Abstract Effective Models

Udi Boker and Nachum Dershowitz
School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel
{udiboker,nachumd}@tau.ac.il

Abstract

We modify Gurevich's notion of abstract machine so as to encompass computational models, that is, sets of machines that share the same domain. We also add an effectiveness requirement. The resultant class of "Effective Models" includes all known Turing-complete state-transition models, operating over any countable domain.

1 Sequential Procedures

We first define "sequential procedures", along the lines of the "sequential algorithms" of [3]. These are abstract state transition systems, whose states are algebras.

Definition 1 (States).

- *A* state *is a structure (algebra) s over a (finite-arity) vocabulary $\mathcal{F}$, that is, a domain (nonempty set of elements) D together with interpretations $[\![f]\!]_s$ over D of the function names $f \in \mathcal{F}$.*
- *A* location *of vocabulary $\mathcal{F}$ over a domain D is a pair, denoted $f(\overline{a})$, where f is a k-ary function name in $\mathcal{F}$ and $\overline{a} \in D^k$.*
- *The* value *of a location $f(\overline{a})$ in a state s, denoted $[\![f(\overline{a})]\!]_s$, is the domain element $[\![f]\!]_s(\overline{a})$.*
- *We sometimes use a term $f(t_1, \ldots, t_k)$ to refer to the location $f([\![t_1]\!]_s, \ldots, [\![t_k]\!]_s)$.*
- *Two states s and s' over vocabulary $\mathcal{F}$ with the same domain* coincide *over a set T of $\mathcal{F}$-terms if $[\![t]\!]_s = [\![t]\!]_{s'}$ for all terms $t \in T$.*
- *An* update *of location l over domain D is a pair, denoted $l := v$, where $v \in D$.*
- *The* modification *of a state s into another state s' over the same vocabulary and domain is $\Delta(s, s') = \{l := v' \mid [\![l]\!]_s \neq [\![l]\!]_{s'} = v'\}$.*
- *A* mapping *$\rho(s)$ of state s over vocabulary $\mathcal{F}$ and domain D via injection $\rho : D \to D'$ is a state s' of vocabulary $\mathcal{F}$ over D', such that $\rho([\![f(\overline{a})]\!]_s) = [\![f(\rho(\overline{a}))]\!]_{s'}$ for every location $f(\overline{a})$ of s.*
- *Two states s and s' over the same vocabulary with domains D and D', respectively, are* isomorphic *if there is a bijection $\pi : D \leftrightarrow D'$, such that $s' = \pi(s)$.*

A "sequential procedure" is like Gurevich's [3] "sequential algorithm", with two modifications for computing a specific function, rather than expressing an abstract algorithm: the procedure vocabulary includes special constants "*In*" and "*Out*"; there is a single initial state, up to changes in *In*.

Definition 2 (Sequential Procedures).

- *A* sequential procedure A *is a tuple* $\langle \mathcal{F}, \mathit{In}, \mathit{Out}, D, \mathcal{S}, \mathcal{S}_0, \tau\rangle$*, where:* $\mathcal{F}$ *is a finite vocabulary; In and Out are nullary function names in* $\mathcal{F}$*;* D*, the procedure* domain*, is a domain;* $\mathcal{S}$*, its* states*, is a collection of structures of vocabulary* $\mathcal{F}$*, closed under isomorphism;* $\mathcal{S}_0$*, the* initial states*, is a subset of* $\mathcal{S}$ *over the domain* D*, containing equal states up to changes in the value of In (often referred to as a single state* s_0*); and* $\tau : \mathcal{S} \to \mathcal{S}$*, the* transition function*, such that:*

  - **Domain invariance**. *The domain of* s *and* $\tau(s)$ *is the same for every state* $s \in \mathcal{S}$*.*
  - **Isomorphism constraint**. $\tau(\pi(s)) = \pi(\tau(s))$ *for some bijection* π*.*
  - **Bounded exploration**. *There exists a finite set* T *of "critical" terms, such that* $\Delta(s, \tau(s)) = \Delta(s', \tau(s'))$ *if* s *and* s' *coincide over* T*.*

  Tuple elements of a procedure A *are indexed* $\mathcal{F}_A$*,* τ_A*, etc.*

- *A* run *of a procedure* A *is a finite or infinite sequence* $s_0 \leadsto_\tau s_1 \leadsto_\tau s_2 \leadsto_\tau \cdots$*, where* s_0 *is an initial state and every* $s_{i+1} = \tau_A(s_i)$*.*

- *A run* $s_0 \leadsto_\tau s_1 \leadsto_\tau s_2 \leadsto_\tau \cdots$ terminates *if it is finite or if* $s_i = s_{i+1}$ *from some point on.*

- *The* terminating state *of a terminating run* $s_0 \leadsto_\tau s_1 \leadsto_\tau s_2 \leadsto_\tau \cdots$ *is its last state if it is finite, or its stable state if it is infinite. If there is a terminating run beginning with state* s *and terminating in state* s'*, we write* $s \leadsto^!_\tau s'$*.*

- *The* extensionality *of a sequential procedure* A *over domain* D *is the partial function* $f : D \to D$*, such that* $f(x) = [\![\mathit{Out}]\!]_{s'}$ *whenever there's a run* $s \leadsto^!_\tau s'$ *with* $[\![\mathit{In}]\!]_s = x$*, and is undefined otherwise.*

Domain invariance simply ensures that a specific "run" of the procedure is over a specific domain. The isomorphism constraint reflects the fact that we are working at a fixed level of abstraction. See [3, p. 89]. The bounded-exploration constraint ensures that the behavior of the procedure is effective. This reflects the informal assumption that the program of an algorithm can be given by a finite text [3, p. 90].

2 Programmable Machines

The transition function of a "programmable machine" is given by a finite "flat program":

Definition 3 (Programmable Machines).

- *A* flat program P *of vocabulary* $\mathcal{F}$ *has the following syntax:*

```
if $x_{11} \doteq y_{11}$ and $x_{12} \doteq y_{12}$ and $\ldots$ $x_{1k_1} \doteq y_{1k_1}$
then $l_1 := v_1$
if $x_{21} \doteq y_{21}$ and $x_{22} \doteq y_{22}$ and $\ldots$ $x_{2k_2} \doteq y_{2k_2}$
then $l_2 := v_2$
$\vdots$
if $x_{n1} \doteq y_{n1}$ and $x_{n2} \doteq y_{n2}$ and $\ldots$ $x_{nk_n} \doteq y_{nk_n}$
then $l_n := v_n$
```

  where each $\doteq$ *is either '*$=$*' or '*$\neq$*',* $n, k_1, \ldots, k_n \in \mathbf{N}$*, and all the* x_{ij}*,* y_{ij}*,* l_i*, and* v_i *are* $\mathcal{F}$*-terms.*

- *Each line of the program is called a* rule.
- *The* activation *of a flat program P on an $\mathcal{F}$-structure s, denoted $P(s)$, is a set of updates $\{l := v \mid \texttt{if}\ p\ \texttt{then}\ l := v \in P, [\![p]\!]_s\}$ (under the standard interpretation of $=$, $\neq$, and conjunction), or the empty set $\emptyset$ if the above set includes two values for the same location.*
- *A* programmable machine *is a tuple $\langle \mathcal{F}, In, Out, D, \mathcal{S}, \mathcal{S}_0, P\rangle$, where all but the last component is as in a sequential procedure (Definition 2), and P is a flat program of $\mathcal{F}$.*
- *The* run *of a programmable machine and its* extensionality *are defined as for sequential procedures (Definition 2), where the transition function τ is given by $\tau(s) = s' \in \mathcal{S}$ such that $\Delta(s, s') = P(s)$.*

To make flat programs more readable, we combine rules, as in

```
% comment
if cond-1
   stat-1
   stat-2
else
   stat-3
```

Analogous to the the main lemma of [3], one can show that every programmable machine is a sequential procedure, and every sequential procedure is a programmable machine.

In contradistinction to those Abstract Sequential Machines (ASMs), we do not have built in equality, booleans, or an undefined in the definition of procedures: The equality notion is not presumed in the procedure's initial state, nor can it be a part of the initial state of an "effective procedure", as defined below. Rather, the transition function must be programmed to perform any needed equality checks. Boolean constants and connectives may be defined like any other constant or function. Instead of a special term for undefined values, a default domain value may be used explicitly.

3 Effective Models

We define an "effective procedure" as a sequential procedure satisfying an "initial-data" postulate (Axiom E below). This postulate states that the procedures may have only finite initial data in addition to the domain representation ("base structure"). An "effective model" is, then, any set of effective procedures that share the same domain representation.

We formalize the finiteness of the initial data by allowing the initial state to contain an "almost-constant structure". Since we are heading for a characterization of effectiveness, the domain over which the procedure actually operates should have countably many elements, which have to be nameable. Hence, without loss of generality, one may assume that naming is via terms.

Definition 4 (Almost-Constant and Base Structures).

- *A structure S is* almost constant *if all but a finite number of locations have the same value.*
- *A structure S of finite vocabulary $\mathcal{F}$ over a domain D is a* base structure *if all the domain elements are the value of a unique $\mathcal{F}$-term. That is, for every element $e \in D$ there exists a unique $\mathcal{F}$-term t such that $[\![t]\!]_S = e$.*
- *A structure S of vocabulary $\mathcal{F}$ over domain D is the* union *of structures S' and S'' of vocabularies $\mathcal{F}'$ and $\mathcal{F}''$, respectively, over D, denoted $S = S' \uplus S''$, if $\mathcal{F} = \mathcal{F}' \uplus \mathcal{F}''$, $[\![l]\!]_S = [\![l]\!]_{S'}$ for every location l of S', and $[\![l]\!]_S = [\![l]\!]_{S''}$ for every location l of S''.*

A base structure is isomorphic to the standard free term algebra (Herbrand universe) of its vocabulary.

Proposition 1. *Let S be a base structure over vocabulary G and domain D. Then:*

- *Vocabulary G has at least one nullary function.procedure.*
- *Domain D is countable.*
- *Every domain element is the value of a most one location of S.*

Axiom E (Initial Data). *The procedure's initial states consist of an infinite base structure and an almost-constant structure. That is, for some infinite base structure BS and almost-constant structure AS, and for every initial state s_0, we have $s_0 = \mathit{BS} \uplus \mathit{AS} \uplus \{\mathit{In}\}$ for some In.*

Definition 5 (Effective Procedures and Models).

- *An* effective procedure *A is a sequential procedure satisfying the initial-data postulate. An effective procedure is, accordingly, a tuple $\langle \mathcal{F}, \mathit{In}, \mathit{Out}, D, \mathcal{S}, \mathcal{S}_0, \tau, \mathit{BS}, \mathit{AS} \rangle$, adding a base structure BS and an almost-constant structure AS to the sequential procedure tuple, defined in Definition 2.*
- *An* effective model *E is a set of effective procedures that share the same base structure. That is, $\mathit{BS}_A = \mathit{BS}_B$ for all effective procedures $A, B \in E$.*

A computational model might have some predefined complex operations, as in a RAM model with built-in integer multiplication. Viewing such a model as a sequential algorithm allows the initial state to include these complex functions as oracles [3]. Since we are demanding effectiveness, we cannot allow arbitrary functions as oracles, and force the initial state to include only finite data over and above the domain representation (Axiom E). Hence, the view of the model at the required abstraction level is accomplished by "big steps", which may employ complex functions, while these complex functions are implemented by a finite sequence of "small steps" behind the scenes. That is, (the extensionality of) an effective procedure may be included (as an oracle) in the initial state of another effective procedure. (Cf. the "turbo" steps of [2].)

4 Effective Includes Computable

Turing machines, and other computational methods, can be shown to be effective. For instance, counter machines (CM) can be described by the following effective model E: The domain is the natural numbers $\mathbf{N}$. The base structure consists of a nullary function *Zero* and a unary function *Succ*, interpreted as the regular successor over $\mathbf{N}$. The almost-constant structure has the vocabulary ($name/arity$): *Out*$/0$, $CurrentLine/0$, $Pred/1$, $Next/1$, $Reg\_0, \ldots, Reg\_n/0$, and $Line\_1, \ldots, Line\_k/0$. Its initial data are $True = 1$, $Line\_i = i$, and all other locations are 0. The same structure applies to all machines, except for the number of registers (Reg_i) and the number of lines ($Line\_i$). For every counter machine $m \in \mathrm{CM}$ define an effective procedure $m' \in E$ with the following flat program:

```
% Initialization: fill the values of the
% predecessor function up to the value
% of the input
if CurrentLine = Zero
   if Next = Succ(In)
      CurrentLine := Line_1
   else
      Pred(Succ(Next)) := Next
      Next := Succ(Next)
% Simulate the counter-machine program.
% The values of a,b,c and d are as in
% the CM-program lines.
if CurrentLine = Line_1
   Reg_a := Succ(Reg_a) % or Pred(Reg_a)
   Pred(Succ(Reg_a)) := Reg_a
   if Reg_b = Zero
      CurrentLine := c
   else
      CurrentLine := d
if CurrentLine = Line_2
   ...
% Always:
Out := Reg_0
```

5 Discussion

In [3], Gurevich proved that any algorithm satisfying his postulates can be represented by an Abstract State Procedure. But an ASM is designed to be “abstract”, so is defined on top of an arbitrary structure that may contain *non-effective* functions. Hence, it may compute non-effective functions. We have adopted Gurevich’s postulates, but added an additional postulate (Axiom E) for effectivity: an algorithm’s initial state may contain only finite data in addition to the domain representation. Different runs of the same procedure share the same initial data, except for the input; different procedures of the same model share a base structure.

Here, we showed that counter machines are effective models. In [1], we prove the flip side, namely that Turing machines can simulate all effective models. To cover hypercomputational models, one would need to relax the effectivity axiom or the bounded exploration requirement.

References

[1] Udi Boker and Nachum Dershowitz, A formalization of the Church-Turing Thesis, submitted.

[2] N. G. Fruja and R. F. Stärk. The hidden computation steps of Turbo Abstract State Machines. In E. Börger, A. Gargantini, and E. Riccobene, editors, *Abstract State Machines — Advances in Theory and Applications, 10th International Workshop, ASM 2003, Taormina, Italy*, pages 244–262. Springer-Verlag, Lecture Notes in Computer Science 2589, 2003.

[3] Yuri Gurevich. Sequential abstract state machines capture sequential algorithms. *ACM Transactions on Computational Logic*, 1:77–111, 2000.