Intra-Step Interaction*

Yuri Gurevich

Microsoft Research
One Microsoft Way, Redmond, WA 98052, USA

Abstract. For a while it seemed possible to pretend that all interaction between an algorithm and its environment occurs inter-step, but not anymore. Andreas Blass, Benjamin Rossman and the speaker are extending the Small-Step Characterization Theorem (that asserts the validity of the sequential version of the ASM thesis) and the Wide-Step Characterization Theorem (that asserts the validity of the parallel version of the ASM thesis) to intra-step interacting algorithms.

1 Why Intra-Step Interaction

According to the Lipari guide [9], an abstract state machine interacts with the environment by means of external functions. "The computation steps of a program are supposed to be atomic at an appropriate level of abstraction. A computation step is hardly atomic if during that step the [ASM] queries an oracle and then, depending on the result, submits another query to the same or a different oracle. Thus it seems reasonable to forbid nesting of external functions. Indeed, the need to nest external functions has not arisen in applications so far. But we withhold final judgment and wait for more experimentation."

It was a good idea to withhold final judgment. Many things happened in the meantime. Probably the most consequential — as far as intra-step interaction is concerned — is the development of AsmL, the Abstract State Machine Language [2]. It is not unusual for an AsmL program π to nest external functions. For example, π may have a do-in-parallel form where one of the constituents calls another agent a', gets a callback from a', calls a' again, etc. and thus executes a nontrivial protocol, possibly involving additional agents. All this is done in a single step. (And if you break the protocol into several steps then the do-in-parallel structure is all messed up.)

One may pretend that every interaction between the algorithm and the environment is inter-step. That point of view is taken in [10]. But, as the amount of intra-step interaction increases, it gets harder and less natural to maintain the pretense Of course, an omniscient environment can know ahead of time what external function calls the algorithm will generate during a step. It can thus prepare the answers and store them, so that the algorithm would treat external

* Proc. ASM'2004, Springer Lecture Notes in Computer Science

functions as internal. But a typical environment is not omniscient. And if the environment is omniscient, does it need our algorithm at all?

The time came to recognize the importance of intra-step communication and to deal with it directly. Notice that we have been dealing with intra-step communication all along, albeit implicitly. Consider for example the import command as in the Lipari guide.

```
import v
   R(v)
endimport
```

What really happens here is that the program sends an import query to the environment, and the environment returns a reserve element. The creation of objects in object-oriented languages like C# [8] or Java [11] is similar (when viewed on the abstraction level of the program). Here is another example involving one of the AsmL choose constructs:

```
any x | x in {1, 2, 3} where x > 1
```

To evaluate this expression, the program sends out a choose query and provides a parameter $\{2, 3\}$. The environment replies with an element of $\{2, 3\}$.

In the rest of this extended abstract, an algorithm is intra-step interactive unless the contrary is stated explicitly.

2 Terminology

We restrict attention to sequential time algorithms. Here sequential time means that the computation proceeds in a sequence of discrete steps [10]. We recall some useful terminology introduced recently in [4] and used already in [5].

Small-step algorithms We quote from [5]: "*Small-Step Algorithms* are characterized by two properties:

- (Step) The computation proceeds in a sequence of discrete steps.
- (Small) The amount of work done by the algorithm in any one step is bounded; the bound depends only on the algorithm, not on the state, nor on the input, nor on the actions of the environment.

Many authors call such algorithms sequential (for example the present authors in [3, 10]), but other authors use 'sequential' to mean only the first of these properties (which we call *sequential time*). Because of this ambiguity, we now prefer the term 'small-step'."

Wide-Step Algorithms We quote from [4]: "The term 'parallel algorithm' is used for a number of different notions in the literature. We have in mind sequential-time algorithms that can exhibit unbounded parallelism but only bounded sequentiality within a single step. Bounded sequentiality means that there is an *a priori* bound on the lengths of sequences of events within any one step of the algorithm that must occur in a specified order. To distinguish this notion of parallel algorithms, we call such parallel algorithms *wide-step*. Intuitively the width is the amount of parallelism. The 'step' in 'wide-step' alludes to sequential time."

Small-Step Characterization Theorem This is Theorem 6.13 in [10], according to which every noninteractive small-step algorithm is behaviorally identical to some noninteractive small-step abstract state machine. The notion of noninteractive small-step algorithms is defined axiomatically. Noninteractive small-step ASMs are the sequential ASMs of the Lipari guide without the import command. The point of removing the import command is to make these ASMs noninteractive. In a trivial way, the theorem generalizes to algorithms whose interaction with the environment is purely inter-step.

Wide-Step Characterization Theorem This is Theorem 10.1 in [3] according to which every noninteractive wide-step algorithm is behaviorally identical to some noninteractive wide-step abstract state machine. The notion of noninteractive wide-step algorithms is defined axiomatically. The notion of a noninteractive wide-step ASM is a variant of the Lipari guide notion of parallel ASM. In a trivial way, the theorem generalizes to algorithms whose interaction with the environment is purely inter-step.

3 Ordinary Small-Step Algorithms

This section reflects joint work with Andreas Blass.

What happens if an external function of an ASM "stalls"? The Lipari guide does not address the issue directly but the tradition has been that the ASM is stuck waiting for the environment to deliver a reply to the query. Typically such a behavior is correct. The algorithm may be waiting e.g. for a user to key in necessary information.

However, it does not follow from first principles that, in order to finish a step, an algorithm should wait for the replies to all queries. Consider for example the following algorithm.

> Start a step by posing Boolean queries α and β. If both of them return `true` then output 7, finish the step and halt. If at least one of them returns `false` then output 11, finish the step and halt.

Breaking another assumption implicit in the Lipari guide, we can make the above algorithm sensitive to the order in which the replies to α and β occur.

> Start a step by posing Boolean queries α and β. If both of them return `true` and the reply to α precedes the reply to β then output 6, finish the step and halt. If both of them return `true` but the reply to α does not precede the reply to β (so that either the reply to β precedes the reply to α or else the two replies occur simultaneously) then output 8, finish the step and halt. If at least one of them returns `false` then output 11, finish the step and halt.

On the other hand, we have yet to see applications violating either of the two implicit assumptions. "Accordingly, we study in this paper", we quote from [5], "those algorithms, the *ordinary* ones, that

- never complete a step until all queries from that step have been answered and
- use no information from the environment beyond the function assigning answers to queries.

Here we count a time-out signal as an answer. A more explicit formulation of the second aspect of ordinariness is that whatever the algorithm does is completely determined by its program, its current state, and the answers the environment has already provided for earlier queries."

Ordinary small-step algorithms are axiomatized in [5]. In [6], we generalize the Small-Step Characterization Theorem to ordinary small-step algorithms.

The ordinary small-step ASMs do not adhere to the Lipari guide completely. They can nest external functions. And there is something else. The Lipari guide interprets distinct invocations of the same external function with the same arguments during the same step as the same query.

> Think about an external function as a (dynamic) oracle. The [ASM] provides the arguments and the oracle gives the result. The oracle need not be consistent and may give different results for the same argument at different times. ...
> However, the oracle should be consistent during the execution of any one step of the program. In an implementation, this may be achieved by not reiterating the same question during a one-step execution. Ask the question once and, if necessary, save the result and reuse it.

This interpretation is too narrow. Sometimes the opposite interpretation is appropriate. For example, all invocations of the import command give rise to distinct queries. (Even though import was not treated as an external function in the Lipari guide, such treatment is natural.) But neither of these two extreme interpretations is appropriate in all cases. We need to be more flexible and general. That issue is taken up in [6].

4 General Small-Step Algorithms

This section reflects joint work with Andreas Blass and Benjamin Rossman.

To deal with arbitrary (intra-step interactive) small-step algorithms, we take advantage of the fact that a small-step algorithm presupposes the existence of a single agent who is in charge of the algorithm's execution. This key observation allowed us to axiomatize general small-step algorithms [7]. The work on the generalization of the Small-Step Characterization Theorem to intra-step interactive algorithms is in progress.

5 Wide-Step Algorithms

In order to generalize the Wide-Step Characterization Theorem to intra-step interactive algorithms, we need to "marry" the advance on intra-step interaction, sketched above, with the analysis of wide-step algorithms in [3]. Interestingly, the analysis may be simplified in the following aspect. In [3], every proclet (a small-step component of the wide-step process) can update its mailbox at most once during any one step of the whole wide-step process. In the new framework that allows intra-step communication, a proclet can update its mailbox several times during a single step. This gives some technical benefits.

References

1. ASM Michigan Webpage, `http://www.eecs.umich.edu/gasm/`, maintained by James K. Huggins.
2. The AsmL webpage, `http://research.microsoft.com/foundations/AsmL/`.
3. Andreas Blass and Yuri Gurevich, "Abstract state machines capture parallel algorithms," *A. C. M. Trans. Computational Logic*, 4:4 (2003), 578–651.
4. Andreas Blass and Yuri Gurevich, "Algorithms: A Quest for Absolute Definitions," Bulletin of the European Association for Theoretical Computer Science, Number 81 (October 2003), 195–225.
5. Andreas Blass and Yuri Gurevich, "Ordinary Interactive Small-Step Algorithms, I", Technical Report MSR-TR-2004-16, Microsoft Research, February 2004.
6. Andreas Blass and Yuri Gurevich, "Ordinary Interactive Small-Step Algorithms, II", in preparation.
7. Andreas Blass, Yuri Gurevich, and Benjamin Rossman, "General interactive small-step algorithms" (tentative title), in preparation.
8. Anders Hejlsberg, Scott Wiltamuth and Peter Golde, "The C# Programming Language", Addison-Wesley, 2003.
9. Yuri Gurevich, "Evolving algebra 1993: Lipari guide," in *Specification and Validation Methods*, E. Börger, editor, Oxford University Press (1995), 9–36.
10. Yuri Gurevich, "Sequential abstract state machines capture sequential algorithms," *A. C. M. Trans. Computational Logic* 1:1 (2000), 77–111.
11. Bill Joy, Guy Steele, James Gosling and Gilad Bracha, "Java (TM) Language Specification", Addison-Wesley Pub, 2nd edition, 2000.