INTERACTIVE SMALL-STEP ALGORITHMS II: ABSTRACT STATE MACHINES AND THE CHARACTERIZATION THEOREM

ANDREAS BLASS, YURI GUREVICH, DEAN ROSENZWEIG, AND BENJAMIN ROSSMAN

Mathematics Dept., University of Michigan, Ann Arbor, MI 48109, U.S.A.
e-mail address: ablass@umich.edu

Microsoft Research, One Microsoft Way, Redmond, WA 98052, U.S.A.
e-mail address: gurevich@microsoft.com

University of Zagreb, FSB, I. Lučića 5, 10000 Zagreb, Croatia
e-mail address: dean@math.hr

Computer Science Dept., M.I.T., Cambridge, MA 02139, U.S.A.
e-mail address: brossman@mit.edu

Abstract. In earlier work, the Abstract State Machine Thesis — that arbitrary algorithms are behaviorally equivalent to abstract state machines — was established for several classes of algorithms, including ordinary, interactive, small-step algorithms. This was accomplished on the basis of axiomatizations of these classes of algorithms. In a companion paper [5] the axiomatisation was extended to cover interactive small-step algorithms that are not necessarily ordinary. This means that the algorithms (1) can complete a step without necessarily waiting for replies to all queries from that step and (2) can use not only the environment's replies but also the order in which the replies were received. In order to prove the thesis for algorithms of this generality, we extend here the definition of abstract state machines to incorporate explicit attention to the relative timing of replies and to the possible absence of replies. We prove the characterization theorem for extended ASMs with respect to general algorithms as axiomatised in [5].

1. Introduction

Traditional models of computation, like the venerable Turing machine, are, despite the Church-Turing thesis, rather distant intuitively from many of the concerns of modern computing. Graphical user interfaces, parallel and distributed computing, communication and security protocols, and various other sorts of computation do not easily fit the traditional picture of computing from input strings to output strings. Abstract state machines (ASMs) were introduced for the purpose of modeling algorithms at their natural level of abstraction, as opposed to the far lower level of abstraction usually needed by a Turing machine model. That ASMs fulfill their purpose was at first an empirical fact, supported by numerous case

DOI:10.2168/LMCS-???

studies, not only of algorithms in the usual sense but also of whole programming languages and of hardware; see [11] for many examples. The Abstract State Machine Thesis, first proposed in [7] and then elaborated in [8, 9], asserts that every algorithm is equivalent, on its natural level of abstraction, to an abstract state machine. Beginning in [10] and continuing in [1], [2], [3], and [4], the thesis has been proved for various classes of algorithms. In each case, the class of algorithms under consideration was defined by postulates describing, in very general terms, the nature of the algorithms, and in each case the main theorem was that all algorithms of this class are equivalent, in a strong sense, to ASMs.

The thesis was proved first, in [10], for the class of algorithms that are sequential (i.e., proceed in discrete steps and do only a bounded amount of work per step) and do not interact with the external environment within steps. (The environment is allowed to intervene between steps to change the algorithm's state.)

Subsequent work extended the result in two directions. Parallel algorithms, in which a bound on work per step applies to each processor but not to the algorithm as a whole, were treated in [1] but still without intrastep interaction with the environment. In [2, 3, 4], intrastep interaction was added to sequential computation, subject to a restriction to "ordinary" interaction, and the ASM thesis was proved for the resulting class of algorithms. In both of these directions, the standard syntax of ASMs, as presented in [9], was adequate, with only very minor modifications.

In the present paper and its companion paper [5], we continue this tradition, now removing the restriction to ordinary interaction. That is, we propose the postulates in [5] as a general description of sequential algorithms interacting with their environments, and we show in the present paper that all algorithms that satisfy the postulates are behaviorally equivalent, in a strong sense, to ASMs.

There is, however, an important difference between this work and the earlier proofs of the ASM thesis. The traditional ASM syntax and semantics from [9] are no longer adequate. They require a significant extension, allowing an ASM program (1) to refer to the order in which the values of external functions are received from the environment and (2) to declare a step complete even if not all external function values have been determined. Neither of these two possibilities was permitted by the postulates defining "ordinary algorithm" in [2].

In [5], we presented postulates that permit both of these possibilities, and we argued that these postulates capture the general concept of sequential, interactive algorithm. In the present paper, we extend the syntax and semantics of abstract state machines so that non-ordinary algorithms become expressible. The main contributions of this paper are

- syntax and semantics for ASMs incorporating interaction that need not be ordinary,
- verification that ASMs satisfy the postulates of [5], and
- proof that every algorithm satisfying the postulates is equivalent, in a strong sense, to an ASM.

Most design decisions about the syntax and semantics of general interactive ASMs were guided, and often forced, by the axiomatisation of appropriate algorithms in the companion paper [5]. Sections 2 and 3, defining the syntax and semantics of interactive ASMs are self-contained and could in principle be read independently of [5], but we refer the reader to [5], and sometimes also to [2, 3, 4], for extensive discussion, motivation and justification of some of the choices made, as well as the relation to other work. Sections 4 and 5, relating the ASMs of section 2 to algorithms as axiomatized in [5], use the definitions and results of [5]. We presume that the reader has a copy of the companion paper [5] available.

2. Interactive Small-Step ASMs: Syntax

Ordinary interactive small-step ASMs are defined in [3]. In the companion paper [5], we axiomatized general interactive small-step algorithms. In this and the next sections, we define general interactive small-step ASMs. This new ASM model is an extension of the ASM model in [3]. The extension incorporates capabilities for taking into account the order of the environment's replies and for ending a step before all queries have been answered. We repeat here, for the sake of completeness, some definitions from [3, 5], but we do not repeat the detailed discussion and motivation for these definitions. We provide detailed discussion and motivation for those aspects of the present material that go beyond what was in [3, 5].

In this section we describe the syntax of ASM programs, accompanied with some intuitive indications of their semantics. Precise semantics is given in the next section.

2.1. **Vocabularies.** An ASM has a finite vocabulary Υ complying with the following convention.

Convention 2.1.

- A vocabulary Υ consists of function symbols with specified arities.
- Some of the symbols in Υ may be marked as *static*, and some may be marked as *relational*. Symbols not marked as static are called *dynamic*.
- Among the symbols in Υ are the logic names: nullary symbols `true`, `false`, and `undef`; unary `Boole`; binary equality; and the usual propositional connectives. All of these are static and all but `undef` are relational.
- In any Υ-structure, the interpretations of `true`, `false`, and `undef` are distinct.
- In any Υ-structure, the interpretations of relational symbols are functions whose values lie in $\{\mathtt{true}_X, \mathtt{false}_X\}$.
- In any Υ-structure X, the interpretation of `Boole` maps $\mathtt{true}_X$ and $\mathtt{false}_X$ to $\mathtt{true}_X$ and everything else to $\mathtt{false}_X$.
- In any Υ-structure X, the interpretation of equality maps pairs of equal elements to $\mathtt{true}_X$ and all other pairs to $\mathtt{false}_X$.
- In any Υ-structure X, the propositional connectives are interpreted in the usual way when their arguments are in $\{\mathtt{true}_X, \mathtt{false}_X\}$, and they take the value $\mathtt{false}_X$ whenever any argument is not in $\{\mathtt{true}_X, \mathtt{false}_X\}$.
- We may use the same notation X for a structure and its base set.
- We may omit subscripts X, for example from `true` and `false`, when there is no danger of confusion. □

In addition, the ASM has an *external vocabulary* E, consisting of finitely many *external function symbols*[1]. These symbols are used syntactically exactly like the symbols from Υ, but their semantics is quite different. If f is an n-ary external function symbol and **a** is an n-tuple of arguments from a state X, then the value of f at **a** is not stored in the state but is obtained from the environment as the reply to a query.

Remark 2.2. The ASM syntax of [3] included commands of the form $\mathtt{Output}_l(t)$ where t is a term and l is a so-called output label. These commands produced an outgoing message, regarded as a query with an automatic reply "OK." In the present paper, we shall include

---

[1] The symbol E for the external vocabulary is the Greek capital epsilon, in analogy with the Greek capital upsilon Υ for the algorithm's vocabulary.

commands for issuing the queries associated with external function calls even when the reply might not be used in the evaluation of a term. These `issue` commands subsume the older `Output` commands, so we do not include the latter in our present syntax. This is why the preceding paragraph introduces only the external vocabulary and not an additional set of output labels. Note in this connection that the simulation of ordinary interactive small-step algorithms by ASMs in [4] did not use `Output` rules.   □

Convention 2.3. Note that by Convention 2.1 only function symbols in Υ admit two sorts of markings. They can be either static or dynamic and they can be relational or not. *No such markings* are applied to the external function symbols. All symbols in E are considered static and not relational.   □

Remark 2.4. In this convention, "static" does not mean that the values of external functions cannot change; it means that the algorithm cannot change them, although the environment can. External functions cannot be the subject of updates in an ASM program, and in this respect they have the same syntax as static function symbols from Υ.

We do not declare any external function symbols to be relational because such a declaration would, depending on its semantical interpretation, lead to one of two difficulties.

One possibility would be to demand that the queries resulting from relational external functions get replies that are appropriate values for such functions, namely only `true` and `false`. This imposes a burden on the environment, and a fairly complicated one, since it may not be evident, by inspection of a query, what external function symbol produced it (see the discussion of templates below). We prefer in this paper to keep the environment unconstrained.

A second possibility for handling relational external functions is to allow the environment to give arbitrary, not necessarily Boolean, replies to the queries resulting from these symbols. Then we could have non-Boolean values for Boolean terms, and we would have to decide how to handle this pathological situation, for example when it occurs in the guard of a conditional rule. In [3, Section 5], this approach was used, with the convention that this sort of pathology would cause the conditional rule to fail. In our present situation, that convention no longer looks so natural, because the pathological value might be one that the algorithm didn't really need. (Recall that in [2, 3, 4] algorithms needed replies to all of their queries.) One can probably find a reasonable convention for dealing with this pathology even for general interactive algorithms, but the convention would appear somewhat arbitrary, and it seems simpler to prohibit external function symbols from being relational.

It might appear that this prohibition could cause a problem in programming. Suppose, for example, that we know somehow that the environment will provide a Boolean value for a certain nullary external function symbol p. Then we might want to use p as the guard in a conditional statement. But we can't; since p isn't a relational symbol, it is not a Boolean term, and so (according to the definitions in the following subsections) it is ineligible to serve as a guard. Fortunately, this problem disappears when we observe that $p = $ `true` is a perfectly good guard (since equality is relational) and it has the same value as p (since we allegedly know that p gets a Boolean value). If, on the other hand, we're not sure that the environment will provide a Boolean value, then a particular decision about how to handle a non-Boolean value can be built into the program. For example, the convention from [3] would be given by

```
do in parallel
    if p = true then R1 endif
    if p = false then R2 endif
    if p ≠ true and p ≠ false then fail endif
enddo.
```

If one wanted to adopt a convention such as this, not only in a particular program but throughout some programming language, then one could adjoin external relational symbols to our syntax and treat them as syntactic sugar for pieces of code like that exhibited above.

2.2. Terms.

Definition 2.5. The set of *terms* is the smallest set containing $f(t_1, \ldots, t_n)$ whenever it contains $t_1, \ldots, t_n$ and f is an n-ary function symbol from $\Upsilon \cup \mathrm{E}$. (The basis of this recusive definition is, of course, given by the 0-ary function symbols.) □

This definition formalizes the assertion above that the external function symbols in E are treated syntactically like those of the state vocabulary Υ.

Notice that the terms of ASMs do not involve variables. In this respect they differ from those of [3], those of first-order logic, and those used in the bounded exploration witnesses of [5]. It may be surprising that we can get by without variables while describing algorithms more general than those of [3] where we used variables. Recall, however, that the variables in the ASM programs of [3] are bound by the `let` construct, and that this construct is eliminable according to [4, Section 7]. In the present paper, we use `let` only as syntactic sugar (see Subsection 2.6 below), and so we do not need variables in our basic formalism.

Definition 2.6. A *Boolean term* is a term of the form $f(\mathbf{t})$ where f is a relational symbol. □

Convention 2.7. By Υ-*terms*, we mean terms built using the function symbols in Υ and variables. These are terms in the usual sense of first-order logic for the vocabulary Υ. Terms as defined above, using function symbols from $\Upsilon \cup \mathrm{E}$ but not using variables, will be called *ASM-terms* when we wish to emphasize the distinction from Υ-terms. A term of the form $f(\mathbf{t})$ where $f \in \mathrm{E}$ will be called a *query-term* or simply *q-term*.

The evaluation of an Υ-term produces an element of the state, the value of the term. The same applies to q-terms, but there the situation is more involved. Consider a q-term $s = f(\mathbf{t})$ and suppose that $\mathbf{t}$ has been evaluated to $\mathbf{a}$. First the evaluation of $f(\mathbf{a})$ produces a query, called the q-value of s. If and when a reply to the query is received the evaluation of s is complete and we get the actual value s. See details in section 3.

2.3. Guards. In [3], the guards φ in conditional rules `if` φ `then` R_0 `else` R_1 `endif` were simply Boolean terms. We shall need guards of a new sort to enable our ASMs to take into account the temporal order of the environment's replies and to complete a step even when some queries have not yet been answered.

We introduce timing explicitly into the formalism with the notation $(s \preceq t)$, which is intended to mean that the replies needed to evaluate the term s arrived no later than those needed to evaluate t. It may seem that we are thereby just introducing a new form of Boolean term, but in fact the situation is more complicated.

In the presence of all the replies needed for both s and t, the guard $s \preceq t$ will have a truth value, determined by relative timing of replies. At the other extreme, if neither s nor t can be fully evaluated, then $(s \preceq t)$ must, like s and t themselves, have no value. So far, $(s \preceq t)$ behaves like a term.

Between the two extremes, however, there are situations where the replies provided by the environment suffice for the evaluation of one but not both of s and t. If replies suffice for s but not for t, then $(s \preceq t)$ is true; if replies suffice for t but not for s, then $(s \preceq t)$ is false. Here, $(s \preceq t)$ behaves quite differently from a term, in that it has a value even when one of its subterms does not.

This behavior of $(s \preceq t)$ also enables an ASM to complete its step while some of its queries remain unanswered. The execution of a conditional rule with $(s \preceq t)$ as its guard can proceed to the appropriate branch as soon as it has received enough replies from the environment to evaluate at least one of s and t, without waiting for the replies needed to evaluate the other.

We shall need similar behavior for more complicated guards, and for this purpose we shall use the propositional connectives of Kleene's strong three-valued logic, which perfectly fits this sort of situation [12, §64]. We use the notations $\curlywedge$ and $\curlyvee$ for the conjunction and disjunction of this logic. They differ from the classical connectives $\wedge$ and $\vee$ in that $\varphi \curlywedge \psi$ has the value false as soon as either of φ and ψ does, even if the other has no value, and $\varphi \curlyvee \psi$ has the value true as soon as either of φ and ψ does, even if the other has no value. In other words, if the truth value of one of the constituents φ and ψ suffices to determine the truth value of the compound formula, regardless of what truth value the other constituent gets, then this determination takes effect without waiting for the other constituent to get any truth value at all. (It is customary, in discussions of these modified connectives, to treat "unknown" as a third truth value, but it will be convenient for us to regard it as the absence of a truth value. Such absences occur anyway, even for ordinary terms, when the existing replies do not suffice for a complete evaluation, and it seems superfluous to introduce another entity, "unknown," to serve as a marker of this situation.)

For detailed formal definition of the semantics of guards see section 3 below.

Definition 2.8. The set of *guards* is defined by the following recursion.
- Every Boolean term is a guard.
- If s and t are terms, then $(s \preceq t)$ is a guard.
- If φ and ψ are guards, then so are $(\varphi \curlywedge \psi)$, $(\varphi \curlyvee \psi)$, and $\neg\varphi$.

$\square$

Notice that the first clause of this definition allows, in particular, terms built by means of the ordinary, 2-valued connectives from other Boolean terms.

2.4. **Rules.** Most of the definition of ASM rules is as in [3]. The differences are in the use of `issue` rules in place of the less general `Output` rules of [3] and in the more general notion of guard introduced above.

Definition 2.9. The set of ASM *rules* is defined by the following recursion.
- If $f \in \Upsilon$ is a dynamic n-ary function symbol, if $t_1, \ldots, t_n$ are terms, and if t_0 is a term that is Boolean if f is relational, then

$$f(t_1, \ldots, t_n) := t_0$$

is a rule, called an *update* rule.

- If $f \in \mathrm{E}$ is an external n-ary function symbol and if $t_1, \ldots, t_n$ are terms, then

$$\texttt{issue } f(t_1, \ldots, t_n)$$

  is a rule, called an *issue* rule.
- `fail` is a rule.
- If φ is a guard and if R_0 and R_1 are rules, then

$$\texttt{if } \varphi \texttt{ then } R_0 \texttt{ else } R_1 \texttt{ endif}$$

  is a rule, called a *conditional* rule. R_0 and R_1 are its *true* and *false branches*, respectively.
- If k is a natural number (possibly zero) and if $R_1, \ldots, R_k$ are rules then

$$\texttt{do in parallel } R_1, \ldots, R_k \texttt{ enddo}$$

  is a rule, called a *parallel combination* or *block* with the subrules R_i as its *components*.

□

We may omit the end-markers `endif` and `enddo` when they are not needed, for example in very short rules or in programs formatted so that indentation makes the grouping clear.

Example 2.10. In [5] we have analyzed the algorithm of a broker who offers a block of a shares of stock s at price p to clients i by issuing queries $q_i(s, p, a)$, $i = 0, 1$. The client whose reply reaches the broker first wins the sale. We consider here a variant of the example in which every reply from a client is considered to be positive, so that a client refuses the offer by not answering at all. If both replies reach the broker simultaneously then, for simplicity, client 0 is preferred. There is a further timeout query t, so that if no client replies by timeout, the sale is canceled. Given that $t, q_i \in \mathrm{E}$ and $s, p, a, 0, 1$ are some Υ-terms, an equivalent ASM program might be

```
if  ¬(q₀(s,p,a) ⪯ t) ⋏ ¬(q₁(s,p,a) ⪯ t)   then cancel
else if  q₀(s,p,a) ⪯ q₁(s,p,a) then sell to 0
else sell to 1
```

where `cancel` and `sell to` i stand for some updates recording respectively canceling the sale or selling to client i in the state.

2.5. **Queries and templates.** We recall the query-reply model that is discussed at length in [2, 3] and summarized in [5]. In addition to vocabulary Υ and external vocabulary E, an ASM has a set Λ of *labels*.

Definition 2.11. A *potential query* in Υ-structure X is a finite tuple of elements of $X \sqcup \Lambda$. A *potential reply* in X is an element of X. □

Here $X \sqcup \Lambda$ is the disjoint union of X and Λ. So if they are not disjoint, then they are to be replaced by disjoint isomorphic copies. We shall usually not mention these isomorphisms; that is, we write as though X and Λ were disjoint.

The correspondence between external function calls on the one hand and queries on the other hand is mediated by a template assignment, defined as follows.

Definition 2.12. For a fixed label set Λ, a *template* for n-ary function symbols is any tuple in which certain positions are filled with labels from Λ while the rest are filled with the *placeholders* $\#1, \ldots, \#n$, occurring once each. We assume that these placeholders are distinct from all the other symbols under discussion ($\Upsilon \cup \mathrm{E} \cup \Lambda$). If Q is a template for n-ary functions, then we write $Q[a_1, \ldots, a_n]$ for the result of replacing each placeholder $\#i$ in Q by the corresponding a_i. $\qquad\square$

Thus if the a_i are elements of a state X then $Q[a_1, \ldots, a_n]$ is a potential query in X.

Definition 2.13. For a fixed label set and external vocabulary, a *template assignment* is a function assigning to each n-ary external function symbol f a template $\hat{f}$ for n-ary functions, $\qquad\square$

The intention, which will be formalized in the semantic definitions of the next section, is that when an ASM evaluates a term $f(t_1, \ldots, t_n)$ where $f \in \mathrm{E}$, it first computes the values a_i of the terms t_i, then issues the query $\hat{f}[a_1, \ldots, a_n]$, and finally uses the answer to this query as the value of $f(t_1, \ldots, t_n)$.

Template assignments solve the problem whether two distinct syntactic occurrences of the same function symbol with the same arguments refer to the same query or denote distinct queries. Sometimes it is convenient to have it one way, and sometimes another. For extensive discussion of template assignments we refer the reader to [3].

2.6. **Programs.** Now we are ready to define ASM programs.

Definition 2.14. An *interactive, small-step, ASM program* Π consists of
- a finite vocabulary Υ,
- a finite set Λ of labels,
- a finite external vocabulary E,
- a rule R, using the vocabularies Υ and E, the *underlying rule* of Π,
- a template assignment with respect to E and Λ.

This completes the definition of the syntax of ASMs. It will, however, be convenient notationally and suggestive conceptually to introduce abbreviations, syntactic sugar, for certain expressions. Specifically, we adopt the following conventions and notations.

Convention 2.15. The parallel combination with no components, officially written `do in parallel enddo`, is abbreviated to `skip`. $\qquad\square$

Convention 2.16. The parallel combination with $k \geq 2$ components $R_1, \ldots, R_k$ can be written as R_1 `par` $\ldots$ `par` R_k. $\qquad\square$

Semantically, `par` is commutative and associative, that is, rules that differ only by the order and parenthesization of parallel combinations will have the same semantic behavior. Thus, in contexts where only the semantics matters, parentheses can be omitted in iterated `par`s.

Convention 2.17. We abbreviate if φ then R `else skip endif` as if φ then R `endif`. $\square$

Convention 2.18. For any term t, the Boolean term $t = t$ is denoted by $t!$, read as "t bang." $\qquad\square$

These bang terms may seem trivial, but they can be used to control timing in the execution of an ASM. If the term t involves external function symbols, then the rule `if t! then R endif` differs from R in that it issues the queries needed for the evaluation of t and waits for the replies before proceeding to execute R.

Convention 2.19. We use the following abbreviations:

$(s \prec t)$	for	$\neg(t \preceq s),$
$(s \approx t)$	for	$(s \preceq t) \curlywedge (t \preceq s),$
$(s \succeq t)$	for	$(t \preceq s), \text{and}$
$(s \succ t)$	for	$(t \prec s)$

Parentheses may be omitted when no confusion results. $\qquad\square$

The final two items of syntactic sugar involve two ways of binding variables to terms by `let` operators. Our syntax so far does not include variables, but it is easy to add them.

Definition 2.20. Fix an infinite set of variables. ASM *rules with variables* are defined exactly like ASM rules, with variables playing the role of additional, nullary, static symbols.
$\qquad\square$

Convention 2.21. If $R(v_1, \ldots, v_k)$ is a rule with distinct variables v_i, and if $t_1, \ldots, t_k$ are terms then the *let-by-name* notation

$$\texttt{n-let } v_1 = t_1, \ldots, v_k = t_k \texttt{ in } R(v_1, \ldots, v_k)$$

means $R(t_1, \ldots, t_k)$. $\qquad\square$

Convention 2.22. If $R(v_1, \ldots, v_k)$ is a rule with distinct variables v_i, and if $t_1, \ldots, t_k$ are terms then the *let-by-value* notation

$$\texttt{v-let } v_1 = t_1, \ldots, v_k = t_k \texttt{ in } R(v_1, \ldots, v_k)$$

abbreviates

$$\texttt{if } t_1! \wedge \cdots \wedge t_k! \texttt{ then } R(t_1, \ldots, t_k).$$

$\qquad\square$

For both `n-let` and `v-let` rules, the v_i are called the *variables* of the rule, the t_i its *bindings*, and $R(v_1, \ldots, v_k)$ its *body*. Each of the variables v_i is bound by this rule at its initial occurrence in the context $v_i = t_i$ and at any free occurrences in $R(v_1, \ldots, v_k)$. (Occurrences of the variables v_i in the terms t_j are not bound by the `n-let` or `v-let` construction, regardless of whether $i = j$ or not.)

The let-by-name notation simply uses variables v_i as placeholders for the terms t_i. The let-by-value notation, in contrast, first evaluates all the t_i and only afterward proceeds to execute the rule R. In this sense, the two forms of `let` correspond to call-by-name and call-by-value in other situations.

Example 2.23. Let t be a term representing a query asking the environment for a fresh object, like constructors in object-oriented languages, so that distinct textual occurrences of t in a program represent distinct queries with supposedly distinct replies. Let $R(t)$ be a rule with several syntactic occurrences of t. Then `n-let` $x = t$ `in` $R(x)$ provides just an

abbreviation for $R(t)$ (if it is indeed shorter than $R(t)$), while `v-let` $x = t$ `in` $R(x)$ has a completely different meaning: first ask the environment for a fresh object, await the reply, and then use it repeatedly.

3. Interactive Small-Step ASMs: Semantics

Throughout this section, we refer to a fixed structure X. We start by recalling the notion of history introduced and motivated in the companion paper [5]. Then we define the semantics of terms, guards, and rules in the structure X, relative to histories ξ. In each case, we tacitly presume a template assignment. (Unlike X, the history ξ will not remain fixed, because the meaning of a guard under history ξ can depend on the meanings of its subterms under initial segments of ξ.) In each case, the semantics will specify a causality relation. In addition, for terms and guards the semantics may provide a value (Boolean in the case of guards); for rules, the semantics may declare the history final, successful, or failing, and may provide updates.

3.1. **Histories.** The notion of *history* as a formal model of intrastep interaction of an algorithm and its environment has been introduced and extensively discussed in [5]. We recall the relevant definitions.

Definition 3.1. An *answer function* for a state X is a partial map from potential queries to potential replies. A *history* for X is a pair $\xi = \langle \dot{\xi}, \leq_\xi \rangle$ consisting of an answer function $\dot{\xi}$ together with a linear pre-order $\leq_\xi$ of its domain. By the *domain* of a history ξ, we mean the domain $\mathrm{Dom}(\dot{\xi})$ of its answer function component, which is also the field of its pre-order component. $\square$

Recall that a *pre-order* of a set D is a reflexive, transitive, binary relation on D, and that it is said to be *linear* if, for all $x, y \in D$, $x \leq y$ or $y \leq x$. The equivalence relation defined by a pre-order is given by

$$x \equiv y \iff x \leq y \leq x.$$

The equivalence classes are partially ordered by

$$[x] \leq [y] \iff x \leq y,$$

and this partial order is linear if and only if the pre-order was.

The *length* of a linear pre-order is defined to be the order type of the induced linear ordering of equivalence classes. (We shall use this notion of length only in the case where the number of equivalence classes is finite, in which case this number serves as the length.)

We also write $x < y$ to mean $x \leq y$ and $y \not\leq x$. (Because a pre-order need not be antisymmetric, $x < y$ is in general a stronger statement than the conjunction of $x \leq y$ and $x \neq y$.) When, as in the definition above, a pre-order is written as $\leq_\xi$, we write the corresponding equivalence relation and strict order as $\equiv_\xi$ and $<_\xi$. The same applies to other subscripts and superscripts.

Definition 3.2. Let $\leq$ be a pre-order of a set D. An *initial segment* of D with respect to $\leq$ is a subset S of D such that whenever $x \leq y$ and $y \in S$ then $x \in S$. An *initial segment* of $\leq$ is the restriction of $\leq$ to an initial segment of D with respect to $\leq$. An *initial segment* of a history $\langle \dot{\xi}, \leq_\xi \rangle$ is a history $\langle \dot{\xi} \restriction S, \leq_\xi \restriction S \rangle$, where S is an initial segment of $\mathrm{Dom}(\dot{\xi})$ with

respect to $\leq_\xi$. (We use the standard notation $\upharpoonright$ for the restriction of a function or a relation to a set.) We write $\eta \trianglelefteq \xi$ to mean that the history η is an initial segment of the history ξ. $\square$

3.2. **Terms.** The semantics of terms presumes not only an Υ-structure X and a template assignment but also a history ξ. The semantics is essentially the same as in [3], except that we do not use variables here. In particular, the history ξ is involved only via the answer function $\dot\xi$; the pre-order is irrelevant.

The semantics of terms specifes, by induction on terms t, the queries that are caused by ξ under the associated causality relation $\vdash^t_X$ and sometimes also a value $\mathrm{Val}(t, X, \xi)$. In the case of query-terms, the semantics may specify also a query-value q-$\mathrm{Val}(t, X, \xi)$. An evaluation of a query-term t is intended to produce first a query, called the q-value of t and denoted q-$\mathrm{Val}(t, X, \xi)$; the reply, if any, to the query is the actual value $\mathrm{Val}(t, X, \xi)$ of t.

Definition 3.3. Let t be the term $f(t_1, \ldots, t_n)$.
- If $\mathrm{Val}(t_i, X, \xi)$ is undefined for at least one i, then $\mathrm{Val}(t, X, \xi)$ is also undefined, and $\xi \vdash^t_X q$ if and only if $\xi \vdash^{t_i}_X q$ for at least one i. If $f \in \mathrm{E}$ then q-$\mathrm{Val}(t, X, \xi)$ is also undefined.
- If, for each i, $\mathrm{Val}(t_i, X, \xi) = a_i$ and if $f \in \Upsilon$, then $\mathrm{Val}(t, X, \xi) = f_X(a_1, \ldots, a_n)$, and no query q is caused by ξ.
- If, for each i, $\mathrm{Val}(t_i, X, \xi) = a_i$, and if $f \in \mathrm{E}$, then q-$\mathrm{Val}(t, X, \xi)$ is the query $\hat{f}[a_1, \ldots, a_n]$.
    - If q-$\mathrm{Val}(t, X, \xi) = q \in \mathrm{Dom}(\dot\xi)$, then $\mathrm{Val}(t, X, \xi) = \dot\xi(q)$, and no query is caused by ξ.
    - If q-$\mathrm{Val}(t, X, \xi) = q \notin \mathrm{Dom}(\dot\xi)$, then $\mathrm{Val}(t, X, \xi)$ is undefined, and q is the unique query such that $\xi \vdash^t_X q$.

$\square$

We record for future reference three immediate consequences of this definition; the proofs are routine inductions on terms.

Lemma 3.4. $\mathrm{Val}(t, X, \xi)$ is defined if and only if there is no query q such that $\xi \vdash^t_X q$.

Lemma 3.5. If $\xi \vdash^t_X q$ then $q \notin \mathrm{Dom}(\dot\xi)$.

Lemma 3.6. If $\eta \trianglelefteq \xi$ (or even if merely $\dot\eta \subseteq \dot\xi$) and if $\mathrm{Val}(t, X, \eta)$ is defined, then $\mathrm{Val}(t, X, \xi)$ is also defined and these values are equal. Similarly, if t is a q-term such that q-$\mathrm{Val}(t, X, \eta)$ exists, then q-$\mathrm{Val}(t, X, \eta) = $ q-$\mathrm{Val}(t, X, \xi)$.

3.3. **Guards.** The semantics of guards, unlike that of terms, depends not only on the answer function but also on the preorder in the history. Another difference from the term case is that the values of guards, when defined, are always Boolean values. Guards share with terms the property that they produce queries if and only if their values are undefined.

Definition 3.7. Let φ be a guard and ξ a history in an Υ-structure X.
- If φ is a Boolean term, then its value (if any) and causality relation are already given by Definition 3.3.

- If φ is $(s \preceq t)$ and if both s and t have values with respect to ξ, then $\mathrm{Val}(\varphi, X, \xi) =$ **true** if, for every initial segment $\eta \trianglelefteq \xi$ such that $\mathrm{Val}(t, X, \eta)$ is defined, $\mathrm{Val}(s, X, \eta)$ is also defined. Otherwise, $\mathrm{Val}(\varphi, X, \xi) = $ **false**. Also declare that $\xi \vdash_X^\varphi q$ for no q.
- If φ is $(s \preceq t)$ and if s has a value with respect to ξ but t does not, then define $\mathrm{Val}(\varphi, X, \xi)$ to be **true**; again declare that $\xi \vdash_X^\varphi q$ for no q.
- If φ is $(s \preceq t)$ and if t has a value with respect to ξ but s does not, then define $\mathrm{Val}(\varphi, X, \xi)$ to be **false**; again declare that $\xi \vdash_X^\varphi q$ for no q.
- If φ is $(s \preceq t)$ and if neither s nor t has a value with respect to ξ, then $\mathrm{Val}(\varphi, X, \xi)$ is undefined, and $\xi \vdash_X^\varphi q$ if and only if $\xi \vdash_X^s q$ or $\xi \vdash_X^t q$.
- If φ is $\psi_0 \curlywedge \psi_1$ and both ψ_i have value **true**, then $\mathrm{Val}(\varphi, X, \xi) = $ **true** and no query is produced.
- If φ is $\psi_0 \curlywedge \psi_1$ and at least one ψ_i has value **false**, then $\mathrm{Val}(\varphi, X, \xi) = $ **false** and no query is produced.
- If φ is $\psi_0 \curlywedge \psi_1$ and one ψ_i has value **true** while the other, ψ_{1-i}, has no value, then $\mathrm{Val}(\varphi, X, \xi)$ is undefined, and $\xi \vdash_X^\varphi q$ if and only if $\xi \vdash_X^{\psi_{1-i}} q$.
- If φ is $\psi_0 \curlywedge \psi_1$ and neither ψ_i has a value, then $\mathrm{Val}(\varphi, X, \xi)$ is undefined, and $\xi \vdash_X^\varphi q$ if and only if $\xi \vdash_X^{\psi_i} q$ for some i.
- The preceding four clauses apply with $\curlyvee$ in place of $\curlywedge$ and **true** and **false** interchanged.
- If φ is $\neg\psi$ and ψ has a value, then $\mathrm{Val}(\varphi, X, \xi) = \neg\mathrm{Val}(\psi, X, \xi)$ and no query is produced.
- If φ is $\neg\psi$ and ψ has no value then $\mathrm{Val}(\varphi, X, \xi)$ is undefined and $\xi \vdash_X^\varphi q$ if and only if $\xi \vdash_X^\psi q$.

$\square$

Remark 3.8. An alternative, and perhaps more intuitive, formulation of the definition of $\mathrm{Val}((s \preceq t), X, \xi)$ in the case where both s and t have values is to let ξ' (resp. ξ'') be the shortest initial segment of ξ with respect to which s (resp. t) has a value, and to define $\mathrm{Val}(\varphi, X, \xi)$ to be **true** if $\xi' \trianglelefteq \xi''$ and **false** otherwise. This is equivalent, in the light of Lemma 3.6, to the definition given above, but it requires knowing that the shortest initial segments mentioned here, ξ' and ξ'', exist. That is clearly the case if the partial order associated to the preorder in ξ is a well-ordering, in particular if it is finite. Once we establish that ASMs satisfy the Bounded Work Postulate, it will follow that we can confine our attention to finite histories and so use the alternative explanation of $\mathrm{Val}((s \preceq t), X, \xi)$. The formulation adopted in the definition has the advantage of not presupposing that only finite histories matter. $\square$

Example 3.9. The truth value of a timing guard $(s \preceq t)$ is defined in terms of the syntactic objects s and t, not in terms of their values. As a result, this truth value may not be preserved if s and t are replaced by other terms with the same values (in the given history ξ), not even if the replacement terms ultimately issue the same queries as the original ones. Here is an example of what can happen. Suppose p, q, and r are external function symbols, p being unary and the other two nullary. Suppose further that 0 is a static nullary Υ-symbol. Consider a history ξ with three queries in its domain, pre-ordered as $\hat{p}[0_X] <_\xi \hat{q} <_\xi \hat{r}$, and with $\dot{\xi}(\hat{r}) = 0_X$. Then the term $p(0)$ has a value already for the initial segment of ξ of length 1; q gets a value later, namely for the initial segment of length 2; and $p(r)$ gets a value only for the whole history ξ, of length 3. Thus, the guards $(p(0) \prec q)$ and $(q \prec p(r))$

are true, even though $p(0)$ and $p(r)$ have the same value and have, as the ultimate step in their evaluation, the answer to the query $\hat{p}[0_X]$. $\qquad\square$

Just as for terms, the following lemmas follow immediately, by induction on guards, from the definition plus the corresponding lemmas for terms.

Lemma 3.10. $\mathrm{Val}(\varphi, X, \xi)$ is defined if and only if there is no query q such that $\xi \vdash_X^\varphi q$.

Lemma 3.11. If $\xi \vdash_X^\varphi q$ then $q \notin \mathrm{Dom}(\dot{\xi})$.

Lemma 3.12. If $\eta \trianglelefteq \xi$ and if $\mathrm{Val}(\varphi, X, \eta)$ is defined, then $\mathrm{Val}(\varphi, X, \xi)$ is also defined and these values are equal.

Remark 3.13. Given the semantics of guards, we can amplify the statement, in the Remark 3.10. of [5], that guards express descriptions like "p has reply a and p' has no reply." In view of Lemma 3.12, it is more accurate to say that a guard expresses that such a description either is correct now or was so at some earlier time. The lemma says that, once a guard is true, it remains true when the history is extended by adding new elements later in the preorder, whereas a property like "p' has no reply" need not remain true. Thus, what a guard can really express is something of the form "it either is now true or was once true that p has reply a and p' has no reply yet." This particular example would be expressed by the guard $(p = a) \curlywedge (p \prec p')$, where, for simplicity we have not introduced a separate notation for 0-ary symbols corresponding to the queries p and p' and the element a.

3.4. **Rules.** The semantics of a rule, for an Υ-structure X, an appropriate template assignment, and a history ξ, consists of a *causality relation*, declarations of whether ξ is *final* and whether it *succeeds* or *fails*, and a set of *updates*.

Definition 3.14. Let R be a rule and ξ a history for the Υ-structure X. In the following clauses, whenever we say that a history succeeds or that it fails, we implicitly also declare it to be final; contrapositively, when we say that a history is not final, we implicitly also assert that it neither succeeds nor fails.

- If R is an update rule $f(t_1, \ldots, t_n) := t_0$ and if all the t_i have values $\mathrm{Val}(t_i, X, \xi) = a_i$, then ξ succeeds for R, and it produces the update set $\{\langle f, \langle a_1, \ldots, a_n \rangle, a_0 \rangle\}$ and no queries.
- If R is an update rule $f(t_1, \ldots, t_n) := t_0$ and if some t_i has no value, then ξ is not final for R, it produces the empty update set, and $\xi \vdash_X^R q$ if and only if $\xi \vdash_X^{t_i} q$ for some i.
- If R is $\mathtt{issue}\ f(t_1, \ldots, t_n)$ and if all the t_i have values $\mathrm{Val}(t_i, X, \xi) = a_i$, then ξ succeeds for R, it produces the empty update set, and $\xi \vdash_X^R q$ for the single query $q = \hat{f}[a_1, \ldots, a_n]$ provided $q \notin \mathrm{Dom}(\dot{\xi})$; if $q \in \mathrm{Dom}(\dot{\xi})$ then no query is produced.
- If R is $\mathtt{issue}\ f(t_1, \ldots, t_n)$ and if some t_i has no value, then ξ is not final for R, it produces the empty update set, and $\xi \vdash_X^R q$ if and only if $\xi \vdash_X^{t_i} q$ for some i.
- If R is $\mathtt{fail}$, then ξ fails for R; it produces the empty update set and no queries.
- If R is a conditional rule $\mathtt{if}\ \varphi\ \mathtt{then}\ R_0\ \mathtt{else}\ R_1\ \mathtt{endif}$ and if φ has no value, then ξ is not final for R, and it produces the empty update set. $\xi \vdash_X^R q$ if and only if $\xi \vdash_X^\varphi q$.
- If R is a conditional rule $\mathtt{if}\ \varphi\ \mathtt{then}\ R_0\ \mathtt{else}\ R_1\ \mathtt{endif}$ and if φ has value $\mathtt{true}$ (resp. $\mathtt{false}$), then finality, success, failure, updates, and queries are the same for R as for R_0 (resp. R_1).

- If R is a parallel combination `do in parallel` $R_1, \ldots, R_k$ `enddo` then:
  - $\xi \vdash_X^R q$ if and only if $\xi \vdash_X^{R_i} q$ for some i.
  - The update set for R is the union of the update sets for all the components R_i. If this set contains two distinct updates at the same location, then we say that a *clash* occurs (for R, X, and ξ).
  - ξ is final for R if and only if it is final for all the R_i.
  - ξ succeeds for R if and only if it succeeds for all the R_i and no clash occurs.
  - ξ fails for R if and only if it is final for R and either it fails for some R_i or a clash occurs.

$\square$

There is no analog for rules of Lemmas 3.4 and 3.10. A rule may issue queries even though it is final (in the case of an `issue` rule) or produces updates (in the case of parallel combinations) or both. There are, however, analogs for the other two lemmas that we established for terms and guards; again the proofs are routine inductions.

Lemma 3.15. If $\xi \vdash_X^R q$ then $q \notin \mathrm{Dom}(\dot{\xi})$.

Lemma 3.16. Let $\eta \trianglelefteq \xi$.
- If η is final for R, then so is ξ.
- If η succeeds for R, then so does ξ.
- If η fails for R, then so does ξ.
- The update set for R under ξ includes that under η.

The reader might find it useful at this point to work out the semantic details of the examples 2.10 and 2.23, comparing the results with the intuitive explanations given in the examples.

3.5. **ASM definition.** If ξ is a successful, final history for a rule R over an Υ-structure X, then R and ξ produce a successor for X. We need a preliminary lemma to ensure that this successor will be well-defined. Recall that, in the definition of the semantics of parallel rules, we defined "a clash occurs" (for a rule, template assignment, state, and history) to mean that the update set contains two different updates of the same location.

Lemma 3.17. If an ASM rule with a template assignment is (final and) successful in a certain state with a certain history, then no clash occurs for this rule, template assignment, state, and history.

Proof. Use induction on rules. In the case of a parallel composition, the semantics explicitly provided for failure if a clash occurs. All other cases are trivial thanks to the induction hypothesis. $\square$

Definition 3.18. Fix a rule R endowed with a template assignment, and let X be an Υ-structure and ξ be a history for X. If ξ is successful and final for R over X, and if $\Delta^+(X, \xi)$ is the update set produced by R, X, and ξ, then the *successor* $\tau(X, \xi)$ of X with respect to R and ξ is the Υ-structure Y such that
- Y has the same base set as X,
- $f_Y(\mathbf{a}) = b$ if $\langle f, \mathbf{a}, b \rangle \in \Delta^+(X, \xi)$, and
- otherwise Y interprets function symbols exactly as X does. $\square$

Lemma 3.17 ensures that the second clause of the definition does not attempt to give $f_Y(\mathbf{a})$ two different values.

Now we are ready to give a complete definition of ASMs.

Definition 3.19. An *interactive, small-step, ASM* consists of

- an ASM program Π in some vocabulary Υ,
- a nonempty set $\mathcal{S}$ of Υ-structures called *states* of the ASM, and
- a nonempty set $\mathcal{I} \subseteq \mathcal{S}$ of *initial states*,

subject to the requirements that $\mathcal{S}$ and $\mathcal{I}$ are closed under isomorphism and that $\mathcal{S}$ is closed under transitions in the following sense. If $X \in \mathcal{S}$, if ξ is a successful, final history for Π in X, and if $\Delta^+(X, \xi)$ is the update set produced by Π, X, and ξ, then the successor $\tau(X, \xi)$ of X with respect to Π and ξ is also in $\mathcal{S}$. The successor is the *next state* for X with respect to Π, endowed with the given template assignment, and to ξ. $\qquad\square$

4. ASMs are Algorithms

This section is devoted to checking that ASMs, as just defined, are algorithms, as defined in [5]. In this section (and in the rest of the paper) we freely use the notions and results of [5].

4.1. **Obvious postulates.** Much of this checking is trivial: Everything required by the States Postulate of [5] is in our definition of ASMs. The causality relation required by the Interaction Postulate of [5] is included in our semantics for ASMs. (Strictly speaking, the causality relation defined for ASMs should be restricted to finite histories, to comply with the statement of the Interaction Postulate.) The Isomorphism Postulate of [5] is also obvious, because everything involved in our ASM semantics is invariant under isomorphisms. So the only postulates requiring any real checking are the Step and Bounded Work Postulates.

4.2. **Step Postulate.** The ASM semantics provides notions of finality, success, failure, and updates. In addition to these, the Step Postulate of [5] requires (in Part C) a notion of next state and (in Part A) assurance that every complete, coherent history has a final initial segment[2]. The next state is given by Definition 3.18, and it is well-defined because of Lemma 3.17.

To show that every complete, coherent history has a final initial segment, we actually show more, namely that every complete history is final. The main ingredient here is the following lemma.

Lemma 4.1. If a history ξ is not final for a rule R in a state X, then $\xi \vdash^R_X q$ for some query q.

Proof. We proceed by induction on the rule R, according to the clauses in the definition of the semantics for rules. Since we are given that ξ is not final, we can ignore those clauses that say ξ is final, and there remain the following cases.

---

[2]We implicitly use the notions of coherent, attainable ...history as relative to the causality relation of the ASM program as defined in Section 3

If R is either an update rule $f(t_1, \ldots, t_n) := t_0$ or an issue rule `issue` $f(t_1, \ldots, t_n)$ and some t_i has no value, then by Lemma 3.4 there is a query q such that $\xi \vdash^{t_i}_X q$, and therefore $\xi \vdash^R_X q$.

If R is a conditional rule whose guard has no value, then the same argument applies except that we invoke Lemma 3.10 in place of Lemma 3.4.

If R is a conditional rule whose guard has a truth value, then the lemma for R follows immediately from the lemma for the appropriate branch of R.

Finally, suppose R is a parallel combination. Since ξ is not final for R in X, there is a component R_i for which ξ is not final. By induction hypothesis, $\xi \vdash^{R_i}_X q$ for some q, and then we also have $\xi \vdash^R_X q$. $\square$

To complete the verification of the Step Postulate, we observe that, in the situation of the lemma, $q \in \text{Issued}^R_X(\xi)$ and, by Lemma 3.15, $q \notin \text{Dom}(\dot{\xi})$. Thus, $q \in \text{Pending}^R_X(\xi)$, and so ξ is not complete for R and X.

Remark 4.2. Because we have promised to prove that every algorithm is equivalent to an ASM, one might think that every algorithm enjoys the property established for ASMs in the preceding proof, namely that all complete histories are final. This is, however, not the case, because this property is not preserved by equivalence of algorithms. For a simple example, consider an algorithm where, for every state, the empty history is the only final history, and it causes one query, while all other histories cause no queries. Since the empty history is an initial segment of every history, Part A of the Step Postulate is satisfied, even though the complete histories, those in which the one query is answered, are not final.

Notice, however, that converting an arbitrary algorithm to an equivalent one in which all complete histories are final is much easier than converting it to an equivalent ASM. Simply adjoin all non-final, complete histories for any state to the set of final, failing histories for that state. None of the histories newly adjoined here can be attainable, so the modified algorithm is equivalent to the original. $\square$

4.3. **Bounded Work Postulate.** We turn now to the Bounded Work Postulate of [5]. Its first assertion, about the lengths of queries, is easy to check. Since the postulate refers only to coherent histories (actually to attainable, final histories, but coherence suffices for the present purpose), any query in the domain of such a history is caused by some history. By inspection of the definition of ASM semantics, all queries that are ever caused are of the form $\hat{f}[a_1, \ldots, a_n]$ and thus have the same length as the template $\hat{f}$ assigned to some external function symbol. As there are only finitely many external function symbols, the lengths of the queries are bounded.

The next assertion of the Bounded Work Postulate, bounding the number of queries issued by the algorithm, will be a consequence of the following lemma.

Lemma 4.3. For any term t, guard φ, or rule R, there is a natural number $B(t)$, $B(\varphi)$, or $B(R)$ that bounds the number of queries caused in a state X by initial segments of a history ξ. The bound depends only on t, φ, or R, not on X or ξ.

Proof. Go to the definition of the semantics of ASMs and inspect the clauses that say queries are caused. The result is that, first, we can define the desired $B(t)$ for terms by

$$B(f(t_1, \ldots, t_n)) = 1 + \sum_{i=1}^{n} B(t_i).$$

The sum here comes from the first clause in the definition of semantics of terms, and the additional 1 comes from the last clause. It is important here that, according to Lemma 3.6, all the initial segments of any ξ that produce values for a t_i produce the same value a_i. Thus, the last clause of the definition produces at most one query $\hat{f}[a_1, \ldots, a_n]$.

Similarly, we obtain for guards φ (other than the Boolean terms already treated above) the estimates

$$\begin{aligned} B((s \preceq t)) &= B(s) + B(t) \\ B(\psi_0 \curlywedge \psi_1) = B(\psi_0 \curlyvee \psi_1) &= B(\psi_0) + B(\psi_1) \\ B(\neg \psi) &= B(\psi). \end{aligned}$$

For rules, we obtain

$$\begin{aligned} B(f(t_1, \ldots, t_n) := t_0) &= \sum_{i=0}^{n} B(t_i) \\ B(\texttt{issue}\, f(t_1, \ldots, t_n)) &= 1 + \sum_{i=1}^{n} B(t_i) \\ B(\texttt{fail}) &= 0 \\ B(\texttt{if } \varphi \texttt{ then } R_0 \texttt{ else } R_1) &= B(\varphi) + B(R_0) + B(R_1) \\ B(\texttt{do in parallel } R_1, \ldots R_k) &= \sum_{i=1}^{k} B(R_i). \end{aligned}$$

(In the bound for conditional rules, we could reduce $B(R_0) + B(R_1)$ to $\max\{B(R_0), B(R_1)\}$ by using the fact that all the initial segments of any ξ that produce values for φ produce the same value.) $\square$

Since $\text{Issued}_X^R(\xi)$ is the set of queries caused in state X, under rule R, by initial segments of ξ, the lemma tells us that $|\text{Issued}_X^R(\xi)| \leq B(R)$, independently of X and ξ. This verifies the second assertion of the Bounded Work Postulate. (It actually verifies more, since the proof applies to all histories ξ, not merely to attainable ones.)

To complete the verification of the Bounded Work Postulate, it remains only to produce bounded exploration witnesses for all ASMs. We shall do this by an induction on rules, preceded by proofs of the analogous results for terms and for guards.

Lemma 4.4. For every ASM-term t (without variables) there exists a finite set $W(t)$ of Υ-terms (possibly with variables) such that, whenever (X, ξ) and (X', ξ) agree on $W(t)$, then:

- If $\xi \vdash_X^t q$ then $\xi \vdash_{X'}^t q$.
- $\text{Val}(t, X, \xi) = \text{Val}(t, X', \xi)$.

Recall that "agree on $W(t)$" means that each term in $W(t)$ has the same value in X and in X' when the variables are given the same values in $\text{Range}(\dot{\xi})$. Recall also that an equation between possibly undefined expressions like $\text{Val}(t, X, \xi)$ means that if either side is defined then so is the other and they are equal.

Proof. By a *shadow* of an ASM-term t, we mean a term $\tilde{t}$ obtained from t by putting distinct variables in place of the outermost[3] occurrences of subterms that begin with external function symbols. Thus, $\tilde{t}$ is an Υ-term, and t can be recovered from $\tilde{t}$ by a suitable substitution of ASM-terms (that start with external function symbols) for all the variables.

Notice that $\tilde{t}$ fails to be uniquely determined by t only because we have not specified which variables are to replace the subterms.

We define, by recursion on ASM-terms t, the set $W(t)$ of Υ-terms as follows. If t is $f(t_1, \ldots, t_n)$ then

$$W(t) = \{\tilde{t}\} \cup \bigcup_{i=1}^{n} W(t_i),$$

where $\tilde{t}$ is some shadow of t. To verify that this $W(t)$ satisfies the conclusion of the lemma, we proceed by induction on t, following the clauses in the definition of the semantics of terms.

Assume that (X, ξ) and (X', ξ) agree on $W(t)$. Notice that they also agree on each $W(t_i)$, because $W(t_i) \subseteq W(t)$.

Suppose first that $\mathrm{Val}(t_i, X, \xi)$ is undefined for some i. By induction hypothesis, $\mathrm{Val}(t_i, X', \xi)$ is also undefined, so the same clause of the semantics of terms applies in X and X'. That clause says that t has no value in either state and it issues those queries that are issued by any of the t_i. Those are the same queries in X as in X' by the induction hypothesis.

From now on, suppose that $\mathrm{Val}(t_i, X, \xi) = a_i$ for each i. By induction hypothesis, the same holds for X', with the same a_i's.

So if $f \in \Upsilon$ then t gets the value $f_X(a_1, \ldots, a_n)$ in X and the value $f_{X'}(a_1, \ldots, a_n)$ in X', and we must check that these values are the same. Recall that t is obtained from its shadow $\tilde{t}$ by replacing each variable v in $\tilde{t}$ with a certain ASM-term $\sigma(v)$. Thus, the value $f_X(a_1, \ldots, a_n)$ of t in X is also the value of $\tilde{t}$ in X when each variable v is assigned the value $\mathrm{Val}(\sigma(v), X, \xi)$ and similarly with X' in place of X. By induction hypothesis, these values assigned to the variables are the same in X and X'. (We use here that $\sigma(v)$ is a *proper* subterm of t, which is correct because t begins with a function symbol from Υ.) Furthermore, since $\sigma(v)$ begins with an external function symbol, its value is in $\mathrm{Range}(\dot{\xi})$. Thus, the assumption that (X, ξ) and (X', ξ) agree on $W(t)$, which contains $\tilde{t}$, ensures that $\tilde{t}$ has the same value in both X and X'. Therefore $f_X(a_1, \ldots, a_n) = f_{X'}(a_1, \ldots, a_n)$ as required. Since no queries are issued in this situation, we have completed the proof in the case that $f \in \Upsilon$.

There remains the case that $f \in \mathrm{E}$ and, as before, the subterms t_i have (the same) values a_i in X and X'. If $\hat{f}[a_1, \ldots, a_n] \in \mathrm{Dom}(\dot{\xi})$ then t gets the same value $\dot{\xi}(\hat{f}[a_1, \ldots, a_n])$ in both X and X', no queries are issued in either state, and the lemma is established in this case.

So assume that the query $\hat{f}[a_1, \ldots, a_n]$ is not in $\mathrm{Dom}(\dot{\xi})$. Then this query is the unique query produced by t in either state, and t has no value in either state, so again the conclusion of the lemma holds. $\square$

---

[3] "Outermost" means "maximal" in the sense that the occurrence in question is not properly contained in another such occurrence. In terms of the parse tree of t, it means that, on the path from the root of the whole tree to the root of the subtree given by the occurrence in question, there is no other occurrence of an external function symbol.

The preceding lemma easily implies the corresponding result for guards.

Lemma 4.5. For every guard φ there exists a finite set $W(\varphi)$ of Υ-terms such that, whenever (X, ξ) and (X', ξ) agree on $W(\varphi)$, then:

- If $\xi \vdash_X^\varphi q$ then $\xi \vdash_{X'}^\varphi q$.
- $\mathrm{Val}(\varphi, X, \xi) = \mathrm{Val}(\varphi, X', \xi)$.

Proof. We define $W(\varphi)$ by induction on φ. If φ is a Boolean term, then the preceding lemma provides the required $W(\varphi)$.

If φ is $(s \preceq t)$ then we define

$$W(s \preceq t) = W(s) \cup W(t) \cup \{\texttt{true}, \texttt{false}\}.$$

To check that the conclusion of the lemma is satisfied, we apply the previous lemma to see that, not only for the history ξ in question but also for any $\eta \trianglelefteq \xi$, if either of $\mathrm{Val}(s, X, \eta)$ and $\mathrm{Val}(s, X', \eta)$ is defined then so is the other, and similarly for t. With this information and with the knowledge that $\texttt{true}$ and $\texttt{false}$ denote the same element in X and X' (because of agreement on $W(\varphi)$, which contains $\texttt{true}$ and $\texttt{false}$), one finds by inspection of the relevant clauses in the semantics of guards that the conclusion of the lemma holds.

If φ is $\psi_0 \curlywedge \psi_1$ or $\psi_0 \curlyvee \psi_1$, then we set

$$W(\varphi) = W(\psi_0) \cup W(\psi_1) \cup \{\texttt{true}, \texttt{false}\}.$$

Finally, we set

$$W(\neg\psi) = W(\psi) \cup \{\texttt{true}, \texttt{false}\}.$$

Again, inspection of the relevant clauses in the semantics of guards shows that the conclusion of the lemma holds. $\square$

Finally, we prove the corresponding result for rules.

Lemma 4.6. For every rule R, there is a bounded exploration witness $W(R)$.

Proof. We define $W(R)$ by recursion on R as follows.

If R is an update rule $f(t_1, \ldots, t_n) := t_0$, then

$$W(R) = \bigcup_{i=0}^{n} W(t_i).$$

If R is $\texttt{issue} f(t_1, \ldots, t_n)$, then

$$W(R) = \bigcup_{i=1}^{n} W(t_i).$$

If R is $\texttt{fail}$ then $W(R)$ is empty.

If R is a conditional rule $\texttt{if } \varphi \texttt{ then } R_0 \texttt{ else } R_1$, then

$$W(R) = W(\varphi) \cup W(R_0) \cup W(R_1) \cup \{\texttt{true}, \texttt{false}\}.$$

If R is a parallel combination $\texttt{do in parallel } R_1, \ldots, R_k$ then

$$W(R) = \bigcup_{i=1}^{k} W(R_i).$$

That $W(R)$ serves as a bounded exploration witness for R is proved by induction on R. Every case of the inductive proof is trivial in view of the previous lemmas and the definition of the semantics of rules. $\square$

5. ALGORITHMS ARE EQUIVALENT TO ASMs

In this section, we shall prove the Abstract State Machine Thesis for interactive, small-step algorithms. That is, we shall prove that every algorithm (as defined in [5, Section 3]) is equivalent (as defined in [5, Section 4]) to an ASM (as in Definition 3.19).

Throughout this section, we assume that we are given an interactive, small-step algorithm A. By definition, it has a set $\mathcal{S}$ of states, a set $\mathcal{I}$ of initial states, a finite vocabulary Υ, a finite set Λ of labels, causality relations $\vdash_X$, sets $\mathcal{F}_X$ of final histories, subsets $\mathcal{F}_X^+$ and $\mathcal{F}_X^-$ of successful and failing final histories, and update sets $\Delta^+(X, \xi)$. Here and throughout this section, X ranges over states and ξ over histories for X. Furthermore, A has, by the Bounded Work Postulate of [5] and its corollaries, a bound B for the number and lengths of the queries issued in any state under any attainable history, and it has a bounded exploration witness W. Since W retains the property of being a bounded exploration witness if more Υ-terms are added to it, we may assume that W is closed under subterms and contains `true`, `false`, and some variable.

To define an ASM equivalent to A, we must specify, according to Definition 3.19,

- its vocabulary,
- its set of labels,
- its external vocabulary,
- its program,
- its template assignment,
- its set of states, and its set of initial states.

5.1. **Vocabulary, labels, states.** Some of these specifications are obvious, because the definition of equivalence requires that the vocabulary, the labels, the states, and the initial states be the same for our ASM as they are for the given algorithm A. It remains to define the external vocabulary, the template assignment, and the program.

Before proceeding, we note that Definition 3.19 requires $\mathcal{S}$ and $\mathcal{I}$ to be closed under isomorphisms and requires $\mathcal{S}$ to be closed under the transitions of the ASM. The first of these requirements is satisfied by our choice of $\mathcal{S}$ and $\mathcal{I}$ because A satisfies the Isomorphism Postulate. That the second requirement is also satisfied will be clear once we verify that the update sets and therefore the transition functions of A and of our ASM agree (at least on successful final histories), for the Step Postulate ensures that $\mathcal{S}$ is closed under the transitions of A.

5.2. **External vocabulary and templates.** To define the external vocabulary E and the template assignment for our ASM, we consider all templates, of length at most B, for the given set Λ of labels, in which the placeholders $\#i$ occur in order. (Recall that B is an upper bound on the lengths of queries issued by algorithm A in arbitrary states for arbitrary attainable histories.) These templates, which we call *standard templates*, can be equivalently described as the tuples obtained by taking any initial segment of the list $\#1, \#2, \ldots$ of placeholders and inserting elements of Λ into such a tuple, while keeping the total length of the tuple $\leq B$. We note that any potential query of length $\leq B$ (over any state) is obtained from a unique standard template by substituting elements of the state for the placeholders. We define the external vocabulary E and the template assignment simultaneously by putting into E one function symbol f for each standard template and

writing $\hat{f}$ for the standard template associated to f. Define an external function symbol f to be n-ary if $\hat{f}$ is a template for n-ary functions.

Remark 5.1. For many algorithms, the external vocabulary defined here is larger than necessary; many symbols in E won't occur in the program Π. One can, of course, discard such superfluous symbols once Π is defined. We chose the present definition of E in order to make it independent of the more complicated considerations involved in defining Π. $\square$

Remark 5.2. We have not specified — nor is there any need to specify — exactly what entities should serve as the external function symbols f associated to templates $\hat{f}$. The simplest choice mathematically would be to take the function symbols to be the standard templates themselves, but even with this choice, which would make $\hat{f} = f$, it would seem worthwhile to maintain the notational distinction between f, to be thought of as a function symbol, and $\hat{f}$, to be thought of as a template. $\square$

5.3. **Critical elements, critical terms, agreement.** The preceding discussion completes the easy part of the definition of our ASM; the hard part that remains is to define the program Π. Looking at the characterization in Lemma 4.3 of [5], we find that we have (trivially) satisfied the first requirement for the equivalence of our ASM and the given A (agreement as to states, initial states, vocabulary, and labels), and that we must construct Π so as to satisfy the remaining three requirements (agreement as to queries issued, finality, success, failure, and updates). Notice that these three requirements refer only to histories that are attainable for both algorithms. This means that, in constructing Π, we can safely ignore what A does with unattainable histories.

As in the proofs of the ASM thesis for other classes of algorithms in [10, 1, 4], we use the bounded exploration witness to gain enough control over the behavior of the algorithm A to match it with an ASM. The first step in this process is the following lemma, whose basic idea goes back to [10].

Definition 5.3. Let X be a state and ξ a history for it. An element $a \in X$ is *critical* for X and ξ if there is a term $t \in W$ and there are values in $\text{Range}(\dot{\xi})$ for the variables in t such that the resulting value for t is a. $\square$

Lemma 5.4 (Critical Elements). Let X be a state, ξ a coherent history for it, and a an element of X. Assume that one of the following holds.
- There is a query q such that $\xi \vdash_X q$ and a is one of the components of the tuple q.
- There is an update $\langle f, \langle b_1, \ldots, b_n \rangle, c \rangle \in \Delta^+(X, \xi)$ such that a is one of the b_i's or c.

Then a is critical for X and ξ.

Proof. The proof is very similar to the one in [2, Propositions 5.23 and 5.24], so we shall be rather brief here. We may assume, as an induction hypothesis, that the lemma holds when ξ is replaced with any proper initial segment of ξ. (This is legitimate because initial segments inherit coherence from ξ.) Because ξ is coherent, every query in its domain is caused by some proper initial segment. So all components in X of such a query are critical for that initial segment and therefore also critical for ξ.

Assume that a is not critical for X and ξ; we shall show that neither of the two hypotheses about a can hold.

Form a new state X', isomorphic to X, by replacing a by a new element a'. Since a is not critical, it is neither a component of a query in $\text{Dom}(\dot{\xi})$ nor an element of $\text{Range}(\dot{\xi})$.

Thus ξ is a history for X' as well as for X. Using again the assumption that a is not critical, one finds that (X, ξ) and (X', ξ) agree on W. As W is a bounded exploration witness for A, and as a is obviously neither a component of a query caused by ξ over X' nor a component in an update in $\Delta^+(X', \xi)$ (because $a \notin X'$), it follows that a is neither a component of a query caused by ξ over X nor a component in an update in $\Delta^+(X, \xi)$. $\qquad\square$

The construction of our ASM will be similar to that in [4, Section 5], but some additional work will be needed to take into account the timing information in histories and the possibility of incomplete but final histories.

The role played by element tags (or e-tags) and query tags (or q-tags) in [4] will now be played by ASM-terms, i.e., variable-free terms over the vocabulary $\Upsilon \cup E$. Some of these terms, those with outermost function-symbol in E, will, by Definition 3.3, obtain two kinds of possible values: the ordinary value which is an element of the state, and also a query-value, which is a potential query.

Definition 5.5 (Critical Terms). Recall that an ASM-term is a closed term of the vocabulary $\Upsilon \cup E$.

- A critical term of *level* 0 is a closed term in the bounded exploration witness W.
- If $t_1, \ldots, t_k$ are critical terms with maximal level n and f is a k-ary function symbol in E, then $f(t_1, \ldots, t_k)$ is a critical q-term of level $n + 1$.
- If $t \in W$ contains exactly the variables $x_1, \ldots, x_k$ and if $t_1, \ldots, t_k$ are critical q-terms with maximal level n, then the result of substituting t_i for x_i in t, $i = 1, \ldots, k$, is a critical term of level n.
- By a *critical term* we mean a critical term of some level.

$\qquad\square$

Because we arranged for W to contain a variable, the third clause of the definition implies that every critical q-term is a critical term (of the same level), so our terminology is consistent.

Since W and the external vocabulary E are finite, there are only finitely many critical terms of any one level.

Notice that, although they are obtained from the Υ-terms in W, our critical terms are ASM-terms. That is, they contain no variables, but they can contain external function symbols.

The values of ASM-terms, including in particular critical terms, for a given state X and history ξ, as well as the query-values of q-terms, were defined in Definition 3.3.

Recall also that, according to Lemma 3.6, any term that has a value in state X with respect to an initial segment of ξ will have the same value with respect to ξ itself, and that the same holds of query values. The next lemma records some related facts for future reference. Recall that two pairs (X, ξ) of a state and history are said to agree on W if the two histories are the same and every term in W gets the same values (if any) in both states when the variables are given values in the range of the history.

Lemma 5.6 (Invariance of Values).

- If $i : X \cong Y$ is an isomorphism, ξ is a history for X, and z is any ASM-term, then $i(\mathrm{Val}(z, X, \xi)) = \mathrm{Val}(z, Y, i(\xi))$. If z is a q-term, then also $i(\text{q-Val}(z, X, \xi)) = \text{q-Val}(z, Y, i(\xi))$.
- If (X, ξ) and (Y, ξ) agree on W, then $\mathrm{Val}(z, X, \xi) = \mathrm{Val}(z, Y, \xi)$ for all critical terms z, and $\text{q-Val}(z, X, \xi) = \text{q-Val}(z, Y, \xi)$ for all critical q-terms z.

Proof. The first assertion is proved by induction on terms, using the Isomorphism Postulate. The second is proved by induction on critical terms and critical q-terms, using the facts that all critical terms are, by definition, in the bounded exploration witness W and that the same history ξ is used on both sides of the claimed equations. □

Remark 5.7. An approximation to the intuition behind critical terms is that critical terms of level n represent (for a state X and history ξ) the elements of X and the queries that can play a role in the computation of our algorithm A during the first n rounds or phases of its interaction with the environment. This is based on the intuition that the bounded exploration witness W represents all the things the algorithm can do, with the environment's replies, to focus its attention on elements of X. At first, before receiving any information from the environment (indeed, before even issuing any queries), the algorithm can focus only on the values of closed terms from W, i.e., the values of critical terms of level 0. Using these, it can formulate and issue queries; these will be query-values of q-terms of level 1. Once some replies are received to those queries, the algorithm can focus on the values of non-closed terms from W with the replies as values for the variables. The replies are the values for the q-terms of level 1 that denote the issued queries, and so the elements to which the algorithm now pays attention are the values of critical terms of level ≤ 1. Using them, it assembles and issues queries, query-values of q-terms of level ≤ 2. The replies, used as values of the variables in terms from W, give the new elements to which the algorithm can pay attention, and these are the values of critical terms of level ≤ 2. The process continues similarly for later rounds of the interaction with the environment and correspondingly higher level terms.

One should, however, be careful not to assume too much about the connection between levels of critical terms and rounds of interaction. It is possible for a critical term t of level 1 to acquire a value only after many rounds of interaction, if, for example, the history happens to answer many other queries, one after the other, before finally getting to one that is needed for evaluating t. It is also possible for a critical term of high level to acquire a value earlier than its level would suggest. Consider, for example, a critical term of the form $f(f(f(0)))$, where f is an external function symbol and 0 is a constant symbol from Υ. If the history ξ contains just one reply, giving the query $\hat{f}[0_X]$ the value 0_X, then this suffices to give $f(f(f(0)))$ the value 0_X.

The following lemma formalizes the part of this intuitive explanation that we shall need later. □

Lemma 5.8 (Critical Terms Suffice). Let X be a state, ξ an attainable history for it, and n the length of ξ.

- Every query in $\mathrm{Dom}(\dot{\xi})$ is the query-value (for X and ξ) of some critical q-term of level $\leq n$.
- Every element of $\mathrm{Range}(\dot{\xi})$ is the value (for X and ξ) of some critical q-term of level $\leq n$.
- Every critical element for X and ξ is the value (for X and ξ) of a critical term of level $\leq n$.
- Every query in $\mathrm{Issued}_X(\xi)$ is the query-value (for X and ξ) of some critical q-term of level $\leq n + 1$.

Proof. We proceed by induction on the length n of the history ξ. As ξ is coherent, any query in its domain is issued by a proper initial segment $\eta \lhd \xi$. So, by induction hypothesis (applied

to the last clause), such a query is the query-value of a q-term of level $\leq \text{length}(\eta) + 1 \leq n$. This proves the first assertion of the lemma.

The second follows, because, if a query in $\text{Dom}(\dot{\xi})$ is the query-value of a q-term of level $\leq n$, then the reply given by ξ is the value of the same term.

For the third assertion, consider any critical element, say the value of a term $t \in W$ when the variables of t are given certain values in $\text{Range}(\dot{\xi})$. By the second assertion already proved, these values of the variables are also the values of certain critical q-terms of level $\leq n$. Substituting these terms for the variables in t, we obtain a critical term of level $\leq n$ whose value is the given critical element.

For the final assertion, consider any query issued by ξ. It has length at most B (by our choice of B), so it is obtained by substituting elements of X for the placeholders in some standard template. That is, it has the form $\hat{f}[a_1, \ldots, a_k]$ for some external function symbol f and some elements $a_i \in X$. By Lemma 5.4, each a_i is critical with respect to X and ξ. By the third assertion already proved, each a_i is the value of some critical term t_i of level $\leq n$. Then our query $\hat{f}[a_1, \ldots, a_k]$ is the query-value of the critical q-term $f(t_1, \ldots, t_k)$ of level $\leq n + 1$. $\qquad\square$

As indicated earlier, we can confine our attention to attainable histories. The lengths of these are bounded by B, and so we may, by the lemma just proved, confine our attention to critical terms of level at most B. In particular, only a finite set of critical terms will be under consideration.

We have the following partial converse to the last statement of Invariance of Values Lemma 5.6. We abbreviate the phrase "pair consisting of a state and an attainable history for it" as "attainable pair."

Lemma 5.9 (Agreement)**.** Let $(X, \xi), (Y, \xi)$ be attainable pairs with ξ of length n. If they agree as to the values of all critical terms of level $\leq n$, then they agree on W.

Proof. Note that in the assumption we didn't mention agreement as to query-values. But (X, ξ) and (Y, ξ) will agree as to query-values of critical q-terms of level n as soon as they agree as to the values of critical terms of levels $< n$.

Let $t \in W$. We need to prove that it takes the same value in (X, ξ) and (Y, ξ) when all variables in t are given values in $\text{Range}(\dot{\xi})$. But values in $\text{Range}(\dot{\xi})$ are, by the Critical Terms Suffice Lemma 5.8, the values of some critical q-terms of level $\leq n$. Substituting these terms for the variables in t gives us, by definition, a critical term of level $\leq n$, where by assumption (X, ξ) and (Y, ξ) agree. $\qquad\square$

5.4. **Descriptions, similarity.** The following definitions are intended to capture all the information about a state and history that can be relevant to the execution of our algorithm A. That they succeed will be the content of the subsequent discussion and lemmas.

Definition 5.10. Let (X, ξ) be an attainable pair. Let n be the length of ξ. (Recall that n is finite and in fact $\leq B$.) Define the *truncation* $\xi-$ of ξ to be the initial segment of ξ of length $n - 1$ (undefined if $n = 0$). The *description* $\delta(X, \xi)$ of X and ξ is the Kleene conjunction of the following guards:

- all equations $s = t$ and negated equations $\neg(s = t)$ that have value `true` in (X, ξ), where s and t are critical terms of level $\leq n$, and

- all timing inequalities $(u \prec v)$ and $(u \preceq v)$ that have value $\texttt{true}$ in (X, ξ), where u and v are critical q-terms of level $\leq n$, and where q-Val(v, X, ξ) exists and is in Issued$_X(\xi-)$.

$\square$

Some comments may help to clarify the last clause here, about timing inequalities. First, recall that the strict inequality $(u \prec v)$ is merely an abbreviation of $\neg(v \preceq u)$.

Second, although we explicitly require only v to have a query-value in Issued$_X(\xi-)$, the same requirement for u is included in the requirement that $(u \preceq v)$ or $(u \prec v)$ is true. Indeed, inspection of the definition of the semantics of timing guards (in Definition 3.7) shows that the q-term u must have a value, and this is possible only if u has a query-value in Dom$(\dot{\xi})$. Since ξ is coherent, it follows that q-Val(u, X, ξ) must be in Issued$_X(\xi-)$.

Third, if $n = 0$ then $\xi-$ is undefined, and as a result $\delta(X, \xi)$ contains no timing inequalities.

Our definition of the description of X and ξ is not complete on the syntactic level, for it does not specify the order or parenthesization of the conjuncts in the Kleene conjunction. That is, it is complete only up to associativity and commutativity of $\curlywedge$. The reader is invited to supply any desired syntactic precision; it will never be used. The choice of order and parenthesization of conjuncts makes no semantic difference; the Kleene conjunction and disjunction are commutative and associative as far as truth values and issued queries are concerned.

We shall sometimes refer to descriptions of attainable pairs as *attainable descriptions*, even though "attainable" is redundant here because the descriptions have been defined only for attainable pairs.

The following lemma and its corollary provide useful information about the q-terms occurring in a description.

Lemma 5.11. Let (X, ξ) be an attainable pair, $n \geq 1$ the length of ξ, and v a q-term. The following are equivalent.

(1) v occurs in $\delta(X, \xi)$.
(2) v occurs as one side of a timing inequality in $\delta(X, \xi)$.
(3) v is a critical q-term of level $\leq n$ and it has a query-value q-Val$(v, X, \xi-)$ that is in Issued$_X(\xi-)$.

Proof. Since the implication from (2) to (1) is trivial, we prove that (3) implies (2) and that (1) implies (3).

Suppose first that (3) holds. Let q be any query in the last equivalence class of the preorder in ξ. As ξ is attainable, $q \in$ Issued$_X(\xi-)$. Also, by Lemma 5.8, $q =$ q-Val(u, X, ξ) for some critical q-term u of level $\leq n$. Because q is in the last equivalence class with respect to ξ, Val(u, X, ξ) exists but Val$(u, X, \xi-)$ does not. Now if q-Val$(v, X, \xi-)$, which is also q-Val(v, X, ξ), is in Dom$(\dot{\xi})$, then Val(v, X, ξ) exists and so $\delta(X, \xi)$ contains the conjunct $(v \preceq u)$. Otherwise, Val(v, X, ξ) does not exist, and so $\delta(X, \xi)$ contains the conjunct $(u \prec v)$. In either case, (2) holds.

Finally, we assume (1) and deduce (3). Inspection of the definition of descriptions reveals that any q-term v that occurs in $\delta(X, \xi)$ must be a sub-q-term either of some critical term of level $\leq n$ that has a value with respect to (X, ξ) or of some critical q-term of level $\leq n$ that either has a value with respect to (X, ξ) or at least has a query-value that is issued with respect to $(X, \xi-)$. In any case it follows, thanks to the attainability (and in particular the coherence) of ξ, that (3) holds. $\square$

Corollary 5.12. The q-terms that occur in the description of an attainable pair (X, ξ) depend only on X and $\xi-$, not on the last equivalence class in the preorder of $\mathrm{Dom}(\dot{\xi})$.

Proof. Immediate from the third of the equivalent statements in the lemma. $\qquad\square$

Clearly, $\delta(X, \xi)$ is a guard, and $\mathrm{Val}(\delta(X, \xi), X, \xi) = \texttt{true}$. The next lemma shows that descriptions are invariant under two important equivalence relations on attainable pairs (X, ξ).

Lemma 5.13 (Invariance of Descriptions)**.** Let (X, ξ) be an attainable pair.
- If (Y, ξ) is an attainable pair (with the same ξ) agreeing with (X, ξ) on W, then they have the same descriptions.
- If (Y, η) is an attainable pair isomorphic to (X, ξ), then they have the same descriptions.

Proof. To see that the first statement is true, use the second clause of Invariance of Values Lemma 5.6 to establish that the same critical terms occur in $\delta(X, \xi)$ and $\delta(Y, \xi)$ in the same roles. To see that the second statement is true, use the first clause of the same lemma. $\qquad\square$

Thus, each of agreement and isomorphism is a sufficient condition for similarity in the sense of the following definition. We shall see later, in Corollary 5.16, that the composition of agreement and isomorphism is not only sufficient but also necessary for similarity.

Definition 5.14. Two attainable pairs are *similar* if they have the same descriptions. $\qquad\square$

The next lemma describes the other states and histories in which $\delta(X, \xi)$ is true, and thus leads to a characterization of similar attainable pairs.

Lemma 5.15. Let (X, ξ) and (Y, η) be attainable pairs. Suppose $\delta(X, \xi)$ has value $\texttt{true}$ in (Y, η). Then
- the length of η is at least the length of ξ;
- there is an attainable pair (Z, η') isomorphic to (X, ξ), such that η' is an initial segment of η and (Z, η') agrees with (Y, η') on W.

In other words, any (Y, η) that satisfies the description of (X, ξ) can be obtained from (X, ξ) by the following three-step process. First, replace (X, ξ) by an isomorphic copy (Z, η'). Second, leaving the history η' unchanged, replace Z by a new state Y but maintain agreement on the bounded exploration witness W. Third, extend the history η' by adding new items strictly after the ones in η', so that η' is an initial segment of the resulting η.

Notice that, by virtue of the isomorphism of (X, ξ) and (Z, η'), we can describe η' more specifically as the initial segment of η of the same length as ξ.

Proof. We proceed by induction on the length n of the history ξ.

Length: η **is not shorter than** ξ**.** Choose one query from each of the n equivalence classes in $\mathrm{Dom}(\dot{\xi})$, say q_j from the jth equivalence class. Letting $\xi \restriction j$ denote the initial segment of ξ of length j, and applying Lemma 5.8, we express each q_j as the query-value, with respect to $(X, \xi \restriction j)$, of some critical q-term u_j of level $\leq j$. Thus, u_j has a value $\dot{\xi}(q_j)$ with respect to $\xi \restriction j$ but not with respect to $\xi \restriction (j-1)$. Thus, $\delta(X, \xi)$ includes the conjuncts $(u_j \prec u_{j+1})$ for $j = 1, 2, \ldots, n-1$ and also the conjunct $u_n = u_n$. So these conjuncts must also be true in (Y, η), which means that η has length at least n.

Construction of (Z, η')**.** Our next step will be to define a certain isomorphic copy (Z, η') of (X, ξ). Afterward, we shall verify that η' has the other properties required.

We may assume, by replacing (X, ξ) with an isomorphic copy if necessary, that X is disjoint from Y. Next, obtain an isomorphic copy Z of X as follows. For each critical term t of level $\leq n$, if $\mathrm{Val}(t, X, \xi)$ exists, then remove this element from X and put in its place the element $\mathrm{Val}(t, Y, \eta)$ of Y. To see that this makes sense, we must observe two things. First, the equation $t = t$ is one of the conjuncts in $\delta(X, \xi)$ and is therefore true for Y and η. Thus, the replacement element $\mathrm{Val}(t, Y, \eta)$ exists. Second, if the same element of X is also $\mathrm{Val}(t', X, \xi)$ for another critical term t' of level $\leq n$, then the equation $t' = t$ is a conjunct in $\delta(X, \xi)$ and is therefore true for Y and η. Thus, $\mathrm{Val}(t', Y, \eta){=}\mathrm{Val}(t, Y, \eta)$, which means that the replacement element is uniquely defined.

Let i be the obvious isomorphism from X to Z, sending each of the replaced elements $\mathrm{Val}(t, X, \xi)$ to its replacement $\mathrm{Val}(t, Y, \eta)$ and sending all the other elements of X to themselves. Let $\eta' = i(\xi)$; this is the history for Z obtained by applying i to all components from X in the queries in $\mathrm{Dom}(\dot{\xi})$ and to all the replies in $\mathrm{Range}(\dot{\xi})$. Because of the isomorphism, it is clear that (Z, η') is, like (X, ξ), an attainable pair and that η' has the same length n as ξ.

Values: $\dot{\eta}'$ is a subfunction of $\dot{\eta}$. Consider any query $q = \hat{f}[a_1, \ldots, a_k] \in \mathrm{Dom}(\dot{\xi})$ and its reply $b = \dot{\xi}(q)$. Thus, $i(q) \in \mathrm{Dom}(i(\dot{\xi}))$, and $i(\dot{\xi})(i(q)) = i(b)$. Furthermore, every element of $\mathrm{Dom}(i(\dot{\xi}))$ is $i(q)$ for some such q. By Lemma 5.8, all the a_j are values in (X, ξ) of certain critical terms t_j of level $< n$, and so b is the value of the critical term $f(t_1, \ldots, t_k)$ of level $\leq n$. In forming Z, we replaced the elements a_j by the values $i(a_j)$ of the t_j's in (Y, η), and we replaced b by $i(b)$, the value in (Y, η) of $f(t_1, \ldots, t_k)$. But this last value is, by definition, the result of applying $\dot{\eta}$ to the query that is the query-value of $f(t_1, \ldots, t_k)$, namely the query $\hat{f}[i(a_1), \ldots, i(a_k)] = i(q)$. That is, $i(b) = \dot{\eta}(i(q))$. This shows that, whenever $i(\dot{\xi})$ maps a query $i(q)$ to a reply $i(b)$, then so does $\dot{\eta}$; in other words, $\dot{\eta}'$ is a subfunction of $\dot{\eta}$.

Order: $\leq_{\eta'}$ is a sub-preorder of $\leq_\eta$. We next show that the preordering of η' agrees with that of η. Consider an arbitrary $q \in \mathrm{Dom}(\dot{\xi})$, and suppose it is in the jth equivalence class with respect to the preorder given by ξ. So, as ξ is coherent, $q \in \mathrm{Issued}_X(\xi {\restriction} (j - 1))$, and so, by the last part of Lemma 5.8, we have a critical q-term u of level $\leq j$ such that $q = \text{q-Val}(u, X, \xi {\restriction} (j - 1))$. Note that $\mathrm{Val}(u, X, \xi {\restriction} (j - 1))$ does not exist, because $q \notin \mathrm{Dom}(\dot{\xi} {\restriction} (j - 1))$.

We wish to apply the induction hypothesis to $(X, \xi {\restriction} (j - 1))$. To do so, we observe that $\delta(X, \xi {\restriction} (j - 1))$ is a subconjunction of $\delta(X, \xi)$ and is therefore true in (Y, η). So we can apply the induction hypothesis and find that $(X, \xi {\restriction} (j - 1))$ is isomorphic to an attainable pair that agrees with $(Y, \eta {\restriction} (j - 1))$. By Lemma 5.6, u has a query-value but no value in $(Y, \eta {\restriction} (j - 1))$. Inspection of the definitions shows that its query-value is $i(q)$.

If $j < n$, i.e., if $q \in \mathrm{Dom}(\xi-)$, then we can also apply the induction hypothesis to $(X, \xi {\restriction} j)$, in which u has a value. We conclude that u has a value in $(Y, \eta {\restriction} j)$. Since it had a query-value but no value in $(Y, \eta {\restriction} (j - 1))$, we conclude that its query-value, $i(q)$ must be in exactly the jth equivalence class with respect to η.

If, on the other hand, $j = n$, i.e., if q is in the last equivalence class with respect to ξ, then this last application of the induction hypothesis is not available. Nevertheless, since $q \in \mathrm{Dom}(\dot{\xi})$, we know that u has a value in (X, ξ), so $\delta(X, \xi)$ contains the conjunct $u = u$, so this conjunct is true also in (Y, η), and so u has a value in (Y, η). This means that $i(q)$, the query-value of u, is in $\mathrm{Dom}(\dot{\eta})$. We saw earlier that it is not in $\mathrm{Dom}(\dot{\eta} {\restriction} (j - 1))$, where now $j = n$. So $i(q)$ is in the nth equivalence class or later with respect to η.

What we have proved so far suffices to establish that if $q <_\xi q'$ then $i(q) <_\eta i(q')$ and that the same holds for non-strict inequalities except in the case that both q and q' are in the last equivalence class with respect to ξ. In this exceptional case, we know that $i(q)$ is the query-value, already existing in $(Y, \eta \restriction (n-1))$, of u (as above), yet u has no value in $(Y, \eta \restriction (n-1))$. This means that the smallest m for which $\text{Val}(u, Y, \eta \restriction m)$ exists is the m such that $i(q)$ is in the mth equivalence class with respect to η. Repeating the argument with an analogously defined q-term u' for q', and using the fact that $\delta(X, \xi)$ contains the conjuncts $(u \preceq u')$ and $(u' \preceq u)$, which means that these conjuncts are also true in (Y, η), we find that $i(q)$ and $i(q')$ are in the same equivalence class with respect to η.

This completes the proof that η' — including both the answer function and the pre-order — is the restriction of η to some subset of its domain. In fact, we have shown more, namely that, for $j < n$, i maps the jth equivalence class with respect to ξ into the jth equivalence class with respect to η, and that it maps the last (n^{th}) equivalence class with respect to ξ into a single equivalence class — possibly the nth and possibly later — with repect to η.

The next step is to show that $\eta' = i(\xi)$ is an initial segment of η. This will imply that, in the preceding summary of what was already proved, both occurrences of "into" can be improved to "onto" and "possibly the nth and possibly later" can be improved to "the nth".

Initial segment: η' is an initial segment of η. Suppose, toward a contradiction, that $\text{Dom}(\dot\eta')$ is not an initial segment of $\text{Dom}(\dot\eta)$ (with respect to $\leq_\eta$). So there exist some $q \in \text{Dom}(\dot\eta) - \text{Dom}(\dot\eta')$ and some $q' \in \text{Dom}(\dot\xi)$ (and thus $i(q') \in \text{Dom}(\dot\eta')$) such that $q \leq_\eta i(q')$. Among all such pairs q, q', fix one for which q occurs as early as possible in the preorder $\leq_\eta$. Since $q' \in \text{Dom}(\dot\xi)$, we can fix a critical q-term u' of level $\leq n$ with q-$\text{Val}(u', X, \xi) = q'$ and thus, by definition of i, q-$\text{Val}(u', Y, \eta) = i(q')$. We record for future reference that, since q-$\text{Val}(u', X, \xi) \in \text{Dom}(\dot\xi)$, u' has a value with respect to ξ.

Consider the initial segment of η up to but not including q. By what we have already proved (and our choice of q as the earliest possible), it is $i(\zeta)$ for some proper initial segment ζ of ξ — proper because it doesn't contain q'. In particular, ζ has length at most $n-1$, and so we know, by induction hypothesis, that the lemma is true with ζ in place of ξ. (As before, the lemma can be applied because $\delta(X, \zeta)$ is a subconjunction of $\delta(X, \xi)$, which is true in (Y, η).) So we conclude that (X, ζ) is isomorphic to an attainable pair that agrees on W with $(Y, i(\zeta))$.

Since $i(\zeta)$ is the initial segment of η ending just before q, and since η is a coherent history, we know that $q \in \text{Issued}_Y(i(\zeta))$. By the Critical Terms Suffice Lemma 5.8, q is the query-value in $(Y, i(\zeta))$, and therefore also in (Y, η), of some critical q-term u of level $\leq n$. Thanks to the isomorphism between (X, ζ) and an attainable pair agreeing with $(Y, i(\zeta))$, we have that u also has a query-value, say q'', in (X, ζ) and this value is in $\text{Issued}_X(\zeta)$ and, a fortiori, in $\text{Issued}_X(\xi)$. By definition of i, $i(q'') = q$. As q was chosen outside $\text{Dom}(i(\dot\xi)) = i(\text{Dom}(\dot\xi))$, it follows that $q'' \notin \text{Dom}(\dot\xi)$. From this and $q' \in \text{Dom}(\dot\xi)$, we conclude that $(u' \prec u)$ is one of the conjuncts in $\delta(X, \xi)$ and is therefore true in (Y, η). Since u has a value in the initial segment of η up to and including q (one equivalence class beyond $i(\zeta)$), we infer that u' must have a value in $(Y, i(\zeta))$. That means that the query-value of u', namely $i(q')$ must be in the domain of $i(\zeta)$, i.e., $i(q') <_\eta q$. This contradicts the original choice of q and q', and this contradiction completes the proof that η' is an initial segment of η.

Agreement: (Z, η') **and** (Y, η') **agree on** W**.** It remains to prove that the attainable pairs (Z, η') and (Y, η') agree on W. We prove this in three steps.

First, we show that, if z is any critical term of level $\leq n$, then

$$i(\mathrm{Val}(z, X, \xi)) = \mathrm{Val}(z, Y, \eta').$$

This is almost the definition of i, which says that $i(\mathrm{Val}(z, X, \xi)) = \mathrm{Val}(z, Y, \eta)$. Our task is to replace η on the right side of this equation with η'. That is, we must show that, if $\mathrm{Val}(z, X, \xi)$ exists (and therefore $\mathrm{Val}(z, Y, \eta)$ exists), then $\mathrm{Val}(z, Y, \eta')$ exists, because then we shall have $\mathrm{Val}(z, Y, \eta') = \mathrm{Val}(z, Y, \eta)$ by the monotonicity of values. We proceed by induction on the level of z. The only non-trivial case, i.e., the only case where changing η to η' could matter, is the case that z is a q-term. The possibility that we must exclude is that q-$\mathrm{Val}(z, Y, \eta)$ (which is also q-$\mathrm{Val}(z, Y, \eta')$ as the induction hypothesis applies to the arguments of z) is in the domain of η but not in the domain of η'. But q-$\mathrm{Val}(z, X, \xi)$ exists and is in the domain of ξ (because $\mathrm{Val}(z, X, \xi)$ exists), and its image under i is, by definition of i, q-$\mathrm{Val}(z, Y, \eta)$. So this image is in the domain of $i(\xi) = \eta'$, as desired.

Second, we observe that, since i is an isomorphism from X to Z and sends ξ to η', we have $i(\mathrm{Val}(z, X, \xi)) = \mathrm{Val}(z, Z, \eta')$. Combining this with the result established in the preceding paragraph, we have

$$\mathrm{Val}(z, Z, \eta') = \mathrm{Val}(z, Y, \eta')$$

for all critical terms z of level $\leq n$.

Finally, an application of the Agreement Lemma 5.9 completes the proof that (Z, η') and (Y, η') agree on W. $\qquad\square$

Corollary 5.16 (Factorization). *Let (X, ξ) and (Y, η) be similar attainable pairs. Then there is a state Z such that η is an attainable history for Z, (Z, η) agrees with (Y, η) on W, and (Z, η) is isomorphic to (X, ξ).*

Proof. We can apply Lemma 5.15 to (X, ξ) and (Y, η) in either order, since each satisfies the other's description. Thus ξ and η have the same length, and the η' of the lemma is simply η. The rest of the corollary is contained in the lemma. $\qquad\square$

Corollary 5.17 (Similarity Suffices). *Let (X, ξ) and (Y, η) be similar attainable pairs. Let n be the length of ξ (and of η, by Corollary 5.16). Then*

- *If u is a q-term of level $\leq n+1$ and $\xi \vdash_X$ q-$\mathrm{Val}(u, X, \xi)$ then $\eta \vdash_Y$ q-$\mathrm{Val}(u, Y, \eta)$.*
- *If ξ is in $\mathcal{F}_X^+$ or $\mathcal{F}_X^-$, then η is in $\mathcal{F}_Y^+$ or $\mathcal{F}_Y^-$, respectively.*
- *If $\Delta^+(X, \xi)$ contains an update $\langle f, \langle a_1, \ldots, a_k \rangle, a_0 \rangle$ where each a_i is $\mathrm{Val}(t_i, X, \xi)$ for a critical term t_i of level $\leq n$, then $\Delta^+(Y, \eta)$ contains the update $\langle f, \langle a_1', \ldots, a_k' \rangle, a_0' \rangle$ where each a_i' is $\mathrm{Val}(t_i, Y, \eta)$.*

Proof. Apply the preceding corollary to get Z such that (Z, η) agrees with (Y, η) on W and is isomorphic to (X, ξ). Because of the agreement on the bounded exploration witness W, we have all the desired conclusions with (Z, η) in place of (X, ξ). To complete the proof, we can replace (Z, η) with (X, ξ), thanks to the Isomorphism Postulate and the fact that isomorphisms respect evaluation of terms. $\qquad\square$

We shall also need the notion of a *successor* of an attainable description. This corresponds to adjoining one new equivalence class at the end of a history, while leaving the state unchanged. That is, $\delta(X, \xi)$ is a successor of $\delta(X, \xi-)$, and $\delta(X, \xi-)$ is the *predecessor* of $\delta(X, \xi)$.

Remark 5.18. To avoid possible confusion, we emphasize that a successor of $\delta(X, \eta)$ need not be of the form $\delta(X, \xi)$ with $\xi- = \eta$. It could instead be of the form $\delta(Y, \xi)$ for some other pair (Y, ξ) such that $(Y, \xi-)$ is similar to (X, η), and there might be no way to extend η so as to obtain similarity with (Y, ξ). For a simple example, suppose the bounded exploration witness W contains only true, false, undef, and a variable. Let X be a structure containing only the three elements that are the values of true, false, and undef, and let Y be like X but with one additional element a. Suppose further that the algorithm is such that a single query q, say $\langle \text{true} \rangle$, is caused by the empty history $\varnothing$ in every state. Then $(X, \varnothing)$ and $(Y, \varnothing)$ agree on W, and Y admits an attainable history ξ with $\text{Dom}(\dot{\xi}) = \{q\}$ and with $\dot{\xi}(q) = a$. Then, since $\xi- = \varnothing$, we have that $\delta(Y, \xi)$ is a successor of $\delta(Y, \varnothing) = \delta(X, \varnothing)$. But there is no history ζ for X such that $\delta(Y, \xi) = \delta(X, \zeta)$; such a ζ would have to map q to a value distinct from true, false, and undef, and X has no such element.   □

The use of the definite article in "the predecessor" is justified by the following observation, showing that $\delta(X, \xi-)$ is completely determined by $\delta(X, \xi)$. Thus, "predecessor" is a well-defined operation on attainable descriptions of non-zero length. Of course the situation is quite different for successors; one description can have many successors because there are in general many ways to extend an attainable history by appending one more equivalence class.

Corollary 5.19. Let (X, ξ) and (Y, η) be similar attainable pairs, and assume the (common) length of ξ and η is not zero. Then $\delta(X, \xi-) = \delta(Y, \eta-)$.

Proof. By Corollary 5.16, we have an isomorphism $i : (X, \xi) \cong (Z, \eta)$ such that (Z, η) agrees with (Y, η) on W. Since the isomorphism i must, in particular, respect the pre-orderings, it follows immediately that i is also an isomorphism from $(X, \xi-)$ to $(Z, \eta-)$. From the definition of agreement, it follows immediately that $(Z, \eta-)$ and $(Y, \eta-)$ agree on W. Thus, by Lemma 5.13, $(X, \xi-)$ and $(Y, \eta-)$ are similar.   □

The following information about successors will be useful when we verify that the ASM that we produce is equivalent to the given algorithm A.

Lemma 5.20. Suppose (X, ξ) is an attainable pair and δ' is an attainable description that is a successor of $\delta(X, \xi)$. Then $\delta' = \delta(Y, \eta)$ for some attainable pair (Y, η) such that

- $\xi = \eta-$,
- (X, ξ) and (Y, ξ) agree on W.

Proof. By definition of successor, we have an attainable pair (Z, θ) such that $\delta' = \delta(Z, \theta)$ and $\delta(X, \xi) = \delta(Z, \theta-)$. This last equality implies, by Corollary 5.16, that (X, ξ) and (Y, ξ) agree on W for some attainable pair (Y, ξ) isomorphic to $(Z, \theta-)$. Use the isomorphism to transport θ to an attainable history η for Y. Then $\delta' = \delta(Z, \theta) = \delta(Y, \eta)$ because of the isomorphism, and $\eta-$ is the image, under the isomorphism, of $\theta-$, i.e., $\eta- = \xi$.   □

5.5. **The ASM program.** We are now ready to describe the ASM program that will simulate our given algorithm A. Its structure will be a nested alternation of conditionals and parallel combinations, with updates, issue rules, and fail as the innermost constituents. The guards of the conditional subrules will be attainable descriptions. Recall that the critical terms involved in attainable descriptions all have levels $\leq B$, and there are only finitely many such terms and therefore only finitely many attainable descriptions. An

attainable description $\delta(X, \xi)$ will be said to have *depth* equal to the length of ξ. Lemma 5.15 ensures that this depth depends only on the description $\delta(X, \xi)$, not on the particular attainable pair (X, ξ) from which it is obtained. Notice that the definition of descriptions immediately implies that any critical term occurring in a description has level $\leq$ the depth of the description.

We construct the program Π for an ASM equivalent to the given algorithm A as follows. Π is a parallel combination, with one component for each attainable description δ of depth zero. We describe the component associated to δ under the assumption that δ is not final, by which we mean that, in the attainable pairs (X, ξ) with description δ, the history ξ is not final; we shall return later to the final case. (Recall that, by Corollary 5.17, whether ξ is final in X depends only on the description $\delta(X, \xi)$, so our case distinction here is unambiguous.)

The component associated to a non-final δ is a conditional rule of the form if δ then R_δ, i.e., a conditional whose guard is δ itself. The body R_δ is a parallel combination, with one component for each successor δ' of δ.

When δ' is not final, the associated component is a conditional rule if δ' then $R_{\delta'}$. The body $R_{\delta'}$ here is a parallel combination, with one component for each successor δ'' of δ'.

Continue in this manner until a final description ε is reached. Since the depth increases by one when we pass from a description to a successor, and since all attainable histories have length (i.e., the depth of their descriptions) at most B, we will have reached final descriptions after at most B iterations of the procedure. The component associated to a final description $\varepsilon = \delta(X, \xi)$ is if ε then R_ε endif, where R_ε is the parallel combination of the following:

- fail if $\xi \in \mathcal{F}_X^-$,
- issue u if u is a q-term of level at most one more than the length of ξ (that is, the depth of ε) and $\xi \vdash_X$ q-Val(u, X, ξ), and
- $f(t_1, \ldots, t_k) := t_0$ if the t_i are critical terms of level at most the length of ξ and they have values $a_i = \mathrm{Val}(t_i, X, \xi)$ such that $\langle f, \langle a_1, \ldots, a_k \rangle, a_0 \rangle \in \Delta^+(X, \xi)$.

It is important to note that, although the attainable pair (X, ξ) was used in the specification of these components, they actually depend only on the description ε, by Corollary 5.17. This completes the definition of the program Π.

Remark 5.21. As in previous work on the ASM thesis, this program Π is designed specifically for the proof of the thesis. That is, it works in complete generality and it admits a fairly simple, uniform construction. For practical programming of specific algorithms, there will normally be ASM programs far simpler than the one produced by our general method.

5.6. **Equivalence.** It remains to show that the ASM defined by Π is equivalent to the given algorithm A. For brevity, we sometimes refer to this ASM as simply Π.

Theorem 5.22. *The ASM defined by Π together with $\mathcal{S}$, $\mathcal{I}$, Υ, Λ, E, and the template assignment of subsection 5.2 is equivalent to algorithm A.*

Proof. Referring to Lemma 4.3 of [5], we see that it suffices to show the following, for every pair (X, ξ) that is attainable for both the algorithm A and our ASM.

(1) Issued$_X(\xi)$ is the same for our ASM as for A.
(2) If ξ is in $\mathcal{F}_X^+$ or $\mathcal{F}_X^-$ with respect to one of A and our ASM, then the same is true with respect to the other.

(3) If $\xi \in \mathcal{F}_X^+$, then $\Delta^+(X, \xi)$ is the same with respect to our ASM and with respect to A.

Consider, therefore, an attainable pair (X, ξ) (with respect to A) and the behavior of our ASM in this pair.

Let n be the length of ξ, and for each $m \leq n$ let $\xi \restriction m$ be the initial segment of ξ of length m. According to Lemma 5.15, the only attainable descriptions satisfied by (X, ξ) are those of the form $\delta(X, \xi \restriction m)$, one of each depth $m \leq n$.

Issuing Queries.

We begin by analyzing the queries issued by our ASM in state X with history ξ. (Parts of this analysis will be useful again later, when we analyze finality, success, failure, and updates.) For readability, our analysis will be phrased in terms of the ASM performing various actions, such as issuing queries or passing control to a branch of a conditional rule. Of course, this could be rewritten more formally in terms of the detailed semantics of ASMs, but the formalization seems to entail more costs, both for the reader and for the authors, than benefits.

The ASM acting in state X with history ξ begins, since Π is a parallel combination, by executing all the components associated with attainable descriptions of depth 0. Recall that these components are conditional rules whose guards are the descriptions themselves. These descriptions contain only critical terms of depth 0, so there are no external function symbols here. Therefore, no queries result from the evaluation of the guards. By Lemma 5.15 the ASM finds exactly one of the guards to be true, namely $\delta(X, \xi \restriction 0)$, and it proceeds to execute the body $R_{\delta(X, \xi \restriction 0)}$ of this conditional rule.

Let us suppose, temporarily, that $n > 0$, so, as ξ is attainable, $\xi \restriction 0$ is not final. (We shall return to the other case later.) So $R_{\delta(X, \xi \restriction 0)}$ is a parallel combination, and our ASM proceeds to execute its components. These are conditionals, whose guards δ' are the successors of $\delta(X, \xi \restriction 0)$. So these guards are $\delta(Y, \eta)$ for attainable pairs (Y, η) as in Lemma 5.20. In particular, η has length 1 and $\eta- = \xi \restriction 0$. (This last equation is redundant as both sides are histories of length 0, but we include it to match what will occur in later parts of our analysis.) Inspection of the definition of descriptions shows that every query issued during the evaluation of such a guard is also issued by the algorithm A operating in the attainable pair $(Y, \eta-) = (Y, \xi \restriction 0)$. Since $(Y, \xi \restriction 0)$ agrees with $(X, \xi \restriction 0)$ on W, these are queries issued by A in $(X, \xi \restriction 0)$.

The converse also holds. If a query q is issued by A in $(X, \xi \restriction 0)$, then there is an attainable history η for X in which q is in the first and only equivalence class of $\text{Dom}(\dot\eta)$; simply define η to give q an arbitrary reply and to do nothing more. By Lemma 5.8, q is the query-value of some q-term u of level 1, and therefore $\delta(X, \eta)$ contains the conjunct $u = u$. Thus, in evaluating the guard $\delta(X, \eta)$, our ASM will issue q.

Having evaluated the guards of depth 1, our ASM finds, according to Lemma 5.15, that exactly one of them is true, namely $\delta(X, \xi \restriction 1)$, so it proceeds to evaluate the corresponding body $R_{\delta(X, \xi \restriction 1)}$. Let us suppose, temporarily, that $n > 1$, so, as ξ is attainable, $\xi \restriction 1$ is not final. So $R_{\delta(X, \xi \restriction 1)}$ is a parallel combination, and our ASM proceeds to execute its components. These are conditionals, whose guards δ' are the successors of $\delta(X, \xi \restriction 1)$. So these guards are $\delta(Y, \eta)$ for attainable pairs (Y, η) as in Lemma 5.20. In particular, η has length 2 and $\eta- = \xi \restriction 1$. Inspection of the definition of descriptions shows that every query issued during the evaluation of such a guard is also issued by the algorithm A operating in

the attainable pair $(Y, \eta-) = (Y, \xi \upharpoonright 1)$. Since $(Y, \xi \upharpoonright 1)$ agrees with $(X, \xi \upharpoonright 1)$ on W, these are queries issued by A in $(X, \xi \upharpoonright 1)$.

The converse also holds. If a query q is issued by A in $(X, \xi \upharpoonright 1)$, but not already in $(X, \xi \upharpoonright 0)$, then there is an attainable history η for X, which has $\xi \upharpoonright 1$ as an initial segment, and in which q is in the second and last equivalence class of $\mathrm{Dom}(\dot{\eta})$; simply define η by extending $\xi \upharpoonright 1$ to give q an arbitrary reply, in a new, second equivalence class, and to do nothing more. By Lemma 5.8, q is the query-value of some critical term u of level 2, and therefore $\delta(X, \eta)$ contains the conjunct $u = u$. Thus, in evaluating the guard $\delta(X, \eta)$, our ASM will issue q.

The reader should, at this point, experience déjà vu, since the argument we have just given concerning the behavior of our ASM while executing $R_{\delta(X, \xi \upharpoonright 1)}$ is exactly parallel to the previous argument concerning $R_{\delta(X, \xi \upharpoonright 0)}$. The same pattern continues as long as the depths of the guards are $< n$ so that we have not arrived at a final history.

Consider now what happens when the ASM evaluates $R_{\delta(X, \xi \upharpoonright n)} = R_{\delta(X, \xi)}$. If the history ξ is not final, then the same argument as before shows that the ASM will issue, while evaluating the guards of the components of $R_{\delta(X, \xi)}$, the same queries as the original algorithm A. Furthermore, the ASM will find none of the guards here to be true, for these guards are descriptions of depth $n + 1$ and can, by Lemma 5.15, be satisfied only with histories of length at least $n + 1$. So the execution of the ASM produces no additional queries beyond those that we have already shown to agree with those produced by A.

There remains the situation that ξ is final for A and X. In this case, the components of $R_{\delta(X, \xi)}$ are no longer conditional rules, the evaluation of whose guards causes the appropriate queries to be issued by the ASM. Rather, the components are issue rules, updates, or `fail`. Only the issue rules here will result in new queries; the queries involved in evaluating the terms in update rules and in the issue rules have already been issued during the evaluation of guards. And the issue rules are chosen precisely to issue the queries that A would issue in (X, ξ).

This completes the proof that our ASM and A agree as to issuing queries. They therefore agree as to which histories are coherent.

Finality, Success, and Failure. We next consider which histories are declared final by our ASM. Suppose first that ξ is final for A in X. Then, as the preceding analysis of the ASM's behavior shows, the ASM will, after evaluating a lot of guards, find itself executing $R_{\delta(X, \xi)}$, which is a parallel combination of issue rules, update rules, or `fail`. The subterms of any update rules here will already have been evaluated during the evaluation of the guards, so ξ is final for these update rules. The same goes for the issue rules; their subterms have already been evaluated, and so ξ is final. Any history is final for `fail`. Thus ξ is final for all the components of $R_{\delta(X, \xi)}$ and is therefore final for $R_{\delta(X, \xi)}$ itself. From the definition of the semantics of parallel combinations and conditional rules, it follows that ξ is also final for Π, as required.

Now suppose that ξ is (attainable but) not final for A in state X. There will be some queries that have been issued by A but not answered, i.e., that are in $\mathrm{Issued}_X(\xi) - \mathrm{Dom}(\dot{\xi}) = \mathrm{Pending}_X(\xi)$, for otherwise ξ would be complete and attainable and therefore, by the Step Postulate, final. So our ASM will issue some queries whose answers are needed for the evaluation of the guards of some components of $R_{\delta(X, \xi)}$, but whose answers are not in ξ. Therefore, ξ is not a final history for the ASM in state X. This completes the proof that our ASM agrees with A as to finality of histories.

We check next that a final history ξ succeeds or fails for our ASM according to whether it succeeds or fails for A. It fails for our ASM if and only if, after evaluating all the guards and while executing $R_{\delta(X,\xi)}$, it encounters either `fail` or clashing updates (see the definition of failure for parallel combinations). By definition of our ASM, it encounters `fail` if and only if A fails in (X, ξ). Furthermore, it will not encounter clashing updates unless A fails, because, as we shall see below, it encounters exactly the updates produced by A, and these cannot, by the Step Postulate, clash unless A fails.

Updates. To complete the proof, we have to check what updates our ASM encounters. Our construction of Π is such that update rules are encountered only in the subrules $R_{\delta(X,\xi)}$ for final histories ξ. Furthermore, these update subrules are chosen to match the updates performed by A. So our ASM and A produce the same updates in any final history.

This completes the verification that our ASM is equivalent to the given algorithm A. $\square$

6. Concluding Remarks

Theorem 5.22 establishes the ASM thesis for small-step, interactive algorithms, as defined by the postulates of [5]. This completes the program of proving the ASM thesis in the small-step case. The case of parallel algorithms, without intrastep interaction, was treated in [1], but the task of combining intrastep interaction with parallelism remains for future work. Beyond that, there is the task of treating distributed algorithms.

The ASM syntax and semantics presented in Sections 2 and 3 serve to describe just what has to be added to the traditional ASM syntax and semantics of [9] in order to accommodate non-ordinary interaction. Essentially, one needs timing guards and the Kleene connectives. It remains to be seen whether these additions will also suffice to model all interactive parallel algorithms.

References

[1] Andreas Blass and Yuri Gurevich, "Abstract state machines capture parallel algorithms," *ACM Trans. Computational Logic*, 4 (2003) 578–651.

[2] Andreas Blass and Yuri Gurevich, "Ordinary Interactive Small-Step Algorithms, I," *ACM Trans. Computational Logic*, 7 (2006) 363–419.

[3] Andreas Blass and Yuri Gurevich, "Ordinary Interactive Small-Step Algorithms, II," *ACM Trans. Computational Logic*, to appear.

[4] Andreas Blass and Yuri Gurevich, "Ordinary Interactive Small-Step Algorithms, III," *ACM Trans. Computational Logic*, to appear.

[5] Andreas Blass, Yuri Gurevich, Dean Rosenzweig, and Benjamin Rossman, "Interactive Small-Step Computation I: Axiomatization," in preparation.

[6] Andreas Blass, Yuri Gurevich, Dean Rosenzweig, and Benjamin Rossman, "Composite interactive algorithms" (tentative title), in preparation.

[7] Yuri Gurevich, "A new thesis," Abstract 85T-68-203, *Amer. Math. Soc. Abstracts* 6 (August, 1985) p.317.

[8] Yuri Gurevich, "Evolving Algebras: An Introductory Tutorial," Bull. EATCS 43 (February 1991), 264–284. Reprinted with slight revisions in *Current Trends in Theoretical Computer Science: Essays and Tutorials*, G. Rozenberg and A. Salomaa (editors), World Scientific, 1993, 266-292

[9] Yuri Gurevich, "Evolving algebra 1993: Lipari guide," in *Specification and Validation Methods*, E. Börger (editor), Oxford Univ. Press (1995) 9–36.

[10] Yuri Gurevich, "Sequential abstract state machines capture sequential algorithms," *ACM Trans. Computational Logic* 1 (2000) 77–111.

[11] James K. Huggins, ASM Michigan web page, `http : //www.eecs.umich.edu/gasm`

[12] Stephen Cole Kleene, *Introduction to Metamathematics*, Van Nostrand, 1952.