A Polynomial-Time Slicing Algorithm

Antje Nowack

Mathematische Grundlagen der Informatik,
RWTH Aachen, D-52056 Aachen, Germany,
email: nowack@informatik.rwth-aachen.de

Abstract. In an earlier work, we have introduced slicing for Abstract State Machines (ASMs). The idea of this concept is analogous to the one of program slicing which is an established technique for extracting statements from a program that are relevant for its behaviour at a given point of interest. These statements form again a syntactically correct program called a slice.

We have identified a class of ASMs (so-called guarded ASMs) for which a minimal slice is computable. Though this class is very expressive, the guardedness condition still remains a restriction. Furthermore, the resulting algorithm has a rather high worst-case runtime though an implementation would allow many optimizations such that it behaves much better in many cases.

In this work, we present a new algorithm for slicing ASMs. There are no restrictions on the ASMs, the algorithm is much faster than the first one and it always yields a slice. Though this slice is not minimal in all cases, we can guarantee minimality of the slice under certain very reasonable assumptions.

1 Introduction

Abstract State Machines (ASMs) are a formal model of computation as well as an executable specification language used in practice. This suggests to combine practically established concepts with theoretically founded approaches for validating ASM specifications.

In [4] we have suggested the notion of slicing ASMs. The motivation and the idea of the definition can be shortly described as follows. As writing specifications is usually not an error-free process we are quite often faced with debugging. In order to debug or to modify an ASM specification we are most often not interested in its complete behaviour but only in its behaviour at a given point of interest. This means that we do not observe each complete program state but only a part of it. Consider for instance the case of debugging. When an error is observed, the programmer tries to extract that part of the ASM which is responsible for the erroneous behaviour. This set of statements might be much smaller than the original one and therefore, it might be much easier to catch and correct the error(s) if he knows this set. An ASM whose program is a subprogram of the original one consisting of these statements is called a slice. The smaller such a slice is the better as the understandability increases with a decrease of the size. Consequently, we are interested in a minimal slice.

A minimal slice is not computable for arbitrary ASMs. This can be easily proven via a reduction of the satisfiability problem for first-order logic. Nevertheless, it would be very useful to have an algorithm computing a minimal slice for a huge class of ASMs. Therefore, in an earlier work, we have posed the question whether there exists an expressive class of ASMs for which a minimal slice is computable. In [4] and [3], we have identified the class of guarded ASMs to be such a class. The resulting algorithm has a rather high worst-case runtime though an implementation would allow many optimizations such that it behaves much better in many cases. Furthermore, it can not be adapted so that it works for arbitrary ASMs. It is even not possible to guarantee that it provides a slice in case that the input is not guarded. Therefore, it is straightforward to ask whether there are other ways to approximate an algorithm computing a minimal slice. Another approach is to decrease the claims on the output of the algorithm instead of restricting the input. One such possibility is to quit requiring minimality of the output. More concrete, this means to develop an algorithm that outputs a slice for every ASM. Furthermore, this slice should be as small as possible in order to approximate a minimal slice and there should be a huge class of ASMs for which the output is minimal. Finally, we require the runtime of the algorithm to be as low as possible.

In this work, we present a new algorithm for slicing ASMs. This algorithm computes a slice for arbitrary ASMs. Though this slice is not minimal in all cases, we can guarantee minimality of the slice under certain very reasonable assumptions. Furthermore, the worst-case runtime is much lower than the one of the algorithm presented in [4]. It is quadratic in the size of the ASM.

Outline. First, we give a short introduction to slicing. We then proceed with some definitions preparing the aimed algorithm. Afterwards, we present a new algorithm for slicing ASMs and prove that the result of the algorithm is always a slice and that we can guarantee minimality under certain reasonable assumptions. Finally, we analyze the runtime of the algorithm and give some extensions, improvements and remarks.

2 A Short Introduction to Slicing

As this paper is written for the ASM community, we assume that the reader is familiar with ASMs. For a detailed introduction to ASMs see e.g. [1], [2].

If no restriction is given then the guard of an if-clause, forall-rule or choice-rule may be any first-order formula (possibly containing quantifiers).

In this section, we shortly repeat the definition of the notion of slice for ASMs. A state of an ASM is defined by the content of its locations. Locations can be described by atomic formulae.

Definition 1 (Slicing Criterion) *A slicing criterion is a set of atomic formulae (possibly containing variables).*

ψ	$\mathbf{sub}(\psi)$
atomic	$\{\psi, \text{true}, \text{false}\}$
$\neg\varphi$	$\{\psi\} \cup \text{sub}(\varphi) \cup \{\neg\vartheta : \vartheta \in \text{sub}(\varphi)\}$
$\varphi_1 \vee \varphi_2$	$\{\psi\} \cup \text{sub}(\varphi_1) \cup \text{sub}(\varphi_2) \cup \{\vartheta_1 \vee \vartheta_2 : \vartheta_1 \in \text{sub}(\varphi_1), \vartheta_2 \in \text{sub}(\varphi_2)\}$
$\varphi_1 \wedge \varphi_2$	$\{\psi\} \cup \text{sub}(\varphi_1) \cup \text{sub}(\varphi_2) \cup \{\vartheta_1 \wedge \vartheta_2 : \vartheta_1 \in \text{sub}(\varphi_1), \vartheta_2 \in \text{sub}(\varphi_2)$
$\exists \bar{x}\ \varphi$	$\{\psi\} \cup \text{sub}(\varphi) \cup \{\exists \bar{x}\ \vartheta : \vartheta \in \text{sub}(\varphi)\}$
$\forall \bar{x}\ \varphi$	$\{\psi\} \cup \text{sub}(\varphi) \cup \{\forall \bar{x}\ \vartheta : \vartheta \in \text{sub}(\varphi)\}$

Fig. 1. Formal definition of sub(ψ) for an FO-formula ψ

$\mathbf{\Pi}$	$\mathbf{sub}(\mathbf{\Pi})$
Skip	$\{$**Skip**$\}$
$s := t$	$\{$**Skip**$, \mathit{\Pi}\}$
if φ **then** $\mathit{\Pi}'$ **endif**	$\{$**Skip**$, \mathit{\Pi}\} \cup \text{sub}(\mathit{\Pi}') \cup$ $\{$**if** ϑ **then** $\tilde{\mathit{\Pi}}$ **endif** : $\tilde{\mathit{\Pi}} \in \text{sub}(\mathit{\Pi}'), \vartheta \in \text{sub}(\varphi)\}$
do-in-parallel $\mathit{\Pi}_1\ \mathit{\Pi}_2$ **enddo**	$\{$**Skip**$, \mathit{\Pi}\} \cup \text{sub}(\mathit{\Pi}_1) \cup \text{sub}(\mathit{\Pi}_2) \cup$ $\{$**do-in-parallel** $\mathit{\Pi}_1'\ \mathit{\Pi}_2'$ **enddo** : $\mathit{\Pi}_1' \in \text{sub}(\mathit{\Pi}_1), \mathit{\Pi}_2' \in \text{sub}(\mathit{\Pi}_2)\}$
import v R' **endimport**	$\{$**Skip**$, \mathit{\Pi}\} \cup \text{sub}(R') \cup$ $\{$**import** $v\ \tilde{R}$ **endimport** : $\tilde{R} \in \text{sub}(R')\}$
forall $\bar{x} : \varphi(\bar{x})$ **do** R' **endforall**	$\{$**Skip**$, \mathit{\Pi}\} \cup \text{sub}(R') \cup$ $\{$**forall** $\bar{x} : \vartheta(\bar{x})$ **do** $\tilde{R}$ **endforall** : $\tilde{R} \in \text{sub}(R'), \vartheta \in \text{sub}(\varphi)\}$
choose $\bar{x} : \varphi(\bar{x})$ **do** R' **endchoose**	$\{$**Skip**$, \mathit{\Pi}\} \cup \text{sub}(R') \cup$ $\{$**choose** $\bar{x} : \vartheta(\bar{x})$ **do** $\tilde{R}$ **endchoose** : $\tilde{R} \in \text{sub}(R'), \vartheta \in \text{sub}(\varphi)\}$

Fig. 2. Formal definition of sub($\mathit{\Pi}$) for an ASM $\mathit{\Pi}$

A slice of an ASM $\mathit{\Pi}$ results from $\mathit{\Pi}$ by removing parts from $\mathit{\Pi}$ and resulting in a syntactically correct ASM again. The formal definition proceeds via the notion of subprogram. The formal definition of he notions of *subformula* of a first-order formula and *subrule* of an ASM is displayed in figure 1 and figure 2 where sub(ψ) abbreviates the set of subformulae of a FO-formula ψ and sub($\mathit{\Pi}$) abbreviates the set of subrules of an ASM $\mathit{\Pi}$. A subrule without free variables is called a *subprogram*. $L(\mathcal{A}, S)$ provides the possibility to conclude from a state and a slicing criterion (and therefore from atomic formulae) to the corresponding locations.

Definition 2 *Let $\mathcal{A}$ be a state and S be a slicing criterion.*
$L(\mathcal{A}, S)$ denotes the set of all locations $l = (Q, a_1 \ldots a_r)$ in $\mathcal{A}$ such that the atom $Qa_1 \ldots a_r$ is an instance of an atomic formula $\alpha \in S$.

We define the notion of S-equivalence on a state $\mathcal{A}$ in order to be able to define S-equivalence in general where S is a slicing criterion.

Definition 3 (S-equivalence) *Let $\mathit{\Pi}$, $\mathit{\Pi}'$ be ASMs and S a slicing criterion.*

1. *For a state $\mathcal{A}$ and a set L of locations of $\mathcal{A}$, let $\mathcal{A}|_L$ be the set of all pairs (l, a) where $l \in L$ and a is the content of l in $\mathcal{A}$.*
2. *Given a state $\mathcal{A}$ of Π resp. Π'. If for every run of Π $\mathcal{A} = \mathcal{A}_0 \vdash_\Pi \mathcal{A}_1 \vdash_\Pi \cdots \vdash_\Pi \mathcal{A}_i \vdash_\Pi \mathcal{A}_{i+1} \vdash_\Pi \ldots$ there is a run of Π' $\mathcal{A} = \mathcal{A}_0 \vdash_{\Pi'} \mathcal{A}'_1 \vdash_{\Pi'} \cdots \vdash_{\Pi'} \mathcal{A}'_i \vdash_{\Pi'} \mathcal{A}'_{i+1} \vdash_{\Pi'} \ldots$ such that $\mathcal{A}_i|_{L(\mathcal{A}_i, S)} = \mathcal{A}'_i|_{L(\mathcal{A}'_i, S)}$ for all $i \in \mathbb{N}$ then Π' S-captures Π on $\mathcal{A}$ ($\Pi \leq^{\mathcal{A}}_S \Pi'$).*
3. *Given a state $\mathcal{A}$ of Π resp. Π'. Π and Π' are called S-equivalent on $\mathcal{A}$ ($\Pi \equiv^{\mathcal{A}}_S \Pi'$) if $\Pi \leq^{\mathcal{A}}_S \Pi'$ and $\Pi' \leq^{\mathcal{A}}_S \Pi$.*
4. *If $\Pi \equiv^{\mathcal{A}}_S \Pi'$ for all states $\mathcal{A}$ of Π resp. Π' then Π and Π' are called S-equivalent ($\Pi \equiv_S \Pi'$).*

For deterministic ASMs, the notion of S-capturing and S-equivalence coincide.

Informally, one could formulate the condition for $\Pi \equiv_S \Pi'$ as follows. Π' behaves on all locations $(R, \bar{a})$ in exactly the same way as Π where the atom $R\bar{a}$ is an instance of an atomic formula $\alpha \in S$.

Definition 4 (Slice) *Let Π be an ASM-program and S be a slicing criterion. An S-slice of Π is an ASM-program $\Pi_S \in \mathrm{sub}(\Pi)$ such that $\Pi \equiv_S \Pi_S$.*

In the rest of this work, we refer to the size of a formula as the number of symbols in the string representing it where variables, constant and relation symbols are counted as one. For a formula φ, $|\varphi|$ denotes its size.
The size of an ASM-rule is defined analogously but keywords are counted as one (then, endif, endforall, endchoose, enddo are not counted separately), and the size of the guards is defined by the above definition of the size of a formula. For an ASM Π, $|\Pi|$ denotes its size.

Definition 5 (Minimal Slice) *A minimal slice of an ASM-program Π and a slicing criterion S is an S-slice Π_S of Π such that there is no S-slice $\tilde{\Pi}$ of Π with $|\tilde{\Pi}| < |\Pi_S|$.*

In [4], we have proved that a minimal slice does always exists but it is not unique. Furthermore, a minimal slice is not computable in the general case. The reason is that satisfiability of first-order logic can be directly reduced to to the computation of a minimal slice in the general case.

3 The Absolute Guard

The basic idea of the algorithm presented in this work is to identify all atoms that may influence the evaluation of the relevant locations. This is mainly done by determining the condition for executing an update-rule. For this reason, we define the absolute guard of a rule in a program. Essentially, the absolute guard of a rule in a deterministic ASM is an ordered list of all guards of conditional rules and forall-rules in whose scope the rule is situated.

Definition 1 (Absolute Guard).

1. *For a list of items l and and item x, let $x \circ l$ be the list resulting by inserting x in the front of l.*

2. *Let Π be a deterministic ASM not using* **import** *and R be a rule occurring only once in Π and $R \neq$* **Skip**.
   The absolute guard of R in Π is defined (by induction on Π) as follows.

   – *If Π is an update rule $s := t$ then the rule R must be equal to $s := t$ and* $\mathrm{absGuard}(\Pi, R) := ()$.

   – *If Π is a parallel composition of the form* **do-in-parallel** Π_1 Π_2 **enddo** *then* $\mathrm{absGuard}(\Pi, R) := \mathrm{absGuard}(\Pi_i, R)$ *where R occurs in Π_i, $i \in \{1, 2\}$.*

   – *If Π is a conditional rule of the form* **if** α **then** Π' **endif** *then we define* $\mathrm{absGuard}(\Pi, R) := \alpha \circ \mathrm{absGuard}(\Pi', R)$.

   – *If Π is a forall rule of the form* **forall** $\bar{x} : \alpha(\bar{x})$ **do** Π' **endforall** *then* $\mathrm{absGuard}(\Pi, R) := (\alpha(\bar{x}), \bar{x}) \circ \mathrm{absGuard}(\Pi', R)$

The restriction that the rule R occurs only once in Π is not really a restriction as one can always easily construct an equivalent program in which every subrule occurs at most once. If we do not want to carry out a transformation then we can modify the definition by not defining the absolute guard of a rule R but of an occurrence of a rule R. An occurrence of a rule is unique.

In the remainder of this section, we investigate the information that is provided by an absolute guard. In order to simplify the formulation, we define the two mappings conj and impl. Both map absolute guards to FO-formulae.

Definition 6 *Let a be an absolute guard in an ASM. The formulae* $\mathrm{conj}(a) \in \mathrm{FO}, \mathrm{impl}(a) \in \mathrm{FO}$ *are defined by induction on a.*

1. *conj is defined by*
   – $\mathrm{conj}(()) := \mathrm{true}$
   – *if φ_0 is an FO-formula*

$$\mathrm{conj}((\varphi_0, \varphi_1, \ldots, \varphi_{n-1})) := \varphi_0 \wedge \mathrm{conj}((\varphi_1, \ldots, \varphi_{n-1}))$$

   – $\mathrm{conj}((((\varphi_0', \bar{x})\varphi_1, \ldots, \varphi_{n-1})) := \exists \bar{x}(\varphi_0 \wedge \mathrm{conj}((\varphi_1, \ldots, \varphi_{n-1})))$

2. *impl is defined by*
   – $\mathrm{impl}(()) := \mathrm{false}$
   – *if φ_0 is an FO-formula then*

$$\mathrm{impl}((\varphi_0, \varphi_1, \ldots, \varphi_{n-1})) := \varphi_0 \rightarrow \mathrm{impl}((\varphi_1, \ldots, \varphi_{n-1}))$$

   – $\mathrm{impl}((((\varphi_0', \bar{x})\varphi_1, \ldots, \varphi_{n-1})) := \forall \bar{x}(\varphi_0 \rightarrow \mathrm{impl}((\varphi_1, \ldots, \varphi_{n-1})))$

Essentially, the following information is provided by an absolute guard.

Within one execution step of an ASM-program Π, the rule R appearing in Π is executed if, and only if, $\mathrm{conj}(\mathrm{absGuard}(\Pi, R))$ is satisfied. Therefore, if $\mathrm{conj}(\mathrm{absGuard}(\Pi, R))$ is not satisfiable then R is never executed. If we replace R by **Skip** in Π then we obtain a program which is equivalent to Π.

If, for any rule R appearing in the program Π, conj(absGuard(Π, R)) is valid then R is executed in every execution step of Π (independently from the state assuming that the complete update set is consistent). Consequently, the execution of R within an execution step of Π does not depend on the valuation of the guards. If we replace R by Skip in Π (leading to $\Pi[R/\mathbf{Skip}]$) and then construct **do-in-parallel** $\Pi[R/\mathbf{Skip}]$ R **enddo** in case that free(R) $\neq \emptyset$ and else **do-in-parallel** $\Pi[R/\mathbf{Skip}]$ **forall** $\bar{x}$: true **do** R **endforall** **enddo** (where free(R) = free($\bar{x}$)) then we obtain a program that is equivalent to Π.

Consider the rule **if** α **then** R **endif** (inside the program Π). If the formula impl(absGuard(Π, R)) is valid then we obtain a program equivalent to Π by replacing **if** α **then** R **endif** by R in Π.

Consider the rule **forall** $\bar{x}$: α **do** R **endforall** (in the program Π). If the formula impl(absGuard(Π, R)) is valid then we obtain a program equivalent to Π by replacing **forall** $\bar{x}$: α **do** R **endforall** by **forall** $\bar{x}$: true **do** R **endforall** in Π.

Let Π be a program and $s := t$ appear in Π. If impl(absGuard($\Pi, s := t) \circ (s \leftrightarrow t)$) is valid then replacing the update-rule $s := t$ by **Skip** in Π leads to a program that is equivalent to Π.

4 *Π*-Graph and Distance of Atoms

In this section, we introduce the notion of a Π-graph where Π is an arbitrary ASM. It already reflects the basic idea of the polynomial-time slicing algorithm and we will use it later in the proofs of the main properties of the algorithm.

In some sense, a Π-graph can be considered as a dependency graph containing the atoms appearing in Π. The set of nodes consists of all atoms appearing in the Π. There is an edge from a to b iff b is unifiable with an atom in absGuard(Π, R) for an update rule R whose lefthand side is unifiable with a. Consequently, the valuation of b may directly influence the update of a. An atom b can only influence the evaluation of an atom c if there is a path from a to c in the Π-graph.

Definition 7 (Π-graph) *For an ASM Π, the Π-graph is the graph $G_\Pi = (V_\Pi, E_\Pi)$ where*

- *V_Π is the set of all atoms appearing in Π*
- *for two nodes $a, b \in V_\Pi$, $(a, b) \in E_\Pi$ holds iff b is unifiable with an atom in* absGuard(Π, R) *for an update rule R whose lefthand side is a.*

The distance between two nodes reflects the number of steps are needed until the influence of an atom affects the evaluation of another one.

Definition 8 (Distance) *Let Π be an ASM and S be a slicing criterion. For an atom α, the distance from S with respect to Π is defined as follows*

- $\mathrm{distance}_\Pi(\alpha, S) = 0$ *if α is unifiable with an element from S*
- $\mathrm{distance}_\Pi(\alpha, S) = \min\{k :$ *there exist $a, b \in V_\Pi$ such that α is unifiable with a, b is unifiable with an element from S, k is the distance of b from a in G_Π*$\}$

In the following definition, we introduce a sequence of sets that is strongly connected to the notion of distance.

Definition 9 *The set $I(\Pi, S)$ is defined by $I(\Pi, S) := \bigcup_{i \in \mathbb{N}} I^i(\Pi, S)$ where*

$$I^0(\Pi, S) := S \cup \{\alpha : \alpha \textit{ is unifiable with an element of } S\}$$

$$I^{n+1}(\Pi, S) := I^n(\Pi, S) \cup \{\alpha \in V_\Pi : \alpha \textit{ is unifiable with an atom in the absolute guard of an update rule whose lefthand side is unifiable with an element of } I^n(\Pi, S)\}$$

As Π is finite and $I^n(\Pi, S) \subseteq I^{n+1}(\Pi, S)$ for all $n \in \mathbb{N}$, there exists $n_\Pi \in \mathbb{N}$ such that

$$I^{n_\Pi}(\Pi, S) = I^{n_\Pi + 1}(\Pi, S)$$

and

$$\begin{aligned} I(\Pi, S) &= \bigcup_{i=0}^{n_\Pi} I^i(\Pi, S) \\ &= I^{n_\Pi}(\Pi, S) \end{aligned}$$

An upper bound for n_Π is $|\Pi|$, the size of Π.

Lemma 10 *Let Π be an ASM, α be an atom appearing in Π and S be a slicing criterion. Furthermore, let α be reachable from an atom that is unifiable with an element from S. Then the following holds:*

$$\text{distance}_\Pi(\alpha, S) = \min\{i \in \mathbb{N} : \alpha \in I^i(\Pi, S)\}$$

Proof. Let Π be an ASM, α be an atom appearing in Π and S be a slicing criterion. Furthermore, let α be reachable from an atom that is unifiable with an element from S. We prove the proposition by on the minimal natural number n with $\alpha \in I^n(\Pi, S)$

If $n = 0$, α is unifiable with an element S. Therefore, $\text{distance}_\Pi(\alpha, S) = 0$.

If $\min\{i \in \mathbb{N} : \alpha \in I^i(\Pi, S)\} = n+1$ then, by definition, $\alpha \in I^n(\Pi, S) \cup \tilde{I}^n(\Pi, S)$ where $\tilde{I}^n(\Pi, S) := \{\alpha : \alpha$ appears in the absolute guard of an update rule whose lefthand side is unifiable with an element of $I^n(\Pi, S)\}$. As $n+1$ is the minimal number such that $\alpha \in I^{n+1}(\Pi, S)$, we can conclude that $\alpha \notin I^n(\Pi, S)$ and therefore, $\alpha \in \tilde{I}^n(\Pi, S)$. As $n + 1 \leq n_\Pi$ (number $> n_\Pi$ cannot satisfy the minimality condition) there is at least one atom $\beta \in I^n(\Pi, S)$ such that $\min\{i \in \mathbb{N} : \beta \in I^i(\Pi, S)\} = n$. By the induction hypothesis, $\text{distance}_\Pi(\beta, \Pi) = n$. By the definition of the distance, $\text{distance}_\Pi(\beta, \Pi) = n + 1$. ☐

Lemma 11 *$I(\Pi, S)$ contains all atoms possibly influencing the valuation of an element of $L(\mathcal{A}, S)$. I.e., for all states $\mathcal{A}, \mathcal{A}'$ with* $\text{content}(\alpha, \mathcal{A}) = \text{content}(\alpha, \mathcal{A}')$

for all $\alpha \in L(I(\Pi,S),\mathcal{A}) = L(I(\Pi,S),\mathcal{A}')$ *and for all* $n \in \mathbb{N}, \mathcal{B}, \mathcal{B}'$ *with* $\mathcal{A} \vdash^n_\Pi \mathcal{B}$*,* $\mathcal{A} \vdash^n_\Pi \mathcal{B}$ *the following holds:*

$$\text{content}(\beta,\mathcal{B}) = \text{content}(\beta,\mathcal{B}') \textit{ for all } \beta \in L(I(\Pi,S),\mathcal{A})$$

Proof. We prove this proposition by induction on the distance from S with respect to Π.

If $\text{distance}_\Pi(\alpha,S) = 0$ for an atom α appearing in Π then α is unifiable with an element from S. By definition, $\alpha \in I^0(\Pi,S)$.

If $\text{distance}_\Pi(\alpha,S) = n+1$ for an atom α appearing in Π then α is appears in the absolute guard of an update rule whose lefthand side is unifiable with an element of distance n. By the induction hypothesis and the definition of $I^{n+1}(\Pi,S)$, $\alpha \in I^{n+1}(\Pi,S)$. □

5 A New Slicing Algorithm

In this section we present a new algorithm for slicing ASMs. The main properties of this algorithm are reflected by the propositions presented in the next section. Though we restrict the input of the algorithm to deterministic ASMs not using **import** it is straightforward to generalize it so it works for ASMs that nondeterministic and/or use **import**. We briefly describe this extension at the end of this work.

Input $\langle \Pi, S\rangle$ where

1. deterministic ASM Π not using **import**
2. slicing criterion S

Algorithm

1. res $:= \Pi$

   [*res will be the result of the algorithm.*]

2. Replace in res all update rules whose lefthand side and righthand side are equal by **Skip**.

   [*If both sides of an update rule are equal then it is equivalent to* **Skip**.]

3. Simplify res according to the equivalence-preserving rewriting rules given in figure 3.

4. $H := S$

   [*Introduce a new set* H *that is initially equal to to* S*. In the* i*-the pass of the while-loop,* H *contains the atoms in* $I(\Pi,S) - I^{i-1}(\Pi,S)$]

5. $R := H$

   [*Introduce a new set* R *that is initially equal to* S *and that is intended to contain all relevant atoms (with respect to* S*) when the algorithm terminates.*]

original rule	equivalent rule
do-in-parallel R **Skip enddo**	R
do-in-parallel Skip R **enddo**	R
if g **then Skip endif**	**Skip**
forall $x : g$ **do Skip endforall**	**Skip**

Fig. 3. Rewriting rules used in the linear-time algorithm

6. While $H \neq \emptyset$ do

   (a) For every $e \in H$

   $H(e) :=$ set of all update rules whose lefthand side is unifiable with e

   [*The update rules in* $H(e)$ *possibly change the interpretation of* e.]

   (b) $U := \bigcup_{e \in H} H(e)$

   [U *contains all update rules possibly influencing an element of* H.]

   (c) For every $r \in U$

   $H'(r) :=$ set of all atoms appearing in the absolute guard of r in Π

   [$H'(r)$ *is the set of all atoms possibly directly influencing the update of* r*;* $H'(r)$ *contains all successors of the lefthand side of* r.]

   (d) $H' := \bigcup_{r \in U} H'(u)$

   [H' *contains all atoms that possibly directly influence the update of an element from* U.]

   (e) $H := H' - (\{h \in H : h \text{ is unifiable with an element from } H\} \cup R)$

   (f) $R := R \cup H$

   od

7. Replace in res all update rules whose lefthand side is not unifiable with an element from R by **Skip**.

   [*The update rules do not influence the evaluation of the locations induced by* S.]

8. Simplify res according to the equivalence-preserving rewriting rules given in figure 3.

9. Output res.

For illustrating the work of the algorithm, we give a very simple example. As an input, we choose the slicing criterion $S = \{Qxy\}$ and the ASM Π defined by

```
do-in-parallel
   forall xy : Rxy do
      if P₁x ∧ P₂y then Qxy := true endif
      P₂x := true
   endforall
   forall xy : Qxy do
      Sxy := true
   endforall
enddo
```

In step 1., res is set to Π. Steps 2. and 3. do not change res as no update rule in Π satisfies the condition. In steps 4. and 5., the sets H and R are set to $\{Qxy\}$. As this set is not empty, we enter the while-loop for the first time. In step (a), we set $H(Qxy)$ to $\{Qxy := \text{true}\}$ and as Qxy is the only element of H at this point, this is the value of U after executing (b). The set of atoms in the guard of $Qxy := \text{true}$ is $\{Rxy, P_1x, P_2y\}$ and this is the set assigned to H' in step (d). In step (e), this becomes the new value of H and afterwards R is set to $\{Qxy, Rxy, P_1x, P_2y\}$. These are the atoms whose evaluation might influence (and do influence in this case) the evaluation of Qxy. As H is not empty, we enter the loop for the second time. The only atom that is unifiable with the lefthand side of an update rule in Π is P_2y, therefore U is set to $\{P_2x := \text{true}\}$ and afterwards, H' to $\{Rxy\}$. As this is already contained in H, the new value of H is $\emptyset$ and R is unchanged. The condition for the while-loop is no more satisfied and therefore, we do not enter the loop for a third time.

Finally, we replace $Sxy := \text{true}$ by **Skip** as Sxy is not unifiable with an element of R and therefore, it does not influence the evaluation of x. The application of the rewriting rules leads to the output

```
forall xy : Rxy do
   if P₁x ∧ P₂y then Qxy := true endif
   P₂x := true
endforall
```

6 Properties of the Slicing Algorithm

In this and the next section, we analyze the basic properties of the algorithm.

Proposition 1. *Let Π be a deterministic ASM not using* **import** *and S be a slicing criterion. The above algorithm outputs an S-slice of Π if Π and S are the input.*

Proof. Having left the while loop, $I(\text{res}, S) = R$ holds. Consequently, removing the updates of atoms not unifiable with an element of R does not have any influence on the evaluation of the content of locations that are an instance of an element of S. Therefore, res is an S-slice of Π before the execution of assignment 8. As the rewriting rules displayed in figure 3 preserve equivalence and correspond to the notion of subrules resp. -programs, the output is an S-slice of Π. □

Let us now consider the size of the resulting slice. It is clear that the computed slice is not always minimal. A simple example is the ASM given by **do-in-parallel** R R **enddo** where R is an arbitrary ASM-program. If $\tilde{R}$ is the result of the algorithm on $\langle R, S\rangle$ then the algorithm would output **do-in-parallel** $\tilde{R}$ $\tilde{R}$ **enddo**.

But it is possible to give a criterion resp. a restriction on the ASM(s) such that we can guarantee minimality. Though the above example is a very simple one and the reason for the non-minimality is clear, it already provides the basic idea for the restriction. In order to formulate it, we give some definitions in advance.

Definition 12 *For a first-order formula the set* $\mathrm{lsub}(\varphi)$ *of linear subformulae is defined as follows.*

- *if* φ *is an atomic formula then* $\mathrm{lsub}(\varphi) = \{\varphi\}$
- $\mathrm{lsub}(\neg\psi) = \mathrm{lsub}(\psi) \cup \{\neg\psi\}$
- $\mathrm{lsub}(\varphi_1 \wedge \varphi_2) = \mathrm{lsub}(\varphi_1) \cup \mathrm{lsub}(\varphi_2) \cup \{\varphi_1 \wedge \varphi_2)$
- $\mathrm{lsub}(\varphi_1 \vee \varphi_2) = \mathrm{lsub}(\varphi_1) \cup \mathrm{lsub}(\varphi_2) \cup \{\varphi_1 \vee \varphi_2)$
- $\mathrm{lsub}(\forall \bar{x}\psi(\bar{x})) = \mathrm{lsub}(\psi(\bar{x})) \cup \forall \bar{x}\psi(\bar{x})\}$
- $\mathrm{lsub}(\exists \bar{x}\psi(\bar{x})) = \mathrm{lsub}(\psi(\bar{x})) \cup \exists \bar{x}\psi(\bar{x})\}$

Actually, the notion of linear subformulae corresponds to the usual notion of subformula. We call it linear in order to emphasize the difference to the notion of subformula introduced before and because the size of $\mathrm{lsub}(\varphi)$ is linear in the size of φ.

Definition 13 *For an ASM* Π *the set* $\mathrm{lsub}(\Pi)$ *of linear subrules is defined as follows.*

- *if* Π *is an update rule or* **Skip** *then* $\mathrm{lsub}(\varphi) = \{\Pi\}$
- $\mathrm{lsub}($**if** φ **then** R **endif**$) = \mathrm{lsub}(R) \cup \{$**if** φ **then** R **endif**$\}$
- $\mathrm{lsub}($**do-in-parallel** R_1 R_2 **enddo**$) = \mathrm{lsub}(R_1) \cup \mathrm{lsub}(R_2) \cup \{$**do-in-parallel** R_1 R_2 **enddo**$\}$
- $\mathrm{lsub}($**forall** $\bar{x}$ **do** R **endforall**$) = \mathrm{lsub}(R) \cup \{$**forall** $\bar{x}$ **do** R **endforall**$\}$

Definition 14 *Let* Π *be an ASM and* R *be a linear subrule of* Π*. Furthermore, let* $\Pi_{o,R}$ *be the ASM resulting from* Π *by replacing the occurrence* o *of* R *by* **Skip**.

An occurrence o *of* R *is insignificant in* Π *iff for all states* $\mathcal{A}$ *and for all variable assignments* ξ *the following holds*

$$\mathrm{US}(\mathcal{A}, \Pi, \xi) = \mathrm{US}(\mathcal{A}, \Pi_{o,R}, \xi)$$

If an occurrence of a rule is insignificant then one or more other rules already generate the updates generated by this rule. Therefore, the insignificant rule is redundant and replacing it by **Skip** leads to an equivalent ASM. A similar notion can be defined for FO-formulae.

Definition 15 *Let* φ *be an FO-formula and* ψ *be a linear subformula of* φ*. Furthermore, let* $\varphi_{o,\psi,v}$ *be the ASM resulting from* φ *by replacing the occurrence* o *of* ψ *by* v*.*

An occurrence o of ψ is insignificant in φ iff for all states $\mathcal{A}$ and for all variable assignments ξ the following holds

$$\varphi_{o,\psi,\mathrm{true}} \equiv \varphi \qquad \textit{or} \qquad \varphi_{o,\psi,\mathrm{false}} \equiv \varphi$$

In the subsequent definition we introduce the property of being *reasonably structured* for an ASM. The idea can be explained informally as follows. All occurrences of atoms that have been identified syntactically to be influencing atoms are also semantically influencing. This means that it is not possible there exists a path from α to β in the Π-graph but the evaluation of β does not influence the evaluation of α. This might be the case if β occurs in the absolute guard of an update rule whose lefthand side is unifiable with α and a linear subformula containing β is valid or not satisfiable. Consequently, no matter whether β is true or false, the respective subformula always true or always false and the valuation of β does not have any influence. Such constructs are excluded in ASMs satisfying the following property.

In order to simplify the definition, we do not speak about occurrences of rules but about rules. The reason is again that we assume that every rule occurs only once in a program. If this does not apply, we have to substitute the notion of rule by the occurrence of a rule.

Definition 16 *A deterministic ASM Π not using* **import** *is reasonably structured if Π satisfies the following conditions.*

- *No linear subrule of Π is insignificant in Π.*
- *Π does not contain any guard g with a linear subformula that is insignificant in g.*
- *There is no update rule R in Π such that* $\mathrm{conj}(\mathrm{absGuard}(\Pi, R))$ *contains an insignificant linear subformula.*
- *There are no absolute guard $g = (g_1, \ldots, g_n)$ of an update rule in Π and no $i \in \{1, \ldots, n-1\}$ such that the formula* $\mathrm{impl}((\mathrm{conj}((g_1, \ldots, g_i), g_{i+1}))$ *contains an insignificant linear subformula.*

Note that it is not decidable whether an arbitrary ASM satisfies this property. The reason is the undecidability of the finite satisfiability problem for first-order logic. But in the case that the formulae to check are in a fragment of first-order logic for which finite satisfiability is decidable then we can decide whether an ASM is reasonably structured. For example, this property is decidable for guarded ASMs as the formulae to check are in the guarded fragment of first-order logic. For the guarded fragment, finite satisfiability and validity are decidable.

Lemma 17 *Let Π be an ASM, S be a slicing criterion and Π_A^S be the output of the algorithm on input $\langle \Pi, S \rangle$. If Π is reasonably structured then Π_A^S is reasonably structured.*

Proof. Let Π be an ASM, S be a slicing criterion and Π_A^S be the output of the algorithm on input $\langle \Pi, S \rangle$. Furthermore, assume that Π is reasonably structured. This difference can not be caused by a guard. As the guards are not splitted by the algorithm, they cannot cause a difference concerning this property. Therefore, we just have to prove that no rule R is insignificant in Π_A^S. W.l.o.g., we can assume that R is an update rule. Let l be the lefthand side of R. Following the construction of the algorithm, there exists $i \in \mathbb{N}$ such that $l \in I^i(\Pi, S)$. The rest of the proof will proceed via an induction on the minimal such number.

If $l \in I^0(\Pi, S)$ then l is unifiable with an element from S. In this case, R is obviously not insignificant in Π_A^S.

If $l \in I^{n+1}(\Pi, S) - I^n(\Pi, S)$ then R influences the evaluation of an element from $I^n(\Pi, S)$ within one step. By the induction hypothesis, no update-rule whose lefthand side is in $I^n(\Pi, S)$ is insignificant in Π_A^S. Therefore, R is not insignificant in Π_A^S. □

Proposition 18 *For every reasonably structured ASM not using equality, the algorithm outputs a minimal slice.*

Proof. Let Π be a reasonably structured ASM and S be a slicing criterion. Furthermore, let Π_A^S be the output of the algorithm on $\langle \Pi, S \rangle$. By proposition 1, Π_A^S is an S-slice of Π.

By contradiction, we assume that Π_A^S is not minimal. Consequently, there exists an S-slice $\tilde{\Pi}$ of Π such that $|\tilde{\Pi}| < |\Pi_A^S|$.

$\tilde{\Pi}$ has to be a subprogram of Π_A^S as Π_A^S already contains all rules from Π that might influence the evaluation of the locations indicated by S. Therefore, there exists a linear subformula that is insignificant in Π_A^S or a guard containing an insignificant linear subformula. Consequently, Π_A^s is not reasonably structured. By lemma 17, this is a contradiction to the assumption that Π is reasonably structured. □

7 Runtime of the Algorithm

The algorithm runs in time $O(|\Pi|^2)$. In order to prove this we determine for each step of the algorithm how much time it needs.

Steps 1., 4., 5. and 9. can be carried out in constant time.

In order to carry out the second step of the algorithm, we have to run through Π and check for each update rule whether its lefthand side and its righthand side coincide. As every update-rule is considered exactly once, this step can be performed in time $O(|\Pi|)$. Similar arguments hold for the steps 3., 7., and 8. where we have to note additionally that $|\mathrm{res}| < |\Pi|$.

It remains to analyze the while-loop in step 6. Consider the i-th pass of the loop. By H_i, we denote the value of H in the i-th pass and by $H_i(e)$ the one of $H(e)$. The assignment in step (a) can be carried out in time $O(|\Pi|)$ and therefore, step (a) can be carried out in time $O(|H_i| \cdot |\Pi|)$. The loop terminates after at most n_Π passes. Therefore, step (a) contributes $\sum_{i=1}^{n_\Pi} O(|H_i| \cdot |\Pi|)$ time to the complete runtime of the algorithm. Furthermore, $\sum_{i=1}^{n_\Pi} O(|H_i| \cdot |\Pi|) = O(|\Pi| \cdot \sum_{i=1}^{n_\Pi} |H_i|) \subseteq O(|\Pi|^2)$ as $H_i \cap H_j$ for $i \neq j$, each H_i contains only atoms appearing in Π and therefore, $\sum_{i=1}^{n_\Pi} |H_i| \leq |\Pi|$.

Using the fact that every update-rule of Π is considered at most once in U, we obtain a contribution of $O(|\Pi|^2)$ time by step (c) to the complete runtime.

Steps (b), (d), (e), (f) can each carried out in time $O(|\Pi|)$, all sets appearing in the execution of the algorithm are of size $\leq |\Pi| + |S|$. Note that this does also include e.g. unions of sets.

Putting together all of the previous arguments and assuming that $|S| \leq |\Pi|$, we obtain that the algorithm can be carried out in time $O(|\Pi|^2)$.

8 Extensions

Extension by nondeterminism. In order to extend the algorithm to nondeterministic ASMs, we first have to add to the definition of the absolute guard an item treating the respective rules. This can be done as follows.

If Π is a choice rule of the form **choose** $\bar{x}$: $\alpha(\bar{x})$ Π' **endchoose** then $\text{absGuard}(\Pi, R) := (\alpha(\bar{x}), \bar{x}) \circ \text{absGuard}(\Pi', R)$

Given this extended version of an absolute guard, the generalization of the algorithm and the notion of being reasonably structured are straightforward. Nothing has to be added as the nondeterminism is already treated in the extended notion of the absolute guard. The proofs are analogous to the deterministic case.

Extension by import. In order to extend the algorithm to ASMs using import, we again have to add to the definition of the absolute guard an item treating the respective rules. The approach for this case is the following.

If Π is an import rule of the form **import** v Π' **endimport** then $\text{absGuard}(\Pi, R) := \text{absGuard}(\Pi', R)$.

W.l.o.g., we can assume that a variable bounded by **import** does not appear free in a guard or the righthand side of an update rule. The reason is that every atom in which such a variable appears can be substituted by false (except that it appears on the left-hand side of an update-rule). Consequently, in the case that this happens, we treat the righthand side as false. Again, the algorithm, the property of being reasonably structured and the proofs can be directly transferred to ASMs using **import**.

9 Remarks and Improvements

The property of being reasonably structured is sufficient but not necessary for the minimality of the result. Consequently, there are ASMs that are not reasonably structured but the output of the algorithm is a minimal slice. This applies if for example the output of the algorithm is reasonably structured because the algorithm has cut off the insignificant rules in the original ASM. Nevertheless, we can not guarantee that the result is minimal. The difference between the size of a minimal slice and a slice computed by the algorithm depends on a number of issues. The difference is greater than zero only in case that the ASM is not reasonably structured. The number of cases where this property is not satisfied increases the difference. For example, the size of the output increases with an increase of the number of insignificant rules.

Though a slice is not minimal it might be nevertheless much smaller than the original ASM. The time saved by using the presented algorithm instead of the algorithm presented in [4] (assumed that it is applicable) possibly compensates the difference and the additional time that is needed in order to understand the larger ASM.

It remains to remark that the number of insignificant linear subrules or subformula is lower the more attention is paid during the development process of the ASM. In the best case, the ASM is reasonably structured. Of course, the knowledge that an ASM is reasonably structured already simplifies the task of determining a minimal slice but if an ASM is large then it is nevertheless not possible to determine one manually.

Furthermore, there is a number of ways to improve the algorithm, mainly with respect to the size of the result. An example for such an improvement in the general case is to figure out typical cases why minimality is not achieved and to implement additional tests. A simple example is that an ASM might contain a parallel composition of n rules where two of them are equal up to renamings of variables. Therefore, they are equivalent and at least one of them is insignificant and can be removed in order to obtain a slice. The original algorithm would keep both rules. But note that this improvement of the result would mean an increase of the runtime.

Another example has already been included in steps 2. and 3. of the algorithm.

In case that we allow equality in the guards of an ASMs, the property of being reasonably structured is not sufficient in order to guarantee that the result is minimal. But we can introduce constraints on the variables in the algorithm. For example, in an ASM, it can be enforced by an appropriate that an update of a binary relation might happen only for pairs where the components are not equal. Therefore, update rules with a lefthand side Rxx do not have an influence though Rxx is unifiable with Ryz. In the algorithm we have to check additionally such constraints and possibly exclude certain atoms (and the respective updates) from those that might influence the evaluation.

In case that we do not extend the algorithm in the previously described manner, it would nevertheless provide a slice for ASMs containing equality. But it possibly contains more rules than needed and therefore, it might be larger.

References

1. E. Börger and R. Stärk. *Abstract State Machines.* Springer-Verlag, 2003.
2. Y. Gurevich. May 1997 Draft of the ASM Guide. Technical Report CSE-TR-336-97, EECS Dept, University of Michigan, 1997.
3. A. Nowack. A Guarded Fragment for Abstract State Machines. In *Proceedings of the ESSLLI 2004 Workshop on Guarded Fragments*, 2004.
4. A. Nowack. Slicing Abstract State Machines. In *Abstract State Machines 2004. Advances in Theory and Practice*, volume 3052 of *LNCS*, pages 186–201. Springer-Verlag, 2004.