following discussions, we only consider reachable states.

Function symbols where $[\|]_q = [\|]_{q'}$ for all (reachable) states $q, q'$ of an ASM $A$ are called static. Otherwise they are called dynamic. A constant function symbol which is dynamic is called a dynamic constant. An ASM is sensible iff for any term $t$ and any state $q$, there is a ground-term $t'$ without dynamic function symbols such that $q = t = t'$. Sensibility means that in any state each element $x$ of a carrier set can be represented by a ground-term only using static function symbols. In this paper, we only consider sensible ASMs.

Example 1 Figure 1 shows an ASM computing the greatest common divisor. In all states, the carrier sets for sorts $Q, INT,$ and $BOOL$ are $Q \equiv \{0, 1, \ldots\}$, $INT \equiv \mathbb{Z}$, and $BOOL \equiv \{false, true\}$. Also, in all states the constants $-1, 0, 1, \ldots$ are interpreted as integers, $+$ and $-$ as integer addition or integer subtraction, respectively, and $>$ as the greater-than-comparison on integers. Thus, the states may have only different interpretations for the dynamic constants $cs, i, j, m_1, m_2,$ and $res$. Fig. 2 shows how two runs. The first run is complete since none of the guards of the transition rules is satisfied. The second run can only be completed to an infinite run.

A control state ASM is an ASM $A \equiv (\Sigma, \Phi_{init}, Trans)$ where (i) $\Sigma \equiv (S, F)$, has a sort $Q$ in $S$ of control states, (ii) the carrier set $A_Q$ of $Q$ is a non-empty and finite set denoting the set of possible states of $A$, (iii) there are constants $s_k \rightarrow Q \in F$ for each $x \in A_Q$, (iv) there is a constant $cs \rightarrow Q \in F$, (v) the initial states satisfy $\Phi_{init} := cs = s_0$ is called the initial control state, and all the transition rules have the form

$$if \text{ cs = s_k } \wedge \varphi \text{ then } U \triangleright q$$

where $U$ is a set of updates without an update on $cs$ ($U = \emptyset$ is allowed), $\varphi$ is a first-order formula that does not contain $cs$ and each state $q$ satisfies at most one guard. A control state $s$ of a control state ASM $A$ is final iff $q = cs = s$ for a final state $q$ of $A$.

Remark 1: For reasons of simplicity we assume that all transition rules have one of the forms:

$$if \text{ cs = s_k } \text{ then } U \triangleright q$$

and that there is exactly one final control state. Both assumptions are no restrictions for our purpose. By adding a new control state, any transition rule of a control state ASM can be divided into the two transition rules of the above forms. Furthermore, if there is more than one final control state, a new control state state $s_f$ can be added and we add a transition $if \text{ cs = s then } cs := s_f$ for each (previously) final state $s$. Then the states $s$ are not final and the state $s_f$ is final.

A control-state ASM $A \equiv (\Sigma, \Phi_{init}, Trans)$ can be represented as an edge-labelled directed graph $G_A \equiv (Q, E)$ (the control-state graph of $A$) where $Q$ is the set of control-states of $A$, $\rightarrow$ is a transition rule if $cs = s \wedge \varphi$ then $cs := s'$ where

$$s \rightarrow s' \in E$$

is there is a transition rule if $cs = s \wedge \varphi$ and $\triangleright s' \in Trans$.

Example 2 The ASM in Fig. 1 is a control-state ASM satisfying the assumptions of the above remark. The control-state $s_0$ is final. Fig. 3 shows the control-state graph of the ASM in Fig. 1. The final state is visualized as a square while the other states are visualized as circles.

3. A HOARE-STYLE VERIFICATION

APPROACH

The main goal is to verify ASMs, i.e., to show that if the initial states of an ASM $A \equiv (\Sigma, \Phi_{init}, Trans)$ satisfy a precondition $\varphi$, then any final state $q_f$ reached by an initial
state \( q \) with \( q \models \varphi \) satisfies a postcondition \( \psi \), i.e., \( q_f \models \psi \).

This property can be denoted as a Hoare-Triple \( \{ \varphi \}, A \{ \psi \} \).

In this section, we show that a verification of control-state ASMs can be purely based on the assignment axiom, a variant for the conditional rule, the sequentialization rule, on weakening postconditions and on strengthening preconditions.

**Theorem 1.** Let \( A = (\Sigma, \PhiInit, \text{Trans}) \) be a control-state ASM with control states \( Q, \varphi \) be a precondition for \( A \), and \( \chi_s, s \in Q \) be a set of first-order formulas satisfying the following conditions:

(i) \( q \models \varphi \Rightarrow \chi_s \) for each initial state \( q \) (with initial control-state \( s_0 \)).

(ii) If \( (cs = s) \) then \( U \{ cs := s' \} \in \text{Trans} \) then

\[ \chi_s \models (cs = s) \wedge \chi_s \text{, it holds } q' \models (cs = s') \Rightarrow \chi_{s'} \] for the successor state \( q' \).

(iii) If \( (cs = s) \wedge \rho \) then \( \{ cs := s' \} \in \text{Trans} \) for a first-order formula \( \rho \) then for any state \( q \) satisfying \( q \models (cs = s) \wedge \chi_s \), it holds \( q' \models (cs = s') \Rightarrow \chi_{s'} \).

**Proof.** We prove the claim by induction on \( i \). Let \( \{ q_i : 0 \leq i < n \} \), \( n \in \mathbb{N} \cup \{ \infty \} \), be a run of \( A \) such that \( q_0 \models \PhiInit \land \varphi \).

**Case 1:** \( i = 0 \): Since \( q_0 \models \PhiInit \land \varphi \), \( q_0 \) is an initial state and by definition of control-state ASMs there is a unique initial control-state \( s_0 \) such that \( q_0 \models cs = s_0 \). Thus, \( q_0 \models \varphi \) because \( \varphi \) is precondition. Together with (i), this implies \( q_0 \models cs = s_0 \Rightarrow \chi_{s_0} \). Hence, \( q_0 \models cs = s \Rightarrow \chi_s \) for all \( s \in Q \) since \( q_0 \not\models cs = s \Rightarrow \chi_s \).

**INDUCTION HYPOTHESIS:** \( q_{i-1} \models cs = s \Rightarrow \chi_s \) for all \( s \in Q \) for all \( i \geq 1 \).

**INDUCTION STEP:** We have to prove that \( q_i \models cs = s \Rightarrow \chi_s \) for all \( s \in Q \).

Since \( q_{i-1} \) is not final, there is a unique transition rule

\[ \text{if } cs = s \land \rho \text{ then } \{ cs = s' \} \in \text{Trans} \]

such that \( q_{i-1} \models cs = s \land \rho \) for a control state \( s \in Q \). By induction hypothesis and logic, \( q_{i-1} \models \chi_s \). Hence \( q_{i-1} \models cs = s \land \chi_s \land \rho \) and therefore, by (ii), \( q_i \models cs = s' \Rightarrow \chi_{s'} \).

Since \( q_i \not\models cs = s \) it also holds \( q_i \models cs = s \Rightarrow \chi_s \) for all \( s \in Q \setminus \{ s' \} \).

The formulas \( \chi_s \) satisfying conditions (i) and (ii) of Theorem 1 are called the control-state invariant for \( s \).

**Corollary 1.** If \( q \models (cs = s) \wedge (\chi_s \Rightarrow \psi) \) for all final states \( q \), then \( \{ \varphi \}, A \{ \psi \} \)

**Remark 2:** Theorem 1 basically corresponds to the sequentialization rule in the Hoare-calculus for program verification. However, with control-state ASMs, there is no direct sequentialization in the ASM specification language.

**Fig. 4** shows the control-state invariants of the ASM in Example 1. They are denoted close to the corresponding control-state.

It now remains to show how to prove that \( \chi_s \) is a control-state invariant for \( s \). Property Theorem 1(i) doesn’t include transition rules of the ASM. Hence, it can be proven purely by logical inference. Property Theorem 1(i) requires reasoning on a single state transition. Note that a single state transition can be viewed as an ASM with one transition rule. For a single transition rule, we write the Hoare-Triple \( \{ \varphi \} \text{ if } \rho \text{ then } U \{ \psi \} \) which means that \( q \models \varphi \land \rho \) implies \( q' \models \psi \) for any state \( q' \) that \( q \) is the successor of \( q' \).

For simplicity, we only consider rules of the two forms as defined in Remark 1.

**Lemma 1 (Conditional State Transition).** Let \( A = (\Sigma, \PhiInit, \text{Trans}) \) be a control-state ASM, \( \varphi \) and \( \psi \) be first-order formulas not containing the symbol \( cs \), and \( s, s' \in Q \) be control-states. Then

\[ \{(cs = s) \land \varphi \land \psi\} \text{ if } \rho \text{ then } U \{ \psi \} \] if \( cs = s \) and \( \psi \) then \( cs := s' \) for any state \( q' \) where \( q' \) is the successor of \( q \).

**Proof.** Straightforward as only \( [cs]_s \neq [cs]_s' \) and the rule fires only if \( q \models (cs = s) \land \varphi \).

**Lemma 2 (Update Rule).** Let \( A = (\Sigma, \PhiInit, \text{Trans}) \) be a Haskell control-state ASM, \( \varphi \), \( \psi \) be first-order formulas not containing the symbol \( cs \), and \( s, s' \in Q \) be control-states, and \( U\{f_1(t_1^{(i_1)}, \ldots, t_1^{(i_{l_1})}) := u_1, \ldots, f_k(t_k^{(i_1)}, \ldots, t_k^{(i_l)}) := u_k\} \).

**Proof.** Straightforward as only \( [cs]_s \neq [cs]_s' \) and the rule fires only if \( q \models (cs = s) \land \varphi \).

**Case 1:** \( u \) doesn’t contain a subterm \( f_i(t_i^{(k)}) \).

Then it is syntactically \( \sigma = u \) and the updates don’t change the interpretation. Hence, \( [u]_s = [u]_s' \).

**Case 2:** \( u = f_i(t_i^{(1)}, \ldots, t_i^{(l)}) \) for an \( 1 \leq i \leq k \) where the terms \( t_i^{(j)} \) are ground-terms such that \( [t_i^{(j)}]_s = [t_i^{(j)}]_s' \).
Then it is syntactically $u_i = u \sigma$. By definition of $q'$ it holds:
\[ [u_i]_{\sigma} = [f_i]_{\sigma}([i_i]_{\sigma}, \ldots, [i^{(n)}]_{\sigma}) \]
by definition of updates
\[ = [f_i]_{\sigma}([i'_i]_{\sigma}, \ldots, [i^{(n)}]_{\sigma}) \]
by definition of $i'_i$
\[ = [f_i(i'_i, \ldots, i^{(n)})]_{\sigma'} \]
since $i^{(n)}$ contain no dynamic functions

Hence, $[u \sigma]_{\sigma} = [f_i(i'_i, \ldots, i^{(n)})]_{\sigma'}$.

CASE 3 $w = q_i(t_1, \ldots, t_m)$ and $w \neq f_i(t^{(k)}, \ldots, t^{(k)})$ for any $i$.

Then $\sigma = q_i(t_1, \ldots, t_m)$ and
\[ [u \sigma]_{\sigma'} = [g]_{\sigma'}([t_1]_{\sigma}, \ldots, [t_m]_{\sigma}) \]
see above
\[ = [g]_{\sigma'}([t_1]_{\sigma}, \ldots, [t_m]_{\sigma})_{\sigma'} \]
this arguments
\[ = [g(t_1, \ldots, t_m)]_{\sigma'} \]
by induction hypothesis.

Using the result $[\sigma]_{\varphi} = [f]_{\varphi'}$ it can be shown by induction on the definition of formulas that $q = \varphi \Leftarrow q' = \varphi'$. □

The conditional state transition rule is similar to the Hoare-Calculus rule for if-statement and the update rule is similar to the assignment axiom of the Hoare-Calculus. As in the Hoare-calculus preconditions can be strengthened and postconditions can be weakened, i.e., the rule
\[ \varphi \Rightarrow \varphi' \quad \{ \varphi' \} A \{ \psi' \} \Rightarrow \psi \quad \text{(Pre,Post)} \]
is correct.

**Example 3** Consider in Fig. 3 the edge $s_1 \xrightarrow{i 
eq j} s_2$. By the conditional rule in Lemma 1 it holds:
\[ \{ n > 0 \land m > 0 \land \gcd(n, m) = \gcd(i, j) \} \quad \text{(Pre,Post)} \]
if $cs = s_1$ and $i \neq j$ then $cs = s_2$
\[ \{ n > 0 \land m > 0 \land \gcd(n, m) = \gcd(i, j) \land i \neq j \} \]
This proves the correctness of the state invariants on the state transition $s_1 \rightarrow s_2$.

Consider in Fig. 3 the edge $s_3 \xrightarrow{i = m + j} s_1$. Then by the update rule (Lemma 2) it holds:
\[ \{ n > 0 \land m > 0 \land \gcd(n, m) = \gcd(i - j, j) \} \]
if $cs = s_3$ then $cs = s_1, i := i - j$
\[ \{ n > 0 \land m > 0 \land \gcd(n, m) = \gcd(i, j) \} \]
By strengthening the precondition, the correctness of the state invariants on edge $s_3 \rightarrow s_1$ are proven:
\[ \{ n > 0 \land m > 0 \land \gcd(n, m) = \gcd(i - j, j) \} \]
if $cs = s_3$ then $cs = s_1, i := i - j$
\[ \{ n > 0 \land m > 0 \land \gcd(n, m) = \gcd(i, j) \} \]
Thus, the correctness of a control-state ASM can be proven by finding control-state invariants for each control-state and prove individually the corresponding Hoare-Triple for each edge in the control-state graph.

**4. FINDING CONTROL-STATE INVARIANTS**

For the practical verification, the main problem of the approach in Section 3 is to find the control-state invariants. Once these are obtained, their verification can be done by the update rule, conditional rule, and by strengthening preconditions or weakening postcondition. In this section, we show an approach that supports the finding of invariants. More concrete, we show for a subclass of control-state graphs (the reducible control-state graphs) that it is possible to re-discover structures as while-loops and conditionals. These notions and the corresponding results are directly taken from standard textbooks for compiler construction such as e.g. [7].

When loops, conditionals etc. can be identified, verification is not more difficult than a program verification with classical Hoare-Calculus, i.e., the main difficulty that remains is finding loop invariants.

Let $G = (V, E)$ be a directed graph with a single source, i.e., there is a unique vertex $u$ without predecessors and there is a path from $u$ to any $v \in V$. A vertex $v \in V$ dominates a vertex $w \in V$ (short: $v$ dom $w$) iff every path from $u$ to $w$ contains $v$. Obviously $u$ dominates each $v \in V$. Furthermore, the dominance relation is a tree, the *dominance tree*. It is straightforward to apply these notions to control-state graphs and control-states, respectively.

Let $T$ be a depth-first search tree of a control-state graph $G = (V, E)$. An edge $v \rightarrow w \in E$ is a *back-edge* if $w$ is an ancestor of $v \in T$. The control-state graph $G = (Q, E)$ is reducible iff $w$ dom $v$ for any back-edge $v \rightarrow w$ in any depth-first traversal of $G$. Otherwise $G$ is called irreducible.

**Example 4** In the control-state graph in Fig. 3, the dominance relation is the reflexive transitive closure of the following dominances: $s_0$ dom $s_1$, $s_1$ dom $s_2$, $s_2$ dom $s_3$, $s_2$ dom $s_4$, and $s_4$ dom $s_6$. Fig. 5(a) shows this relation as a dominance tree. The control-state graph in Fig. 3 is reducible. For example, the sequence $s_0, s_1, s_2, s_3, s_4, s_5, s_6$ is a depth-first traversal of this graph. There are two back-edges: $s_3 \rightarrow s_1$ and $s_4 \rightarrow s_1$. As Fig. 5(a) demonstrates, $s_1$ dominates $s_3$ as well as $s_4$. Fig. 6 shows an irreducible control-state graph of an ASM and its dominance tree. Here, $s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7$ is a depth-first traversal of this graph. Thus, there is back-edge $s_3 \rightarrow s_5$ but $s_2$ doesn’t dominate $s_5$.

For reducible control-state graphs, it is possible to identify structures such as conditionals and while-loops. E.g. the control-state graph in Fig. 3 has a conditional consisting of the subgraph induced by states $\{s_4, s_3, s_4\}$. This sub-graph is the body of a loop which is defined by the sub-graph

**Figure 5: Dominance Tree and Regions of the Control-State Graph in Fig. 3**

**Figure 6: An Irreducible Control-State Graph and Its Dominance Tree**
induced by the states \( \{s_1, s_2, s_3, s_4\} \) Note that this loop can only be entered by passing control-state \( s_1 \) and \( i \neq j \) is the loop condition. In contrast, for irreducible control-state graphs, it is not possible to identify such structures. Consider for example the sub-graph of Fig. 6 induced by \( \{s_2, s_3, s_4, s_5\} \). This cycle can be entered by passing control-state \( s_2 \) or by passing control-state \( s_4 \).

This observation can be formalized by the notion of a region. A region of a control-state graph \( G = (Q, E) \) is a sub-graph \( R \cong (Q', E') \) of \( G \) satisfying the following properties:

(i) There is a unique control-state \( s_R \in Q \) such that each path in \( G \) from the initial control-state \( s_0 \) to any \( s \in Q' \) contains \( s_R \), i.e., \( s_R \) dominates each \( s \in Q' \).

(ii) If there is a path from a control-state \( s \in Q \) to a control-state \( s' \in Q' \) without \( s_R \) then \( s' \in Q' \).

(iii) \( \{s \to s' \in E : s, s' \in Q', s' \neq s_R\} \subseteq E' \subseteq \{s \to s' \in E : s, s' \in Q'\} \), i.e., if the region is a loop then its body (without the back-edges) is also a region.

Example 5 Fig. 8(b) shows the regions of the control-state graph in Fig. 3. Note that each sub-graph \( R_i \cong (\{s_i\}, \emptyset) \) \( i = 0, \ldots, 6 \) is a region. The region \( R_7 \) corresponds to a conditional, the region \( R_8 \) to a while-loop with body \( R_5 \). Obviously, the whole control-state graph is also a region.

For reducible control-state graphs, regions are always either nested or disjoint. Let \( A = (\Sigma, \Phi_{init}, Trans) \) be a control-state ASM with control states \( Q, G = (Q, E) \) its control-state graph, and \( R = (Q_R, E_R) \) be a region of \( G \). Intuitively, each region defines a control-state ASM. Initial states are in control-state \( s_R \) and satisfy the invariant \( \chi_{s_R} \), the transition rules for the edges in the region are taken from \( A \), and the transition rules for the edges leaving a region are taken from \( A \) by just replacing updates \( cs := s' \) by \( cs := s' \) for a new final state \( s_f \notin Q \). Formally, \( A_R \cong (\Sigma, cs := s_R \land \chi_{s_R}, Trans_R) \) where

if \( cs = s \land s' \in E_R \) (if \( cs = s \) then \( \{cs := s'\} \cup \emptyset \in Trans_R \))

iff either \( s \xrightarrow{\delta} s' \in E_R \) (s \( \xrightarrow{\delta} \) \( s' \in E_R \)) or the following conditions are satisfied: \( s' = s_R \), \( s \in Q_R \), and there is a \( s' \) in \( Q \) such that \( s \xrightarrow{\delta} s'' \in E \setminus E_R \) (s \( \xrightarrow{\delta} \) \( s'' \in E \setminus E_R \)).

Fig. 7 shows the transition rules for the control-state ASMs induced by the regions \( R_3, R_5, R_6, R_7 \). Their initial control-state is \( s_3, s_2, s_2, s_1 \) respectively.

A region \( R \) with initial control-state \( s_R \) is a natural loop if it contains back-edges \( s \to s_R \). E.g., in Fig. 5(b) region \( R_6 \) is a natural loop containing region \( R_7 \). For natural loops, it is possible to define verification rules analogous to the while-rule of the Hoare-Calculus. Note that any region \( R \) that is a natural loop contains a sub-region \( R' \) with the same control-states but without the back-edges. The initial control-states of \( R \) and \( R' \) are identical, i.e., \( s_R = s_R' \), and - by Remark 1 - the only edges leaving \( R \) have the form \( s_i \xrightarrow{\delta} s, i = 1, \ldots, k \) , cf. Fig. 8(a) \( s_i = s_R \) is possible. Thus, the corresponding verification rule is:

\[
\{\chi\}A_{R'} \{\chi'\} \quad \text{(While)}
\]

Similarly, we have an analogy to a sequentialization rule, i.e., if a region \( R \) consists of two disjoint regions \( R', R'' \) such that there are only edges \( s \xrightarrow{\delta} s_R \) leaving region \( R' \), cf.

If \( cs = s_{\alpha} \) then \( \{cs := s_{\beta}, i := i - j\} \)

(a) Region \( R_{\alpha} \)
if \( cs = s_{\alpha} \land \alpha > j \) then \( \{cs := s_{\beta}\} \)
if \( cs = s_{\alpha} \land \alpha > j \) then \( \{cs := s_{\beta}\} \)
if \( cs = s_{\alpha} \) then \( \{cs := s_{\beta}, i := i + j\} \)
if \( cs = s_{\alpha} \) then \( \{cs := s_{\beta}, i := i + j\} \)

(b) Region \( R_{\beta} \)
if \( cs = s_{\alpha} \land \alpha = j \) then \( \{cs := s_{\beta}\} \)
if \( cs = s_{\alpha} \land \alpha = j \) then \( \{cs := s_{\beta}\} \)
if \( cs = s_{\alpha} \land \alpha > j \) then \( \{cs := s_{\beta}\} \)
if \( cs = s_{\alpha} \land \alpha > j \) then \( \{cs := s_{\beta}\} \)

(c) Region \( R_0 \)
if \( cs = s_{\alpha} \land \alpha = j \) then \( \{cs := s_{\beta}\} \)
if \( cs = s_{\alpha} \land \alpha > j \) then \( \{cs := s_{\beta}\} \)
if \( cs = s_{\alpha} \land \alpha = j \) then \( \{cs := s_{\beta}\} \)
if \( cs = s_{\alpha} \land \alpha > j \) then \( \{cs := s_{\beta}\} \)

(d) Region \( R_0 \)

Figure 7: Control-State ASMs for Regions in Fig. 5(b)

Figure 8: Region Structure for the Rules (While), (Seq), and (Cond)

Fig 8(b), then they are sequentially executed. By Remark 1 these edges can only be labeled with update sets. Then, the following rule is correct

\[
\{\varphi\}A_R(\rho) \{\varphi, \rho\} \quad \text{(Seq)}
\]

Finally, we have an analogy to the if-rule in the Hoare-Calculus, if a region \( R \) contains sub-regions \( R_0, R_1, \ldots, R_k \), \( k \geq 2 \), and there are edges \( s_i \xrightarrow{\delta} s_R \) leaving region \( R_0 \) to the initial control states of region \( R_i \), \( i = 1, \ldots, k \), cf. Fig. 8(c). By Remark 1, these edges can only be labelled with formulas and the corresponding verification rule is:

\[
\{\chi_1 \land \phi_1\}A_{R_0}(\rho) \cdots \{\chi_k \land \phi_k\}A_{R_k}(\rho)
\]

\[
\{\chi_1 \land \phi_1\}A_{R_0}(\rho)
\]

The correctness of these rules follow directly from the results along Section 3

With these rules, it is possible to verify control-state ASMs along the hierarchy of regions:

Example 6 Consider the invariants \( \chi_1 \) in Fig. 4. Suppose, the invariant for the natural loop \( R_0 \) is

\[
\chi_1 \triangleq n > 0 \land m > 0 \land \gcd(i, j) = \gcd(n, m)
\]

Then, the other control-state invariants can be directly obtained (on some places preconditions are strengthened or postconditions are weakened, indicated by Pre and Post, respectively, the formulas \( \chi_0 \) are the state invariants of the control states \( s_i \) in Fig. 4):

\[
\begin{align*}
(1) & \quad \chi_{X_1} A_{R_0} \chi_{X_1} \\
(2) & \quad \chi_{X_2} A_{R_0} \chi_{X_1} \\
(3) & \quad \chi_{X_3} A_{R_0} \chi_{X_1} \\
(4) & \quad \chi_{X_4} A_{R_0} \chi_{X_1} \\
(5) & \quad \chi_{X_5} A_{R_0} \chi_{X_1} \\
(6) & \quad \chi_{X_6} A_{R_0} \chi_{X_1} \\
(7) & \quad \chi_{X_7} A_{R_0} \chi_{X_1} \\
(8) & \quad n > 0 \land m > 0 \land A_{R_0}(\chi_{X_1}) \\
(9) & \quad n > 0 \land m > 0 \land A_{R_0}(\chi_{X_1}) \\
(10) & \quad \chi_{\varphi_0} A_{R_0}(\chi_{\varphi_0}) \\
(11) & \quad \chi_{\varphi_0} A_{R_0}(\chi_{\varphi_0})
\end{align*}
\]

by Lemma 2, (Pre)
by Lemma 2, (Pre)
by (1), (2), and (Cond)
by Lemma 1
by (3), (4) and (Seq)
by (6), (While)
by (6), (Post)
by Lemma 2, (Pre)
by (8), (7), and (Seq)
by (9), (10), and (Seq)
5. RELATED WORK

Winter uses a model-checking approach for verifying ASMs [21, 5, 23, 22, 24, 7]. The main idea is to abstract ASMs to finite state machines that can be checked automatically by a model-checker such as e.g. SMV. The specification language for the model-checkers is similar to the ASM transition rules. This work only allows dynamic constants of a finite domain and therefore may restrict the expressiveness of the assertions. E.g. checking the greatest common divisor example would become difficult if the pre- and postconditions consider all natural numbers and not just a finite subset of it.

Stark develops a verification calculus for general ASMs [20, 4] based on temporal logic. Therefore it is possible to verify general temporal behaviour. With their approach, it must be proven that control-state invariants are always satisfied. The approach in this paper enables to prove control-state invariants by Hoare-like proof rules individually for each edge in the control-state graph – independent of the proof for the other edges. Thus, correctness proofs only require to check the assignment axiom which is more simple than a general proof in Stärks verification calculus.

Schellhorn provides a refinement-based approach for verification of ASMs [17, 15, 16]. His approach is better suited for a top-development of ASMs but it cannot be applied to bottom-up or middle-out approaches that are also frequently used in Software Engineering.

6. CONCLUSIONS

We have shown that a Hoare-style verification is possible for control-state ASMs. The approach is surprisingly simple: find control-state invariants for each control-state i.e., assertions that are satisfied whenever the ASM is in control-state i and verify individually the correctness of the state transitions for each edge in the control-state graph. By using classical compiler technology, finding control-state invariants for reducible control-state graphs is not more difficult than finding loop invariants for while-loops in the standard Hoare-Calculus. Irreducible control-state graphs have a tight relation to “spaghetti”-code. In the future, we investigate well-known compiler techniques for transforming irreducible control-flow graphs into reducible control-flow graphs for the verification of general control-state graphs.

Some variants of ASMs have forall-updates and non-deterministic choose-updates. In the future, we will extend our approach towards these kinds of updates.

Section 4 shows that well-known compiler technology can be applied to control-state ASMs. In particular, it seems that results from program analysis can be applied to control-state ASMs. Reducible control-state graphs could be automatically transformeded Turbo-ASMs (see [4] for Turbo-ASMs), since they define classical control-structures.

In future, we will implement automatic invariant checking and invariant finding based on the approaches in this paper. Furthermore, we want to apply this approach to control-state ASMs derived from machine-programs and programs in higher-order programming languages, respectively, induced by the language semantics. In particular, this becomes interesting for machine languages, because their control-state ASMs are likely to be irreducible due to jump instructions.

7. REFERENCES