General Interactive Small-Step Algorithms

Andreas Blass* Yuri Gurevich† Dean Rosenzweig‡
Benjamin Rossman§

Microsoft Research Technical Report MSR-TR-2005-113

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. Here we extend the axiomatization and the proof 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 the definition of abstract state machines to incorporate explicit attention to the relative timing of replies and to the possible absence of replies.

*Partially supported by NSF grant DMS–0070723 and by a grant from Microsoft Research. Address: Mathematics Department, University of Michigan, Ann Arbor, MI 48109–1043, U.S.A., ablass@umich.edu. Much of this paper was written during a visit to Microsoft Research.

†Microsoft Research, One Microsoft Way, Redmond, WA 98052, U.S.A. gurevich@microsoft.com

‡Microsoft Research; and University of Zagreb, FSB, I. Lucica 5, 10000 Zagreb, Croatia, dean@math.hr

§Microsoft Research; current address: Computer Science Dept., M.I.T., Cambridge, MA 02139, U.S.A., brossman@mit.edu

Contents

1 Introduction **2**

2 Postulates for Algorithms **4**
2.1 States and vocabularies 5
2.2 Histories and interaction 6
2.3 Completing a step 16
2.4 Isomorphism 20
2.5 Small steps 21

3 Equivalence of Algorithms **25**

4 Abstract State Machines — Syntax **29**
4.1 Vocabularies and templates 29
4.2 Guards 32
4.3 Rules 34
4.4 Syntactic sugar 35

5 Abstract State Machines — Semantics **37**
5.1 Terms 37
5.2 Guards 38
5.3 Rules 40

6 Algorithms are Equivalent to ASMs **50**
6.1 Vocabulary, labels, states 51
6.2 External vocabulary and templates 51
6.3 Critical elements, critical terms, agreement 52
6.4 Descriptions, similarity 57
6.5 The ASM program 67
6.6 Equivalence 68

1 Introduction

The Abstract State Machine (ASM) Thesis, first proposed in [6], asserts that every algorithm is equivalent, on its natural level of abstraction, to an abstract state machine. Beginning in [8] 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 present paper continues this tradition, but with an important difference. Previously, the standard syntax of ASMs, as presented in [7], was adequate, with only very minor modifications. Our present work, however, requires a significant extension of that syntax. The extension allows an ASM program to refer to the order in which the values of external functions are received from the environment, and it allows the program to declare a step complete even if not all external function values have been determined.

The main purpose of this paper is to extend the analysis of interactive, small-step algorithms, begun in [2, 3, 4], by removing the restriction to "ordinary" algorithms. We also extend the syntax and semantics of abstract state machines (ASMs) so that non-ordinary algorithms become expressible. The main contributions of this paper are

- postulates and definitions describing a general notion of interactive, small-step algorithm,
- syntax and semantics for ASMs incorporating interaction that need not be ordinary,
- verification that ASMs satisfy the postulates, and
- proof that every algorithm satisfying the postulates is equivalent, in a strong sense, to an ASM.

The algorithms considered in this paper proceed in discrete steps and do only a bounded amount of work in each step ("small-step") but can interact with their environments during a step, by issuing queries and receiving replies.

Such algorithms were analyzed in [2, 3, 4], subject to two additional restrictions, which we expressed by the word "ordinary." First, they never complete a step until they have received replies to all the queries issued during that step. Second, what the algorithm does at any step is completely determined by the algorithm, the current state, and the function that maps the current step's queries to the environment's answers. In other words, this answer function is the only information from the environment that the

algorithm uses. In particular, the order in which the environment's answers are received has no bearing on the computation.

In the present paper, we lift these restrictions. We allow an algorithm to complete a step even while some of its queries remain unanswered. We also allow the algorithm's actions to depend on the order in which answers were received from the environment.

It was shown in [4] that ordinary algorithms are equivalent to ASMs of the traditional sort, essentially as described in [7]. In order to similarly characterize the more general algorithms of the present paper, we extend the syntax and semantics of ASMs. In particular, we provide a way to refer to the timing of the evaluations of external functions and a way to terminate a step while some queries remain unanswered. See Subsection 2.2 for a discussion of the intuitive picture of interactive algorithms that leads to these particular extensions and indicates why they suffice.

2 Postulates for Algorithms

This section is devoted to the description of interactive, small-step algorithms by means of suitable definitions and postulates. Some parts of this material are essentially the same as in [8], which dealt with small-step algorithms that interact with the environment only between steps; some other parts are as in [2], which dealt with small-step algorithms that interact with the environment within steps but only in the "ordinary" manner described in the introduction. We shall present these parts again here, but without repeating the explanations and motivations from [8] and [2]. For the genuinely new material, dealing with the non-ordinary aspects of our algorithms' interaction with the environment, we shall present not only the definitions and postulates but also the reasons and intuitions that lie behind them.

Throughout the definitions and postulates that follow, we consider a fixed algorithm A. We may occasionally refer to it explicitly, for example to say that something depends only on A, but usually we leave it implicit.

Remark 2.1. The following postulates refer to the algorithm, yet it is only after the postulates that we define "algorithm" to mean an entity satisfying the postulates. The apparent circularity here can be avoided by starting with a definition of the form "An algorithm is a 9-tuple consisting of data as described in the following postulates — sets of states and of initial states; vocabulary; set of labels; sets of final, successful, and failing histories for all

states; and sets of updates for appropriate states and histories." We believe that the traditional mode of presentation, as in [8, 1, 2] is preferable, so we adopt it here, but any readers troubled by our using "algorithm" before defining it are welcome to imagine the 9-tuple definition inserted here. □

2.1 States and vocabularies

Our first postulate is exactly as in [2], and most of it is assembled from parts of postulates in [8]. We refer the reader to [8] for a careful discussion of the first three parts of the postulate and to [2] for the last part.

States Postulate: The algorithm determines

- a nonempty set $\mathcal{S}$ of *states*,
- a nonempty subset $\mathcal{I} \subseteq \mathcal{S}$ of *initial states*,
- a finite vocabulary Υ such that every $X \in \mathcal{S}$ is an Υ-structure, and
- a finite set Λ of *labels*.

As in the cited earlier papers, we adopt the following conventions concerning vocabularies and structures.

Convention 2.2.

- 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 $\{\texttt{true}, \texttt{false}\}$.

- The interpretation of `Boole` maps `true` and `false` to `true` and everything else to `false`.
- The interpretation of equality maps pairs of equal elements to `true` and all other pairs to `false`.
- The propositional connectives are interpreted in the usual way when their arguments are in $\{\texttt{true}, \texttt{false}\}$, and they take the value `false` whenever any argument is not in $\{\texttt{true}, \texttt{false}\}$.
- We may use the same notation X for a structure and its base set.

□

Definition 2.3. A *potential query* in state X is a finite tuple of elements of $X \sqcup \Lambda$. A *potential reply* in X is an element of X. □

Here $X \sqcup \Lambda$ means 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.

2.2 Histories and interaction

Definition 2.4. 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. □

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.

We use histories to express the information received by the algorithm from its environment during a step. The notion of answer function comes from [2], where these functions, telling which queries have received which replies, represented the whole influence of the environment on the algorithm's work. When algorithms are not required to be ordinary, then additional information from the environment, namely the relative order in which replies were received, becomes relevant to the computation. This information is represented by the pre-order part of a history. If $p, q \in \mathrm{Dom}(\dot{\xi})$ and $p <_\xi q$, this means that the answer $\dot{\xi}(p)$ to p was received strictly before the answer $\dot{\xi}(q)$ to q. If $p \equiv_\xi q$, this means that the two answers were received simultaneously.

The rest of this subsection is devoted to explaining in more detail the intuition behind this formalization of the intuitive notion of the history of an algorithm's interaction with its environment during a step. We do not, however, repeat here Sections 2 and 4 of [2]. The first of these two sections explains in detail our reasons for using queries and replies as our model of the interaction between an algorithm and its environment. The second explains the reasons for our specific definitions of (potential) queries and replies. Here, we presuppose these explanations and adopt the query-reply paradigm of [2]. Our task now is to explain what is added to the picture from [2] when we remove the restriction to ordinary algorithms.

Much of the information provided by an environment can and should be viewed as being part of its replies to queries. This includes not only the information explicitly requested by the query but also such varied information as "how hard did the user (who is part of the environment) bang on the keyboard when typing this input" or "at what time was this input provided" if such information is relevant to the algorithm's execution. Thus, we can

view such information as being included in the answer function $\dot{\xi}$, without referring to the second component of a history, the pre-order.

The purpose of the pre-order, on the other hand, is to represent the order in which the algorithm becomes aware of the environment's answers. Even if the environment provides a time-stamp as part of each reply, this order cannot be read off from the replies. The algorithm may become aware of the replies in an order different from that indicated by the time stamps. For a simple example, consider a broker who has a block of shares to sell. He asks two clients whether they want to buy the shares; both of them want to buy the whole block. The broker will sell the shares to the client whose message reaches him first, even if the message from the other client is sent earlier.

One can even imagine that a reply was provided by the environment before our algorithm even asked the query, perhaps because some other algorithm, running simultaneously with ours or earlier, issued the same query. For example, consider the following scenario. Our algorithm asks (query q_1) the environment for a certain piece of information. Instead of directly providing that information, the environment says "look at the answer that I sent to so-and-so last week." As a result, our algorithm looks (query q_2) into so-and-so's records and finds the desired information. If the environment had put a time-stamp on that information (and if the time-stamp hasn't been altered in the meantime), then our algorithm will find a time-stamp that is earlier than either of its queries. Of course, the situation would be different if the time-stamp were updated when the environment says to look at this previous answer. (This makes some sense, since, by telling us to look there, the environment is certifying or at least suggesting that the old answer is still correct.) Then our algorithm would find a time-stamp later than the time it issued q_1 but not later than the time it issued q_2. The time stamp could be later than q_2 if it were updated by our looking at the record, but that is contrary to the purpose of time-stamps.

Remark 2.5. Even though the pre-order $\leq_\xi$ is about the replies, the formal definition says that it pre-orders the domain of the answer function $\dot{\xi}$, i.e., the set of queries. The reason is a technical one: Different queries may receive the same reply, and in that case the single reply could occur at several positions in the ordering. Each query, on the other hand, occurs just once because, in accordance with the conventions of [2], we regard repetitions of a query as distinct queries (since they can receive different replies). Thus, the order in which replies are received can be conveniently represented by pre-ordering

the associated queries. It may be more intuitive to think of pre-ordering the set of pairs (query, reply), i.e., the graph of the answer function. This view of the situation would make no essential difference, since these pairs are in canonical bijection with the queries.

We emphasize that the timing we are concerned with here is logical time, not physical time. That is, it is measured by the progress of the computation, not by an external clock. If external, physical, clock time is relevant, as in real-time algorithms, it would have to be provided separately by the environment, for it is not part of the program or of the state. The relevant values of the physical time could be regarded as the replies to repeated queries asking "what time is it?"

In particular, we regard a query as being issued by the algorithm as soon as the information causing that query (in the sense of the Interaction Postulate below) is available. This viewpoint would be incorrect in terms of physical time, for the algorithm may be slow to issue a query even after it has the prerequisite information. But we can regard a query as being logically available as soon as its prerequisites are present.

This is why we include, in histories, only the relative ordering of replies. The ordering of queries relative to replies or relative to each other is then determined. The logical time of a query is the same as the logical time of the last of the replies that were prerequisites for that query.

Our use of pre-orders rather than partial orders, i.e., our allowing two distinct replies to be received simultaneously, also reflects our concern with logical rather than physical time. One can argue that no two events are exactly simultaneous in the physical sense (though they could be so nearly simultaneous that one should treat them as simultaneous), but logical simultaneity is certainly possible. It means merely that the two replies were available for exactly the same part of the computation.

The linearity of the pre-ordering, i.e., the requirement that every two replies be received either in one order or in the other or simultaneously, formalizes the following important part of our view of sequential-time (as opposed to distributed) algorithms. (It is not a postulate of our formal development but an informal principle that underlies some of our definitions and postulates.)

One Executor Principle A small-step algorithm is executed by a single, sequential entity. Even if there are (boundedly many) subprocesses running in parallel, they all operate under the control of a single, master executor.

The aspect of this principle that is relevant to the linearity of histories can be summarized in the following informal principle, in which the first part emphasizes the role of the master executor in the interaction with the environment, while the second part is a consequence of the sequentiality of the executor.

Holistic Principle The environment can send replies only to (the executor of) the algorithm, not directly to parts of it; similarly it receives queries only from the algorithm, not from parts of it. Any two replies are received by the algorithm either simultaneously or one after the other.

The pre-orders in our histories are intended to represent the order in which replies are received by the master executor. If there are subprocesses, then the master may pass the replies (or information derived from them) to these subprocesses at different times, but that timing is part of the algorithm's internal operation (possibly also influenced by the replies to other queries like "what time is it?"), not part of the environment interaction that histories are intended to model. The linearity of our pre-orders reflects the fact that they represent the ordering as seen by a single, sequential entity; this entity sees things in a sequence.

In more detail, we picture the execution of one step of an algorithm as follows. First, the algorithm (or, strictly speaking, its executor) computes as much as it can with only the information provided by the current state. This part of the computation, the first phase, will in general include issuing some queries. Then the algorithm pauses until it is "awakened" by the environment, which has replied to some (not necessarily all) of the queries from phase 1. The algorithm proceeds, in phase 2, to compute as much as it can using the state, the new information from the environment, and possibly some "scratch work" recorded during phase 1. Then it again pauses until the environment has provided some more replies (possibly to queries from phase 2 and possibly to previously unanswered queries from phase 1) and awakens the algorithm. Then phase 3 begins, and this pattern continues until the algorithm determines, in some phase, that it is ready to complete the current step, either by executing certain updates (computed during the various phases) of its state or by failing (in which case the whole computation fails and there is no next state).

The logical ordering of replies refers to the phases at which replies were received. That is, if $q_1 <_\xi q_2$, this means that the reply $\dot{\xi}(q_1)$ to q_1 was

received at an earlier phase than the reply $\dot{\xi}(q_2)$ to q_2. Similarly, $q_1 \equiv_\xi q_2$ means that these two replies were received at the same phase.

Remark 2.6. We have assumed, in the preceding description of phases, that the algorithm is awakened to begin a new phase only when some new reply has been provided. We should, however, discuss the possibility that the environment awakens the algorithm when no new replies are available. In such a case, it is natural to assume that the algorithm, having determined that it has no new information with which to advance the computation, simply resumes its pause until awakened again. In order for the algorithm to be a small-step algorithm, such fruitless awakenings must happen only a bounded number of times per step, with the bound depending only on the algorithm. The reason is that, even if all the algorithm does when awakened is to observe that no new replies have arrived, this observing is work, and a small-step algorithm can do only a bounded amount of it per step.

It seems possible, however, for an algorithm to admit a few fruitless awakenings per step, and the results of the computation could even depend on these awakenings. Consider, for example, an algorithm that works as follows. It begins by issuing a query q and pausing. When awakened, it outputs 1 and halts if there is a reply to q; otherwise, it pauses again. When awakened a second time, it outputs 2 if there is now a reply to q, and, whether or not there is a reply, it halts. Notice that, if q receives the reply r, then whether this is seen at the first or the second awakening doesn't affect the history, which consists of the function $\{\langle q, r\rangle\}$ and the unique pre-order on its domain $\{q\}$. So the history fails to capture all the information from the environment that is relevant to the computation.

There are two ways to correct this discrepancy. One, which we shall adopt, is to regard the fruitless awakening as amounting to a reply to an implicit query, of the form "I'm willing to respond to awakening." (For several fruitless awakenings, there would have to be several such queries, distinguished perhaps by numerical labels.) Now the two scenarios considered above, where q has received a reply at the first awakening or only at the second, are distinguished in the histories, because the reply to the new, implicit query will be simultaneous with the reply to q in the one scenario and will strictly precede the reply to q in the other scenario.

An alternative approach would be to avoid introducing such implicit queries but instead to replace the pre-order constituents of histories by slightly more complicated objects, "pre-orders with holes." The idea is that

the scenario where an answer to q is available only at the second awakening would be represented by the answer function $\{\langle q, r\rangle\}$ as above, but the pre-order would be replaced with something that says "first there's a hole and then q." Formalizing this is actually quite easy, as long as the pre-order is linear and the set D to be pre-ordered is finite (as it will be in the situations of interest to us). A linear pre-order of D with holes is just a function p from D into the natural numbers. For any n, the elements of $p^{-1}(\{n\})$ constitute the n^{th} equivalence class, which may be empty in case of a hole. Ordinary pre-orders correspond to the special case where the image $p(D)$ is an initial segment of $\mathbb{N}$.

Although the approach using pre-orders with holes corresponds more directly to intuition, we prefer the first approach, with implicit queries, for two reasons. First, it allows us to use the standard terminology of pre-orders rather than introducing something new. Second, its chief disadvantage, the need for implicit queries, is mitigated by the fact that we need other sorts of implicit queries for other purposes. For example, in [2, Section 2], implicit queries modeled the algorithm's receptiveness to incoming (unsolicited) messages and to multiple answers to the same query. Notice that, in all cases, our implicit queries represent the algorithm's willingness to pay attention to something provided by the environment. □

Definition 2.7. 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 $\restriction$ 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. □

Notice that any initial segment with respect to a pre-order $\leq$ is closed under the associated equivalence $\equiv$. Notice also that if $\langle \dot{\eta}, \leq_\eta \rangle \trianglelefteq \langle \dot{\xi}, \leq_\xi \rangle$ then $\leq_\eta$ is an initial segment of $\leq_\xi$. We also point out for future reference that, if two histories ξ_1 and ξ_2 are initial segments of the same ξ, then one of ξ_1 and ξ_2 is an initial segment of the other.

Intuitively, if $\langle \dot{\xi}, \leq_\xi \rangle$ is the history of an algorithm's interaction with the environment up to a certain phase, then a proper initial segment of this history describes the interaction up to some earlier phase.

Definition 2.8. If $\leq$ pre-orders the set D and if $q \in D$, then we define two associated initial segments as follows.

$$\begin{aligned}(\leq q) &= \{d \in D : d \leq q\} \\ (< q) &= \{d \in D : d < q\}.\end{aligned}$$

□

The following postulate is almost the same as the postulate of the same name in [2]. The only difference is that we use histories instead of answer functions. This reflects the fact that the decision to issue a query can be influenced by the timing of the replies to previous queries.

Interaction Postulate For each state X, the algorithm determines a binary relation $\vdash_X$, called the *causality relation*, between finite histories and potential queries.

The intended meaning of $\xi \vdash_X q$ is that, if the algorithm's current state is X and the history of its interaction so far (as seen by the algorithm during the current step) is ξ, then it will issue the query q unless it has already done so. When we say that the history so far is ξ, we mean not only that the environment has given the replies indicated in $\dot{\xi}$ in the order given by $\leq_\xi$, but also that no other queries have been answered. Thus, although ξ explicitly contains only positive information about the replies received so far, it also implicitly contains the negative information that there have been no other replies. Of course, if additional replies are received later, so that the new history has ξ as a proper initial segment, then q is still among the issued queries, because it was issued at the earlier time when the history was only ξ. This observation is formalized as follows.

Definition 2.9. For any state X and history ξ, we define sets of queries

$$\begin{aligned}\mathrm{Issued}_X(\xi) &= \{q : (\exists \eta \trianglelefteq \xi)\, \eta \vdash_X q\} \\ \mathrm{Pending}_X(\xi) &= \mathrm{Issued}_X(\xi) - \mathrm{Dom}(\dot{\xi}).\end{aligned}$$

□

Thus, $\mathrm{Issued}_X(\xi)$ is the set of queries that have been issued by the algorithm, in state X, by the time the history is ξ, and $\mathrm{Pending}_X(\xi)$ is the subset of those that have, as yet, no replies.

Remark 2.10. We have described the causality relation in terms of detailed causes, histories ξ that contain the algorithm's whole interaction with environment up to the time the caused query is issued. A text describing the algorithm, either informally or in some programming language, would be more likely to describe causes in terms of partial information about the history. For example, it might say "if the query p has received the reply a but p' has no reply yet, then issue q." This description would correspond to a large (possibly infinite) set of instances $\xi \vdash q$ of the causality relation, namely one instance for every history ξ that fits the description, i.e., such that $\dot{\xi}(p) = a$ and $p' \notin \mathrm{Dom}(\dot{\xi})$. More generally, whenever we are given a description of the conditions under which various queries are to be issued, we can similarly convert it into a causality relation; each of the conditions would be replaced by the set of finite histories that satisfy the condition.

The reverse transformation, from detailed causes representing the whole history to finite descriptions, is not possible in general. The difficulty is that the finite descriptions might have to specify all of the negative information implicit in a history, and this might be an infinite amount of information. In most situations, however, the set of queries that might be issued is finite and known. Then any finite history ξ can be converted into a finite description, simply by adding, to the positive information explicit in ξ, the negative information that there have been no replies to any other queries among the finitely many that could be issued.

For semantical purposes, as in the present section, it seems natural to think of causes as being the detailed histories. From the behavior of an algorithm, one can easily determine whether $\xi \vdash_X q$ (at least when ξ is attainable as in Definition 2.16 below and we ignore causes of q that have proper initial segments already causing q); just provide the algorithm with replies according to ξ and see whether it produces q at that moment (and not earlier). It is not easy to determine, on the basis of the behavior, whether q is issued whenever a description, like "p has reply a and p' has no reply" is satisfied, as this involves the behavior in a large, possibly infinite number of situations.

For syntactic purposes, on the other hand, descriptions seem more natural than detailed histories. And indeed, the syntax of our ASMs (in Section 4) will not involve histories directly but will involve descriptions like "p has reply a and p' has no reply" in the guards of conditional rules. □

Notice that there is no guarantee that $\mathrm{Dom}(\dot{\xi}) \subseteq \mathrm{Issued}_X(\xi)$, although

this will be the case for attainable histories (defined below). In general, a history as defined above may contain answers for queries that it and its initial segments don't cause. It may also contain answers for queries that would be issued but only at a later phase than where the answers appear. In this sense, histories need not be possible records of actual interactions, between the algorithm and its environment, under the causality relation that is given by the algorithm. The following definition describes the histories that are consistent with the given causality relation. Informally, these are the histories where every query in the domain has a legitimate reason, under the causality relation, for being there.

Definition 2.11. A history ξ is *coherent*, with respect to a state X or its associated causality relation $\vdash_X$, if

- $(\forall q \in \mathrm{Dom}(\dot{\xi}))\, q \in \mathrm{Issued}_X(\xi \restriction (< q))$, and
- the linear order of the $\equiv_\xi$-equivalence classes induced by $\leq_\xi$ is a well-order.

□

The first requirement in this definition, which can be equivalently rewritten as

$$(\forall q \in \mathrm{Dom}(\dot{\xi}))(\exists \eta \trianglelefteq \xi)\, (\eta \vdash_X q \text{ and } q \notin \mathrm{Dom}(\dot{\eta})),$$

says that, if a query q receives an answer at some phase, then it must have been issued on the basis of what happened in strictly earlier phases. In particular, the queries answered in the first phase, i.e., those in the first $\equiv_\xi$-class, must be caused by the empty history.

The second requirement is needed only because we allow infinite histories; finite linear orders are automatically well-orders. The purpose of the second requirement is to support the usual understanding of "cause" by prohibiting an infinite regress of causes.

Remark 2.12. It will follow from the Bounded Work Postulate below in conjunction with the Interaction Postulate above, that we can confine attention to histories whose domains are finite. Then the second bullet item in the definition of coherence would be automatically satisfied. Nevertheless, we allow a coherent history ξ to have infinite domain and even to have infinitely many equivalence classes, with respect to $\leq_\xi$, provided their ordering

induced by $\leq_\xi$ is a well-ordering. There are two reasons for this generality. First, it will allow us to state the Bounded Work Postulate in a relatively weak form and then to deduce the stronger fact that the histories we really need to consider (the attainable ones) are finite. Second, it avoids formally combining issues that are conceptually separate. Finiteness is, of course, an essential aspect of computation, especially of small-step computation, and the Bounded Work Postulate will formalize this aspect. But the notions of coherent and attainable histories are conceptually independent of finiteness, and so we do not impose finiteness in their definitions. □

Remark 2.13. As mentioned above, the notion of coherence is intended to capture the idea of a history that makes sense, given the algorithm's causality relation. Here we implicitly use the fact that we are describing an entire algorithm, not a component that works with other components to constitute an algorithm. If we were dealing with a component C, working in the presence of other components, then it would be possible for queries to be answered without having been asked by C, simply because another component could have asked the query. See [5] for a study of components. □

It follows immediately from the definitions that any initial segment of a coherent history is coherent.

Definition 2.14. A history ξ for a state X is *complete* if $\mathrm{Pending}_X(\xi) = \varnothing$. □

Thus, completeness means that all queries that have been issued have also been answered. It can be regarded as a sort of converse to coherence, since the latter means that all queries that have been answered have also been issued earlier.

If a complete history has arisen in the course of a computation, then there will be no further interaction with the environment during this step. No further interaction can originate with the environment, because no queries remain to be answered. No further interaction can originate with the algorithm, since ξ and its initial segments don't cause any further queries. So the algorithm must either proceed to the next step (by updating its state) or fail.

2.3 Completing a step

At the end of a step the algorithm will, unless it fails, perform some set of updates. The next postulate will express this property of algorithms formally.

First, we adopt the formalization of the notion of update that was used in earlier work, starting with [8].

Definition 2.15. A *location* in a state X is a pair $\langle f, \mathbf{a}\rangle$ where f is a dynamic function symbol from Υ and $\mathbf{a}$ is a tuple of elements of X, of the right length to serve as an argument for the function f_X interpreting the symbol f in the state X. The *value* of this location in X is $f_X(\mathbf{a})$. An *update* for X is a pair (l, b) consisting of a location l and an element b of X. An update (l, b) is *trivial* (in X) if b is the value of l in X. We often omit parentheses and brackets, writing locations as $\langle f, a_1, \ldots, a_n\rangle$ instead of $\langle f, \langle a_1, \ldots, a_n\rangle\rangle$ and writing updates as $\langle f, \mathbf{a}, b\rangle$ or $\langle f, a_1, \ldots, a_n, b\rangle$ instead of $(\langle f, \mathbf{a}\rangle, b)$ or $(\langle f, \langle a_1, \ldots, a_n\rangle\rangle, b)$. □

The intended meaning of an update $\langle f, \mathbf{a}, b\rangle$ is that the interpretation of f is to be changed (if necessary, i.e., if the update is not trivial) so that its value at $\mathbf{a}$ is b.

Because our algorithms are not required to be ordinary, they may finish a step without reaching a complete history. The following postulate says that they must finish the step when they reach a complete history, if they have not already done so earlier.

Step Postulate — Part A The algorithm determines, for each state X, a set $\mathcal{F}_X$ of *final histories*. Every complete, coherent history has an initial segment (possibly the whole history) in $\mathcal{F}_X$.

Intuitively, a history is final for X if, whenever it arises in the course of a computation in X, the algorithm completes its step, either by failing or by executing its updates and proceeding to the next step.

Since incoherent histories cannot arise under the algorithm's causality relation, it will not matter which such histories are final. We could have postulated that only coherent histories can be final, but there is no need to do so. It does no harm to allow incoherent final histories, though they are irrelevant to the algorithm's computation. By allowing them, we may slightly simplify the description, as algorithms subject to our postulates, of programs written in some programming language; if the language allows one to say "finish the step" then this can be directly translated into final histories, without having to check for coherence. Similar comments apply to the remaining parts of the Step Postulate below, where we require update sets or failure for coherent histories but allow them also for incoherent ones.

Definition 2.16. A history for a state X is *attainable* (in X) if it is coherent and no proper initial segment of it is final. □

Since initial segments of coherent histories are coherent, it follows that initial segments of attainable histories are attainable.

The attainable histories are those that can occur under the given causality relation and the given choice of final histories. That is, not only are the queries answered in an order consistent with $\vdash_X$ (coherence), but the history does not continue beyond where $\mathcal{F}_X$ says it should stop.

Our earlier statements to the effect that incoherent histories don't matter can be strengthened to say that unattainable histories don't matter. Thus, for example, although we allow one final history to be a proper initial segment of another (even when both are coherent), the longer of the two will be unattainable, so its being final is irrelevant.

Step Postulate — Part B For each state X, the algorithm determines that certain histories *succeed* and others *fail*. Every final, attainable history either succeeds or fails but not both.

Non-final or unattainable histories may succeed or fail or both, but this will be irrelevant to the behavior of the algorithm. The intended meaning of "succeed" and "fail" is that a successful final history is one in which the algorithm finishes its step and performs a set of updates of its state, while a failing final history is one in which the algorithm cannot continue — the step ends, but there is no next state, not even a repetition of the current state. Such a situation can arise if the algorithm computes inconsistent updates. It can also arise if the environment gives inappropriate answers to some queries.

For more about failures, see the discussion following the Update Postulate in [2, Section 5]. There is, however, one difference between our present situation and that in [2]. There, the algorithm depended upon getting answers to all its queries, so it is reasonable to expect that an inappropriate answer always results in failure. In our present situation, however, the algorithm may be willing to finish its step without getting the answer to a certain query. In this case, an inappropriate answer to that query need not force failure.

Definition 2.17. The set of successful final histories is denoted by $\mathcal{F}_X^+$, and the set of failing final histories is denoted by $\mathcal{F}_X^-$. □

Thus, $\mathcal{F}_X$ is the disjoint union of $\mathcal{F}_X^+$ and $\mathcal{F}_X^-$.

When a history in $\mathcal{F}_X^+$ arises during the computation, the algorithm should perform its updates and proceed to the next step. We formalize this in the third and last part of the Step Postulate, keeping in mind that only attainable histories can arise. The notation Δ^+ for the update set is taken from [2]. The superscript + refers to the fact that trivial updates can be included in the update set. Although they do not affect the next state, they can affect whether clashes occur when our algorithm is run in parallel with another.

Step Postulate — Part C For each attainable history $\xi \in \mathcal{F}_X^+$ for a state X, the algorithm determines an *update set* $\Delta^+(X, \xi)$, whose elements are updates for X. It also produces a *next state* $\tau(X, \xi)$, which

- has the same base set as X,
- has $f_{\tau(X,\xi)}(\mathbf{a}) = b$ if $\langle f, \mathbf{a}, b\rangle \in \Delta^+(X, \xi)$, and
- otherwise interprets function symbols as in X.

The Step Postulate is the analog, for our present situation, of the Update Postulate of [2]. There are two differences between these postulates, reflecting the two requirements of ordinariness in [2] that have been lifted in the present work.

First, since we now consider algorithms that can finish a step without waiting for all queries to be answered, we include in the notion of algorithm the information about when to finish a step. This is the notion of final history introduced in Part A of the Step Postulate. If an algorithm never finishes a step while queries are still pending, then its final histories could be defined as being the complete histories (or the complete, coherent histories), and there would be no need to specify $\mathcal{F}$ separately in the algorithm. The complete, coherent histories correspond to the contexts, as defined in [2].

Second, because we allow an algorithm to take into account the order in which it receives replies to its queries, decisions about failures and updates depend not only on the answer function $\dot{\xi}$ but on the whole history ξ, including the ordering information contained in $\leq_\xi$.

The intuition and motivation behind the other aspects of the Step Postulate are discussed in detail in [8] and [2].

Convention 2.18. In notations like $\mathcal{F}_X$, $\mathcal{F}_X^{\pm}$, $\Delta^+(X,\xi)$, and $\tau(X,\xi)$, we may omit X if only one X is under discussion. We may also add the algorithm A as a superscript if several algorithms are under discussion. □

Notice that the Step Postulate requires $\Delta^+(X,\xi)$ and $\tau(X,\xi)$ to be defined when ξ is an attainable, final, successful history, but it allows them to be defined also for other histories. Any values they may have for other histories, however, will not affect the algorithm's computation.

Notice also that the next state is completely determined by the current state and the update set. So when describing an algorithm, we need not explicitly describe τ if we have described Δ^+.

If $\Delta^+(X,\xi)$ clashes, i.e., if it contains two distinct updates of the same location, then the description of $\tau(X,\xi)$ is contradictory, so the next state cannot exist. Thus, if such a ξ is attainable and final, it must be failing. That is, clashes imply failure.

Remark 2.19. Although the primary subject of this paper is intra-step interaction between an algorithm and its environment, it is not our intention to prohibit inter-step interaction of the sort described in [7]. Inter-step interaction is, however, quite easy to describe; the environment is permitted to make arbitrary changes to the state's dynamic functions between the steps of the algorithm. Thus, a *run* of an algorithm is a (finite or infinite) sequence of states, starting with an initial state, in which each state but the first is obtained from its predecessor either by an intervention of the environment or by a step of the algorithm. In the intervention case, the successor state has the same base set and static functions as the predecessor, but the dynamic functions can be arbitrarily altered. In the case of an algorithm step, there is a successful final history ξ, describing the environment's replies to the algorithm's queries during the step, and if the state before the step was X then the state after the step is $\tau(X,\xi)$. Since what happens in the intervention steps is quite arbitrary, our concern in this paper is to analyze what happens in the algorithmic steps. □

2.4 Isomorphism

As in previous work, starting with [8], we require that the information relevant to the algorithm's computation that is given by the state must be explicitly given by the structure of the state, not implicitly given by the par-

ticular nature of its elements. Formally, this means that the computation must be invariant under isomorphisms.

Any isomorphism $i : X \cong Y$ between states can be extended in an obvious, canonical way to act on queries, answer functions, histories, locations, updates, etc. We use the same symbol i for all these extensions.

Isomorphism Postulate Suppose X is a state and $i : X \cong Y$ is an isomorphism of Υ-structures. Then:

- Y is a state, initial if X is.
- i preserves causality, that is, if $\xi \vdash_X q$ then $i(\xi) \vdash_Y i(q)$.
- i preserves finality, success, and failure, that is, $i(\mathcal{F}_X^{\pm}) = \mathcal{F}_Y^{\pm}$ and $i(\mathcal{F}_X) = \mathcal{F}_Y$.
- i preserves updates, that is, $i(\Delta^+(X, \xi)) = \Delta^+(Y, i(\xi))$ for all histories ξ for X.

Convention 2.20. In the last part of this postulate, and throughout this paper, we adopt the convention that an equation between possibly undefined expressions is to be understood as implying that if either side is defined then so is the other. □

Remark 2.21. We have required that isomorphisms preserve even the irrelevant parts of the algorithm, like update sets for unattainable or non-final histories. This requirement could be dropped without any damage to our results. Nevertheless, it seems a natural requirement in a general description of algorithms. The intuition behind it is that, even if an algorithm includes irrelevant information, the states should still be abstract; not even the irrelevant information should be able to "see" the particular nature of the elements of the state. □

2.5 Small steps

The final postulate formalizes the requirement that a small-step algorithm can do only a bounded amount of work in any one step. The bound depends only on the algorithm. Work includes assembling queries (from elements of

the state and labels) and issuing them, computing what queries to issue, deciding whether the current history suffices to finish the step, deciding whether the computation has failed, and computing updates. Our formalization of this closely follows the corresponding postulate in [2, Section 5]; the use of a set of terms to represent the bounded part of the state that the algorithm looks at goes back to [8].

Bounded Work Postulate

- There is a bound, depending only on the algorithm, for the lengths of the tuples in $\mathrm{Issued}_X(\xi)$, for all states X and final, attainable histories ξ.
- There is a bound, depending only on the algorithm, for the cardinality $|\mathrm{Issued}_X(\xi)|$, for all states X and final, attainable histories ξ.
- There is a finite set W of Υ-terms (possibly involving variables), depending only on the algorithm, with the following property. Suppose X and X' are two states and ξ is a history for both of them. Suppose further that each term in W has the same value in X as in X' when the variables are given the same values in $\mathrm{Range}(\dot{\xi})$. Then:
  - If $\xi \vdash_X q$ then $\xi \vdash_{X'} q$ (so in particular q is a query for X').
  - If ξ is in $\mathcal{F}^+_X$ or $\mathcal{F}^-_X$, then it is also in $\mathcal{F}^+_{X'}$ or $\mathcal{F}^-_{X'}$, respectively.
  - $\Delta^+(X, \xi) = \Delta^+(X', \xi)$.

This completes our list of postulates, so we are ready to define the class of algorithms to be treated in this paper.

Definition 2.22. An *interactive, small-step algorithm* is any entity satisfying the States, Interaction, Step, Isomorphism, and Bounded Work Postulates. □

Since these are the only algorithms under consideration in most of this paper, we often omit "interactive, small-step"; on the other hand, when we want (as in the title of this paper) to emphasize the difference between these algorithms and the "ordinary algorithms" treated in [2, 3, 4], we may refer to the present class of algorithms as "general, interactive, small-step algorithms".

The remainder of this section is devoted to some terminology and results connected with the Bounded Work Postulate.

Definition 2.23. A set W with the property required in the third part of the Bounded Work Postulate is called a *bounded exploration witness* for the algorithm. Two pairs (X, ξ) and (X', ξ), consisting of states X and X' and a single ξ that is a history for both, are said to *agree* on W if, as in the postulate, each term in W has the same value in X as in X' when the variables are given the same values in $\mathrm{Range}(\dot{\xi})$. □

The first two parts of the Bounded Work Postulate assert bounds for final, attainable histories. They imply the corresponding bounds for all attainable histories, thanks to the following lemma.

Lemma 2.24. *Let X be a state.*

- *Any coherent history for X is an initial segment of a complete, coherent history for X.*
- *Any attainable history for X is an initial segment of a final, attainable history for X.*

Proof. Since X is fixed throughout the proof, we omit it from the notation.

To prove the first assertion, let ξ be a coherent history. According to the definition, its order-type (meaning, strictly speaking, the order-type of the ordering induced by $\leq_\xi$ on the equivalence classes under $\equiv_\xi$) is some ordinal number α. We shall inductively define a sequence of coherent histories ξ_n, starting with $\xi_0 = \xi$. Here n will range over either the set $\mathbb{N}$ of all natural numbers or the set of natural numbers up to some finite N to be determined during the construction. After ξ_n is defined, if it is complete, then stop the construction, i.e., set $N = n$. If ξ_n is incomplete, this means that the set $D_n = \mathrm{Pending}(\xi_n) = \mathrm{Issued}(\xi_n) - \mathrm{Dom}(\dot{\xi}_n)$ is nonempty. Extend the answer function $\dot{\xi}_n$ by adjoining D_n to its domain and assigning it arbitrary values (in X) there. Call the resulting answer function $\dot{\xi}_{n+1}$, and make it into a history ξ_{n+1} by pre-ordering its domain as follows. On $\mathrm{Dom}(\dot{\xi}_n)$, $\leq_{\xi_{n+1}}$ agrees with $\leq_{\xi_n}$. All elements of $\mathrm{Dom}(\dot{\xi}_{n+1})$ are $\leq_{\xi_{n+1}}$ all elements of D_n. Elements of D_n are $\leq_{\xi_{n+1}}$ only each other. In other words, we extend the ordering of $\mathrm{Dom}(\dot{\xi}_n)$ by adding D_n at the end, as a single equivalence class. It is straightforward to check that each ξ_n (if defined, i.e., if the sequence hasn't ended before n) is a coherent history; the order-type of its equivalence classes is $\alpha + n$.

If ξ_n is defined for only finitely many n, then this is because the last ξ_n was complete, and so we have the required result. It remains to consider the case where ξ_n is defined for all natural numbers n. In this case, let ζ be the union of all the ξ_n's. (More formally, if we regard functions and orderings as sets of ordered pairs, then $\dot{\zeta}$ is the union of the $\dot{\xi}_n$'s, and the pre-order $\leq_\zeta$ is the union of the $\leq_{\xi_n}$'s.) Then ζ is also coherent. Indeed, if $q \in \mathrm{Dom}(\dot{\zeta})$ then there is some n such that $q \in \mathrm{Dom}(\dot{\xi}_n)$. As ξ_n is coherent, $q \in \mathrm{Issued}(\xi_n \restriction (< q))$. But, since each ξ_n is an initial segment of the next, and therefore of ζ, we have $\xi_n \restriction (< q) = \zeta \restriction (< q)$ and so $q \in \mathrm{Issued}(\zeta \restriction (< q))$. Furthermore, the order-type of ζ is $\alpha + \omega$ (where ω is the order-type of the natural numbers), so it is well-ordered.

To finish the proof of the first part of the lemma, we need only check that ζ is complete. Suppose, toward a contradiction, that $q \in \mathrm{Pending}(\zeta)$. So there is an initial segment η of ζ such that $\eta \vdash q$. By the Interaction Postulate, η is finite, so it is an initial segment of ξ_n for some n. Then

$$q \in \mathrm{Issued}(\xi_n) \subseteq \mathrm{Dom}(\dot{\xi}_{n+1}) \subseteq \mathrm{Dom}(\dot{\zeta}).$$

That contradicts the assumption that $q \in \mathrm{Pending}(\zeta)$, so we have shown that ζ is complete. Thus, $\xi = \xi_0$ is an initial segment of the complete, coherent history ζ, and the first assertion of the lemma is proved.

To prove the second assertion, let ξ be an attainable history. In particular, it is coherent, so, by the first assertion, it is an initial segment of a complete, coherent history ζ. By Part A of the Step Postulate, ζ has an initial segment η that is a final history. If ζ has several initial segments that are final histories, then let η be the shortest of them; thus no proper initial segment of η is final. Since both ξ and η are initial segments of ζ, one of them is an initial segment of the other. Since ξ is attainable and η is final, η cannot be a proper initial segment of ξ. Therefore, ξ is an initial segment of η. Furthermore, η is coherent, because it is an initial segment of the coherent history ζ. It follows, since no proper initial segment of η is final, that η is attainable, as desired. □

The following corollary extends the first assertion in the Bounded Work Postulate to histories that need not be final.

Corollary 2.25. *There is a bound, depending only on the algorithm, for the lengths of the tuples in $\mathrm{Issued}_X(\xi)$ for all states X and all attainable histories ξ.*

Proof. The bound on lengths of queries issued by final, attainable histories, given by the Bounded Work Postulate, applies to all attainable histories, because these are, by the lemma, initial segments of final ones. □

The next corollary similarly extends the second assertion of the Bounded Work Postulate, and adds some related information.

Corollary 2.26. *There is a bound, depending only on the algorithm, for* $|Issued_X(\xi)|$*, for all states* X *and attainable histories* ξ*. The same number also bounds* $|Dom(\dot{\xi})|$ *for all attainable* ξ*.*

Proof. By the lemma, any attainable history ξ is an initial segment of a final, attainable history ζ. Then $\mathrm{Issued}_X(\xi) \subseteq \mathrm{Issued}_X(\zeta)$. The bound provided by the Bounded Work Postulate for $|\mathrm{Issued}_X(\zeta)|$ thus applies to ξ as well. This proves the first assertion of the corollary, and the second follows because ξ, being attainable, is coherent, which implies $\mathrm{Dom}(\dot{\xi}) \subseteq \mathrm{Issued}_X(\xi)$. □

3 Equivalence of Algorithms

One of this paper's principal aims is to show that every algorithm, in the sense defined above, is behaviorally equivalent, in a strong sense, to an ASM. Of course, this goal presupposes a precise definition of the notion of behavioral equivalence, and we devote the present section to presenting and justifying that definition. As in earlier work on the ASM thesis, beginning in [8] and continuing in [1, 2, 3, 4], the definition of equivalence is intended to express the idea that two algorithms behave the same way in all possible situations. We must, of course, make precise what is meant by "behave the same way" and by "possible situations." Much of what needs to be said here was already said in [2, Section 6] in the more restricted context of ordinary interaction, and we shall not repeat all the details here.

Part of the definition of equivalence of algorithms is straightforward. As in previous work, we require equivalent algorithms to have the same states, the same initial states, the same vocabulary, and the same labels. The requirement that they agree as to states and initial states is clearly necessary for any comparison at all between their behaviors, specifically the aspect of behavior given by the progression of states in a run. The requirement that they agree as to vocabulary is actually a consequence of agreement as to states (and the requirement, in the definition of algorithm, that there be

at least one state), because any structure determines its vocabulary. The requirement that they agree as to labels ensures that the algorithms have, in any state, the same potential queries; this is needed for any comparison between their behaviors, since issuing queries is observable behavior.

The requirements just discussed say that equivalent algorithms agree as to all the items introduced in the States Postulate. It is tempting to go through the remaining postulates, look for statements of the form "the algorithm determines" such-and-such, and require equivalent algorithms to have the same such-and-such. Unfortunately, this approach, which in [8] would produce the correct notion of equivalence, is too restrictive when applied to interactive algorithms in [2] and the present paper.

The difficulties were already pointed out in [2, Section 6] in connection with the causality relation. The examples given there exhibit the following two sorts of problems. First, a causality relation could have instances $\xi \vdash_X q$ whose ξ could never actually occur in the execution of the algorithm, for example because $\mathrm{Dom}(\dot{\xi})$ contains queries that the algorithm would never issue. Second, as in [3, Example 6.4], the ξ in an instance $\xi \vdash_X q$ of causality could contain redundant elements, such as a query-reply pair that would have to be present in order for another query in $\mathrm{Dom}(\dot{\xi})$ to be issued. Algorithms whose causality relations differ only in such irrelevant ways should count as equivalent. That is, we should care only about what queries the algorithm issues in response to histories that can actually occur when this algorithm runs.

Similar comments apply to the remaining ingredients of an algorithm, in which histories are used. The notions of final, successful, and failing histories and the update sets should not be required to agree completely when two algorithms are equivalent; it suffices that they agree on those histories that can actually occur.

Remark 3.1. We emphasize again that we are dealing here with a situation where only the algorithm and its environment are involved. In other situations, for example when several algorithms interact (and we do not choose to consider each as a part of the environment for the others), the situation would be more complex. Consider, for example, two algorithms that function as components in a larger computation. The first of these components might issue a query whose reply is used by the second. In that case, instances $\xi \vdash_X q$ of the second component's causality relation can be relevant even if that component would never issue the queries in $\mathrm{Dom}(\dot{\xi})$. Some aspects

of this situation will arise when we discuss the "do in parallel" construct of ASMs; a thorough discussion will be provided in [5]. □

To formalize the preceding discussion, we must still say something about the notion of a history that can actually appear. We have already introduced a precise version of this notion, namely the notion of an attainable history. This notion, however, depends on the causality relation and the notion of finality given with the algorithm. When we define equivalence of two algorithms, which one's attainability should we use? There are two natural choices: Require the algorithms to agree (as to queries issued, finality, success, failure, and updates) on those histories that are attainable for both algorithms, or require agreement on all histories that are attainable under at least one of the algorithms. (There are also some less natural choices, for example to require agreement on the histories attainable under the first of the two algorithms; this threatens to make equivalence unsymmetric.) Fortunately, these options lead, as we shall prove below, to the same notion of equivalence of algorithms. Having advertised our notion of equivalence as a strong one, we take the apparently stronger of the natural definitions as the official one and then prove its equivalence with the apparently weaker one.

Definition 3.2. Two algorithms are *behaviorally equivalent* if

- they have the same states (therefore the same vocabulary), the same initial states, and the same labels,
- in each state, they have the same attainable histories,
- for each state X and each attainable history ξ, they have the same set $\mathrm{Issued}_X(\xi)$,
- for each state, the two algorithms have the same attainable histories in $\mathcal{F}_X^+$ and $\mathcal{F}_X^-$ (and therefore also in $\mathcal{F}_X$).
- for each state and each attainable, final, successful history, they have the same update sets.

□

The following lemma shows that the notion of equivalence is unchanged if we delete the second item in the definition and weaken the subsequent ones to apply only to histories that are attainable for both algorithms.

Lemma 3.3. *Suppose two algorithms have the following properties.*

- *They have the same states (therefore the same vocabulary), the same initial states, and the same labels,*
- *for each state X and each history ξ that is attainable for both algorithms, they have the same set $Issued_X(\xi)$,*
- *for each state, any history that is attainable in both algorithms and is in $\mathcal{F}_X^+$ or $\mathcal{F}_X^-$ for one of the algorithms is also in $\mathcal{F}_X^+$ or $\mathcal{F}_X^-$, respectively, for the other.*
- *for each state and each history that is attainable for both and final and successful (for one and therefore both), they have the same update sets.*

Then these two algorithms are equivalent.

Proof. Comparing the hypotheses of the lemma with the definition of equivalence, we see that it suffices to prove that, under the hypotheses of the lemma, any history that is attainable in a state X for either algorithm must also be attainable in X for the other. Indeed, this would directly establish the second clause of the definition, and it would make the subsequent clauses in the definition equivalent to the corresponding hypotheses in the lemma.

Let A_1 and A_2 be two algorithms satisfying the hypotheses of the lemma. Fix a state X for the rest of the proof; we shall suppress explicit mention of X in our notations. To prove that A_1 and A_2 have the same attainable histories, we first recall that any attainable history (for either algorithm) has finite domain by Corollary 2.26 and therefore has finite length. We can therefore proceed by induction on the length of histories. Since the empty history vacuously satisfies the definitions of "coherent" and "attainable", we need only verify the induction step.

Consider, therefore, a nonempty history ξ that is attainable for A_1. Let the last equivalence class in $\mathrm{Dom}(\dot{\xi})$ be L and let η be the initial segment of ξ obtained by deleting L from the domain. Then η is attainable for A_1 (because ξ is) and therefore also for A_2 (by induction hypothesis). Furthermore, the linear order of the equivalence classes of ξ, being finite, is certainly a well-ordering. So to complete the proof that ξ is attainable for A_2, it suffices to verify that it satisfies, with respect to A_2, the first clause in the definition of coherence (that every $q \in \mathrm{Dom}(\dot{\xi})$ is caused by an initial segment that ends before q) and that no proper initial segment of it is final.

For the first of these goals, observe that the induction hypothesis gives what we need if $q \in \mathrm{Dom}(\dot{\eta})$, so we need only deal with the case that $q \in L$. In this case, we know that $q \in \mathrm{Issued}^{A_1}(\eta)$ because ξ is coherent for A_1. But since η is attainable for both algorithms, the second hypothesis of the lemma applies, and we infer that $q \in \mathrm{Issued}^{A_2}(\eta)$, as required.

For the second goal, we already know that η is attainable for A_2 and so none of its proper initial segments can be final. The only proper initial segment of ξ not covered by this observation is η, so it remains only to show that η is not final for A_2. Since η is not final for A_1 (because ξ is attainable for A_1) and since η is attainable for both algorithms, the third hypothesis of the lemma immediately gives the required conclusion. □

4 Abstract State Machines — Syntax

In this section, we describe the syntactic side of ASMs. The most important part of this is the syntax of ASM programs, but it also includes the template assignments, whose role is to mediate between the notation used in programs and the queries issued when the ASM runs. The ASMs to be described here are an extension of those 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 recapitulate here, for the sake of completeness, the essential definitions from [3], but we do not repeat the detailed discussion and motivation for these definitions. Of course we do provide detailed discussion and motivation for those aspects of the present material that go beyond what was in [3].

4.1 Vocabularies and templates

An ASM has, like any algorithm, a finite vocabulary Υ complying with Convention 2.2. In addition, it 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 $\mathbf{a}$ is an n-tuple of arguments from a state X, then the value of f at $\mathbf{a}$ is not stored in the state but is obtained from the environment as the reply to a query.

[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.

Remark 4.1. The ASM syntax of [3] included commands of the form $\texttt{Output}_l(t)$ 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 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 4.2. Recall from Convention 2.2 that 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 4.3. 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.  □
```

Definition 4.4. 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 Section 2. 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 4.4 below), and so we do not need variables in our basic formalism.

Definition 4.5. A *Boolean term* is a term of the form $f(\mathbf{t})$ where f is a relational symbol. □

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 4.6. 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. □

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 4.7. 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, □

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 $f(t_1, \ldots, t_n)$.

4.2 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 a history ξ containing all the replies needed for both s and t, there will be a truth value, determined by ξ, for $(s \preceq t)$. At the other extreme, if neither s nor t can be fully evaluated under ξ, 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 is the situation where the history ξ suffices for the evaluation of one but not both of s and t. If it suffices for s but not for t, then $(s \preceq t)$ is true; if it suffices 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. 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 histories lack the replies needed for a complete evaluation, and it seems superfluous to introduce another entity, "unknown," to serve as a marker of this situation.)

Definition 4.8. 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 $\neg\varphi$.

□

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.

4.3 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 4.9. 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 $f \in \Upsilon$ 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.

4.4 Syntactic sugar

The preceding subsections complete 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 4.10. The parallel combination with no components, officially written `do in parallel enddo`, is abbreviated `skip`. □

Convention 4.11. The parallel combination with $k \geq 2$ components $R_1, \ldots, R_k$ can be written as R_1 `par` ... `par` R_k. □

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 4.12. We abbreviate `if` φ `then` R `else skip endif` as `if` φ `then` R `endif`. □

Convention 4.13. For any term t, the Boolean term $t = t$ is denoted by $t!$, read as "t bang." □

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 4.14. We use the following abbreviations:

$$\begin{array}{lll} (s \prec t) & \text{for} & \neg(t \preceq s), \\ (s \approx t) & \text{for} & (s \preceq t) \curlywedge (t \preceq s), \\ (s \succeq t) & \text{for} & (t \preceq s), \text{and} \\ (s \succ t) & \text{for} & (t \prec s) \end{array}$$

Parentheses may be omitted when no confusion results. □

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 4.15. 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. □

Convention 4.16. 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 $R(t_1, \ldots, t_k)$. □

Convention 4.17. 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

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

□

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 this `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.

5 Abstract State Machines — Semantics

Throughout this section, we refer to a fixed structure X. We define the semantics of terms, guards, and rules in the structure X, relative to histories ξ. (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.

5.1 Terms

The semantics of terms is 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 $\mathrm{Val}(t, X, \xi)$.

Definition 5.1. 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, 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, $\mathrm{Val}(t_i, X, \xi) = a_i$, if $f \in E$, and if $\hat{f}[a_1, \ldots, a_n] \in \mathrm{Dom}(\dot{\xi})$, then $\mathrm{Val}(t, X, \xi) = \dot{\xi}(\hat{f}[a_1, \ldots, a_n])$, and no query is caused by ξ.
- If, for each i, $\mathrm{Val}(t_i, X, \xi) = a_i$, if $f \in E$, and if $\hat{f}[a_1, \ldots, a_n] \notin \mathrm{Dom}(\dot{\xi})$, then $\mathrm{Val}(t, X, \xi)$ is undefined, and $\xi \vdash^t_X q$ for exactly one query, namely $q = \hat{f}[a_1, \ldots, a_n]$.

□

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

Lemma 5.2. *$\mathrm{Val}(t, X, \xi)$ is defined if and only if there is no query q such that $\xi \vdash^t_X q$.*

Lemma 5.3. *If $\xi \vdash^t_X q$ then $q \notin Dom(\dot{\xi})$.*

Lemma 5.4. *If $\eta \trianglelefteq \xi$ (or even if merely $\dot{\eta} \subseteq \dot{\xi}$) and if $Val(t, X, \eta)$ is defined, then $Val(t, X, \xi)$ is also defined and these values are equal.*

5.2 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 5.5. Let φ be a guard and ξ a history in state X.

- If φ is a Boolean term, then its value (if any) and causality relation are already given by Definition 5.1.
- If φ is $(s \preceq t)$ and if both s and t have values with respect to ξ, then $\mathrm{Val}(\varphi, X, \xi) = \texttt{true}$ 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, $\mathrm{Val}(\varphi, X, \xi) = \texttt{false}$. Also declare that $\xi \vdash^\varphi_X q$ 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 $\xi \vdash^\varphi_X q$ 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 $\xi \vdash^\varphi_X q$ 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 $\xi \vdash^\varphi_X q$ if and only if $\xi \vdash^s_X q$ or $\xi \vdash^t_X q$.
- If φ is $\psi_0 \curlywedge \psi_1$ and both ψ_i have value `true`, then $\mathrm{Val}(\varphi, X, \xi) = \texttt{true}$ 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 $\xi \vdash^\varphi_X q$ if and only if $\xi \vdash^{\psi_{1-i}}_X q$.

- 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 $\xi \vdash_X^{\psi} q$.

□

Remark 5.6. 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 5.4, 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. □

Example 5.7. 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 nullary constant 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 $\hat{p}[0_X]$. □

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

Lemma 5.8. *$Val(\varphi, X, \xi)$ is defined if and only if there is no query q such that $\xi \vdash_X^\varphi q$.*

Lemma 5.9. *If $\xi \vdash_X^\varphi q$ then $q \notin Dom(\dot{\xi})$.*

Lemma 5.10. *If $\eta \trianglelefteq \xi$ and if $Val(\varphi, X, \eta)$ is defined, then $Val(\varphi, X, \xi)$ is also defined and these values are equal.*

Remark 5.11. Given the semantics of guards, we can amplify the statement, at the end of Remark 2.10, that guards express descriptions like "p has reply a and p' has no reply." In view of Lemma 5.10, 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.

5.3 Rules

The semantics of a rule, for a state X 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 5.12. Let R be a rule and ξ a history for the state 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 `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 `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 `fail`, then ξ fails for R; it produces the empty update set and no queries.

- If R is a conditional rule `if` φ `then` R_0 `else` R_1 `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 `if` φ `then` R_0 `else` R_1 `endif` and if φ has value `true` (resp. `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.

□

There is no analog for rules of Lemmas 5.2 and 5.8. 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 5.13. *If* $\xi \vdash^R_X q$ *then* $q \notin Dom(\dot{\xi})$.

Lemma 5.14. *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.

Remark 5.15. It is tempting to view the definition of the semantics of parallel combination as actually defining an operation of parallel composition of algorithms. This temptation should be resisted. On an intuitive level, the reason is that, in composing algorithms in this way, we would not really be dealing with them as algorithms but rather as components. (See Remarks 2.13 and 3.1.) Formally, the problem is that this operation of parallel composition does not respect equivalence of algorithms. For example, consider two algorithms which differ only in that, in some states, one algorithm has $(q, r) \vdash q'$ as the only instance of causality, while the other has an empty causality relation. These algorithms are equivalent because the first algorithm will never get to use $(q, r) \vdash q'$ since it has no way to issue q. That is, no coherent history for either algorithm has q in its domain. Consider what happens when each of these two algorithms is combined in parallel with an algorithm whose causality relation includes $\varnothing \vdash q$. The first of the composite algorithms can issue q' (if the environment gives q the reply r), but the second cannot, so they are inequivalent. Returning to the intuitive level, we can say that the inequivalence of the composite algorithms indicates that the original two algorithms, though equivalent as stand-alone algorithms, are inequivalent as components. See [5] for details about components and their equivalence. □

Now that the semantic definitions for ASM rules are available, we can give the complete definition of ASMs.

Definition 5.16. An *interactive, small-step, ASM* consists of

- a finite vocabulary Υ,
- a finite set Λ of labels,
- a finite external vocabulary E,
- an ASM rule Π, using the vocabularies Υ and E, called the *program* of the ASM,
- a template assignment (with respect to E and Λ),
- a set $\mathcal{S}$ of Υ-structures called *states* of the ASM, and
- a 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 next state $\tau(X, \xi)$, as described in the Step Postulate (Part C), is also in $\mathcal{S}$. □

The rest of this section is devoted to checking that ASMs, as just defined, are algorithms, as defined in Section 2. Much of this checking is trivial: Everything required by the States Postulate is in our definition of ASMs. The causality relation required by the Interaction Postulate 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 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.

The ASM semantics provides notions of finality, success, failure, and updates. In addition to these, the Step Postulate requires (in Part C) a notion of next state and (in Part A) assurance that every complete, coherent history has a final initial segment.

As far as the next state is concerned, the Step Postulate tells us how it is to be defined in terms of the update sets, but we must check that the definition is consistent, i.e., that the updates produced by an attainable history $\xi \in \mathcal{F}_X^+$ do not clash. (Note the importance of the assumption $\xi \in \mathcal{F}_X^+$; clashes are

possible, but not in successful, final histories.) The required checking is a routine induction on ASM rules, showing that a final history must fail if its update set clashes. This is explicitly built into the semantic definition for parallel combinations, and all the remaining cases of the induction are trivial.

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 5.17. *If a history ξ is not final for a rule R in a state X, then $\xi \vdash_X^R q$ 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.

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 5.2 there is a query q such that $\xi \vdash_X^{t_i} q$, and therefore $\xi \vdash_X^R q$.

If R is a conditional rule whose guard has no value, then the same argument applies except that we invoke Lemma 5.8 in place of Lemma 5.2.

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, $\xi \vdash_X^{R_i} q$ for some q, and then we also have $\xi \vdash_X^R q$. □

To complete the verification of the Step Postulate, we observe that, in the situation of the lemma, $q \in \mathrm{Issued}_X^R(\xi)$ and, by Lemma 5.13, $q \notin \mathrm{Dom}(\dot{\xi})$. Thus, $q \in \mathrm{Pending}_X^R(\xi)$, and so ξ is not complete for R and X.

Remark 5.18. 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. □

We turn now to the Bounded Work Postulate. 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 5.19. *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 5.4, 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.) □

Since $\text{Issued}^R_X(\xi)$ is the set of queries caused in state X, under rule R, by initial segments of ξ, the lemma tells us that $|\text{Issued}^R_X(\xi)| \leq B(R)$, 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.

Before treating the first case, terms, we must adopt careful terminology, since we shall need to use, in the same context, two different sorts of terms.

Convention 5.20. By an Υ-*term* we mean a term built using the function symbols from the ASM's vocabulary Υ and variables. These are the terms

that can occur in a bounded exploration witness. By an *ASM-term* we mean a term built from the function symbols in $\Upsilon \cup E$. These are the terms that are used in the syntax of ASMs. In contrast to Υ-terms, ASM-terms can contain external function symbols but cannot contain variables. □

Lemma 5.21. *For every ASM-term t there exists a finite set $W(t)$ of Υ-terms such that, whenever (X, ξ) and (X', ξ) agree on $W(t)$, then:*

- *If $\xi \vdash^t_X q$ then $\xi \vdash^t_{X'} q$.*
- *Val(t, X, ξ) = Val(t, X', ξ).*

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 Range$(\dot{\xi})$. Recall also that an equation between possibly undefined expressions like $\mathrm{Val}(t, X, \xi)$ 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[2] 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 $W(t_i) \subseteq W(t)$.

[2] "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.

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 $f \in E$ 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. □

The preceding lemma easily implies the corresponding result for guards.

Lemma 5.22. *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 $\xi \vdash^{\varphi}_{X} q$ then $\xi \vdash^{\varphi}_{X'} q$.*
- *Val(φ, X, ξ) = Val(φ, X', ξ).*

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 `true` and `false` denote the same element in X and X' (because of agreement on $W(\varphi)$, which contains `true` and `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. □

Finally, we prove the corresponding result for rules.

Lemma 5.23. *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 `fail` then $W(R)$ is empty.

If R is a conditional rule `if` φ `then` R_0 `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 `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. □

6 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 in Definition 2.22) is equivalent (as in Definition 3.2) to an ASM (as in Definition 5.16).

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 and its Corollaries 2.25 and 2.26, 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 5.16,

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

6.1 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 5.16 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.

6.2 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 6.1. 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. (We could also regard

the present definition as an overestimation to be trimmed down after Π is defined.) □

Remark 6.2. 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. □

6.3 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 3.3 of equivalence of algorithms, 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 [8, 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 [8].

Definition 6.3. 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 $\mathrm{Range}(\dot{\xi})$ for the variables in t such that the resulting value for t is a. □

Lemma 6.4 (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 $\mathrm{Dom}(\dot{\xi})$ nor an element of $\mathrm{Range}(\dot{\xi})$. 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 $\Delta^+(X, \xi)$. □

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 obtain two kinds of possible values: the ordinary value (as in Definition 5.1) which is an element of the state, and also a query-value, which is a potential query.

Definition 6.5 (Critical Terms)**.** Recall that an ASM-term is a closed term of the vocabulary $\Upsilon \cup E$. If its outermost function symbol is external, we shall sometimes refer to it as a *query-term* or *q-term*.

- 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.

□

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, were defined in Definition 5.1. For q-terms, we need the additional notion of their query-value, defined as follows.

Definition 6.6. Let X be a state and ξ a history for it, let $t = f(t_1, \ldots, t_k)$ be a q-term, and let $\hat{f}$ be the template associated with its initial function symbol f. The *query-value* of t is

$$\text{q-Val}(t, X, \xi) = \hat{f}[\text{Val}(t_1, X, \xi), \ldots, \text{Val}(t_k, X, \xi)].$$

□

Recall from Definition 5.1 that the value of a term is not always defined; it depends on whether the necessary replies to queries are present in the history ξ. As a result, it is also possible also for q-terms to lack query-values. When values and query-values exist, however, they are always elements of (the base set of) the state X and queries over X respectively. The value of a q-term is obtained from its query-value by applying the answer function $\dot{\xi}$.

Recall also that, according to Lemma 5.4, 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. This monotonicity property immediately implies that query-values are monotone in the same sense. The next lemma records this fact and some related ones 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 6.7 (Invariance of Values)**.**

- *If ξ is an initial segment of η and z is a term such that $Val(z, X, \xi)$ exists, then $Val(z, X, \eta) = Val(z, X, \xi)$. Similarly, if z is a q-term such that q-$Val(z, X, \xi)$ exists, then q-$Val(z, X, \eta) =$ q-$Val(z, X, \xi)$.*
- *If $i : X \cong Y$ is an isomorphism, ξ is a history for X, and z is any ASM-term, then $i(Val(z, X, \xi)) = Val(z, Y, i(\xi))$. If z is a q-term, then also $i($q-$Val(z, X, \xi)) =$ q-$Val(z, Y, i(\xi))$.*
- *If (X, ξ) and (Y, ξ) agree on W, then $Val(z, X, \xi) = Val(z, Y, \xi)$ for all critical terms z, and q-$Val(z, X, \xi) =$ q-$Val(z, Y, \xi)$ for all critical q-terms z.*

Proof. The first assertion was proved above, and the other two are proved by routine inductions. □

Remark 6.8. 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 6.9 (Critical Terms Suffice)**.** *Let X be a state, ξ an attainable history for it, and n the length of ξ.*

- *Every query in $Dom(\dot{\xi})$ is the query-value (for X and ξ) of some critical q-term of level $\leq n$.*
- *Every element of $Range(\dot{\xi})$ 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 $Issued_X(\xi)$ 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 6.4, 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 $\leq n + 1$. □

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 6.7. We abbreviate the phrase "pair consisting of a state and an attainable history for it" as "attainable pair."

Lemma 6.10 (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 Range($\dot{\xi}$). But values in Range($\dot{\xi}$) are, by the Critical Terms Suffice Lemma 6.9, 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. □

6.4 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 6.11. 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 `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$_X(\xi-)$.

□

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 5.5) 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 6.12. *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, $q \in \text{Issued}_X(\xi-)$. Also, by Lemma 6.9, $q = \text{q-Val}(u, X, \xi)$ for some critical q-term u of level $\leq n$. Because q is in the last equivalence class with respect to ξ, $\text{Val}(u, X, \xi)$ exists but $\text{Val}(u, X, \xi-)$ does not. Now if $\text{q-Val}(v, X, \xi-)$, which is also $\text{q-Val}(v, X, \xi)$, is in $\text{Dom}(\dot{\xi})$, then $\text{Val}(v, X, \xi)$ exists and so $\delta(X, \xi)$ contains the conjunct $(v \preceq u)$. Otherwise, $\text{Val}(v, X, \xi)$ 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. □

Corollary 6.13. *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 Dom$(\dot{\xi})$.*

Proof. Immediate from the third of the equivalent statements in the lemma. □

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

Lemma 6.14 (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 third clause of Invariance of Values Lemma 6.7 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 second clause of the same lemma. □

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 6.17, that the composition of agreement and isomorphism is not only sufficient but also necessary for similarity.

Definition 6.15. Two attainable pairs are *similar* if they have the same descriptions. □

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 6.16. *Let (X, ξ) and (Y, η) be attainable pairs. Suppose $\delta(X, \xi)$ has value* `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 j^{th} equivalence class. Letting $\xi \restriction j$ denote the initial segment of ξ of length j, and applying Lemma 6.9, 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 6.9, 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 j^{th} 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 6.9, 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 6.7, 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 j^{th} 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 n^{th} 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 $\mathrm{Val}(u, Y, \eta \restriction m)$ exists is the m such that $i(q)$ is in the m^{th} 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 j^{th} equivalence class with respect to ξ into the j^{th} 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 n^{th} 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 n^{th} and possibly later" can be improved to "the n^{th}".

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 $\text{q-Val}(u', X, \xi) = q'$ and thus, by definition of i, $\text{q-Val}(u', Y, \eta) = i(q')$. We record for future reference that, since $\text{q-Val}(u', X, \xi) \in \text{Dom}(\dot{\xi})$, 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 6.9, 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

$q' \in \mathrm{Dom}(\dot{\xi})$, 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 $\text{q-}\mathrm{Val}(z, Y, \eta)$ (which is also $\text{q-}\mathrm{Val}(z, Y, \eta')$ as the induction hypothesis applies to the arguments of z) is in the domain of η but not in the domain of η'. But $\text{q-}\mathrm{Val}(z, X, \xi)$ 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, $\text{q-}\mathrm{Val}(z, Y, \eta)$. 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 6.10 completes the proof that (Z, η') and (Y, η') agree on W. □

Corollary 6.17 (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 6.16 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. □

Corollary 6.18 (Similarity Suffices)**.** *Let (X, ξ) and (Y, η) be similar attainable pairs. Let n be the length of ξ (and of η, by Corollary 6.17). Then*

- *If u is a q-term of level $\leq n + 1$ and $\xi \vdash_X q\text{-}Val(u, X, \xi)$ then $\eta \vdash_Y q\text{-}Val(u, Y, \eta)$.*
- *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 $Val(t_i, X, \xi)$ 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 $Val(t_i, Y, \eta)$.*

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. □

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 6.19. 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 $\langle$`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 $\mathrm{Dom}(\dot{\xi}) = \{q\}$ 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 6.20. *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 6.17, 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 6.14, $(X, \xi-)$ 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 6.21. *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 6.17, 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., $\eta- = \xi$. □

6.5 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 6.16 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 6.18, 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 6.18. This completes the definition of the program Π.

Remark 6.22. 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.

6.6 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 6.23. *The ASM defined by* Π *together with* $\mathcal{S}$, $\mathcal{I}$, Υ, Λ, E, *and the template assignment of subsection 6.2 is equivalent to algorithm* A.

Proof. Referring to Lemma 3.3, 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. $\mathrm{Issued}_X(\xi)$ 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 6.16, 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 6.16 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 6.21. 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 $\mathrm{Dom}(\dot{\eta})$; simply define η to give q an arbitrary reply and to do nothing more. By Lemma 6.9, 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 6.16, that exactly one of them is true, namely $\delta(X, \xi \restriction 1)$, so it pro-

ceeds 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 6.21. 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 $(Y, \eta-) = (Y, \xi\restriction 1)$. Since $(Y, \xi\restriction 1)$ agrees with $(X, \xi\restriction 1)$ on W, these are queries issued by A in $(X, \xi\restriction 1)$.

The converse also holds. If a query q is issued by A in $(X, \xi\restriction 1)$, but not already in $(X, \xi\restriction 0)$, then there is an attainable history η for X, which has $\xi\restriction 1$ 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 $\xi\restriction 1$ to give q an arbitrary reply, in a new, second equivalence class, and to do nothing more. By Lemma 6.9, 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 $R_{\delta(X,\xi\restriction 1)}$ is exactly parallel to the previous argument concerning $R_{\delta(X,\xi\restriction 0)}$. 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 $R_{\delta(X,\xi\restriction n)} = R_{\delta(X,\xi)}$. 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 6.16, 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 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. □

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*, to appear.

[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, "Composite interactive algorithms" (tentative title), in preparation.

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

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

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