Chapter 8

Introduction to Abstract State Machines

8.1 A New Model of Sequential Computation

In this section we present an approach to a formal definition of computability, which extends the usual approaches using e.g. Turing Machines. This approach has been advocated by Yuri Gurevich since 1984. The earliest reference is probably [Gurevich, 1984]. The most concise exposition to date is [Gurevich, 2000]. We also used [Reisig, 2001].

8.1.1 The *Sequential Time* Postulate

Definition 12 (Sequential Time Postulate) *An algorithm A is determined by*

1. *A set $\mathcal{S}(A)$ of states,*
2. *A subset $\mathcal{I}(A)$ of $\mathcal{S}(A)$, called the initial states of A,*
3. *A function $\tau_A : \mathcal{I}(A) \rightarrow \mathcal{I}(A)$, called the one-step transformation of A.*

Definition 13 (Run) *A run of an algorithm A is a finite or infinite sequence $\langle X_i \rangle$ of states such that*

1. *X_0 is an initial state, i.e. $X_0 \in \mathcal{I}(A)$,*
2. *For every index i in the sequence $\tau_A(X_i) = X_{i+1}$.*

Comments This definition incorporates a particular way of treating the end of a run, if the run terminates. A state $X \in \mathcal{S}(A)$ is final if $\tau_A(X)$ is not defined. One could alternatively have introduced an explicit set $\mathcal{F}(A)$ of final states. This is not a crucial decision.

8.1.2 The *Abstract State* Postulate

Definition 14 (Abstract State Postulate) *Let A be an algorithm.*

1. *The states of A are structures of first-order logic.*

2. *All states of A have the same finite vocabulary, Σ_A.*

3. *The one-step transformation τ_A does not change the base set of any state.*

4. *The sets of states $\mathcal{S}(A)$ and $\mathcal{I}(A)$ are closed under isomorphism.*

5. *Any isomorphism between states X and Y is also an isomorphism between $\tau_A(X)$ and $\tau_A(Y)$.*

Comments The vocabulary Σ_A will typically be many-sorted.

Requirement 4 is probably the least palatable. If we want to talk about an algorithm on the natural numbers $\mathbb{N}$, why should we be forced to consider isomorphic copies of $\mathbb{N}$? This also introduces the set theoretic problem, that the class of all isomorphic copies of $\mathbb{N}$ is not a set. Of course, there are standard ways to deal with this problem. But, could it not have been avoided alltogether? It turns out that all that is needed in the development of a theory of sequential algorithms so far is Lemma 3 below (see page 123). We suggest to take the claim of this Lemma as a postulate and waive closure under isomorphism.

8.1.3 The *Bounded Exploration* Postulate

Definition 15 (Updates) *Let A be some algorithm, and X a state in $\mathcal{S}(A)$.*

1. *A location of X is a pair $(f, \bar{a})$ where f is a function symbol and $\bar{a}$ is a sequence of elements from X. The length of the sequence $\bar{a}$ coincides with the arity of f.*

2. *If $(f, \bar{a})$ is a location for X then $Content_X(f, \bar{a})$ is the element $f^X(\bar{a})$. Here f^X is the interpretatin of the function symbol f in X,*

3. *An update for X is a tripel $(f, \bar{a}, b)$ where $(f, \bar{a})$ is a location for X and b is an element of X.*

4. *An update $(f, \bar{a}, b)$ is trivial if $b = Content_X(f, \bar{a})$.*

5. *Two updates $(f_1, \bar{a}_1, b_1)$, $(f_2, \bar{a}_2, b_2)$* clash *if $f_1 = f_2$, $\bar{a}_1 = \bar{a}_2$ and $b_1 \neq b_2$.*

6. *A set of updates Δ is consistent if if does not contain two clashing updates.*

Definition 16 *Let X be a state, Δ a consistent set of updates for X. Then $X + \Delta$ is the state Y with the same base set as X and the interpretation of function symbols f given by:*

$$f^Y(\bar{a}) = \begin{cases} f^X(\bar{a}) & \text{if there is no } c \text{ such that } (f, \bar{a}, c) \in \Delta, \\ b & \text{if } (f, \bar{a}, b) \in \Delta. \end{cases}$$

Lemma 2 *If X, Y are two first-order structures with the same vocabulary and the same base set then there is a unique consistent set Δ of non-trivial updates such that $Y = X + \Delta$*

Proof Obvious, see also [Gurevich, 2000].

∎

Definition 17 *If A is an algorithm given by $(\mathcal{S}(A), \mathcal{I}(A), \tau_A)$, then we denote for every $X \in \mathcal{S}(A)$ the unique set Δ satisfying $X + \Delta = \tau_A(X)$ by $\Delta(A, X)$.*

Definition 18 (Bounded Exploration Postulate) *Let A be an algorithm. Then there exists a finite set T of (ground) terms in the vocabulary of Σ_A such that for any two states X, Y such that*

$$t^X = t^Y \text{ for all } t \in T$$

then

$$\Delta(A, X) = \Delta(A, Y)$$

The set T is called the set of critical terms for A. In [Gurevich, 2000] T is called a bounded exploration witness *for A.*

The following definition sums up our considerations so far

Definition 19 (Sequential Algorithm) *A sequential algorithm A is an object determined by $(\mathcal{S}(A), \mathcal{I}(A), \tau_A)$ satisfying the postulates 12, 14 and 18.*

Comments

Lemma 3 *Let A be an algorithm and T its finite set of critical terms and T_1 the smallest set of terms containing T and all subterms of terms in T.*

For any state $X \in \mathcal{S}(A)$ and any update $(f, (a_1, \ldots, a_j), a_0) \in \Delta(A, X)$ there is for every i, $0 \leq i \leq j$ a term $t_i \in T_1$ such that $t_i^X = a_i$.

Proof Assume for the sake of a contradiction that for some i, $0 \leq i \leq j$ we have $a_i \neq t^X$ for all $t \in T_1$. Let Y be an isomorphic copy of X arising from replacing a_i by a new element b, i.e. b does not occur in the base set of X and a_i does not occur in the base set of Y. By the postulate 14 Y is also a state in $\mathcal{S}(A)$. It is easy to see that for all $t \in T_1$ we have $t^X = t^Y$. Thus by 18 we have $\Delta(A, X) = \Delta(A, Y)$. Thus $(f, (a_1, \ldots, a_j), a_0) \in \Delta(A, Y)$, but this contradicts the fact the a_i is not in the base set of Y.

This is basically the argument given in [Gurevich, 2000, Lemma 6.2].

∎

Corollary 4 *Let A be an algorithm. Then there is a finite number n such that for all states X the set $\Delta(A, X)$ has not more than n elements.*

Proof Let T be the finite set of critical terms of A and T_1 the smallest set of terms containing T and all subterms of terms in T. Obviously T_1 is still finite. From this the finiteness claim obviously follows.

∎

8.1.4 Example: A Geometric Algorithm

One of the advantages of the definition of an algorithm given in Definition 19 is the fact that no coding in natural numbers or any other fixed data structure is required. An algorithm may be described at any chosen level of abstraction. We want to illustrate this by a geometric algorithm A_{3C}, which determines for three given points P_1, P_2, P_3 in the plane the centre M of the circle that hits all three points.

The structures in $\mathcal{S}(A_{3C})$ will be many-sorted structure with the sorts *Point*, *Line*, *Circle*, and *PairOfPoints*. The intuitive meaning of the last sort,

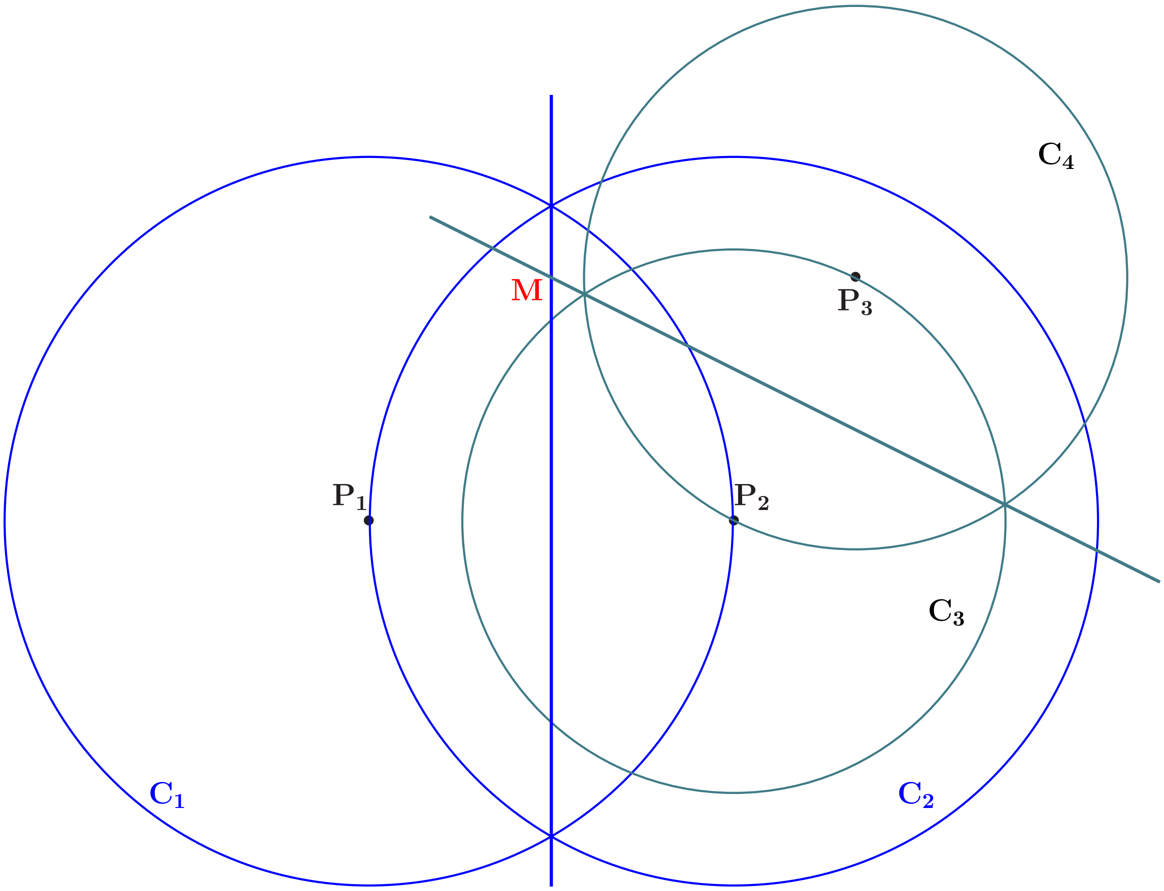

Figure 8.1: Constructing the centre point M

$PairOfPoints$, is a set of at most two points. Thus also the case of a one-element set of points and also the empty set of points is included.

Next, we have to decide which geometric operations we assume to be available to determine the point M? Roughly speaking, we will allow all operations that can be effected with compass and ruler. More precisely the vocabulary $\Sigma_{A_{3C}}$ consists of

$$\begin{array}{ll}
circle & : Point \times Point \times Point \to Circle \\
intersectL & : Line \times Line \to Point \\
line & : PairOfPoints \to Line \\
intersectC & : Circle \times Circle \to PairOfPoints \\
P_1 & : \to Point \\
P_2 & : \to Point \\
P_3 & : \to Point \\
M & : \to Point
\end{array}$$

For any structure $\mathcal{U} \in \mathcal{S}(A_{3C}) = (U, I)$ we define that

1. $circle(Q_1, Q_2, Q_3)$ is the circle with centre Q_1 and radius r given by the distance between Q_2 and Q_3. If $Q_2 = Q_3$ the circle is identified with the singleton set $\{Q_1\}$.
2. $intersectL(L_1, L_2)$ is the point, in which the lines L_1, L_2 intersect or $undef$ if L_1, L_2 are parallel.
3. $line(S)$ is the uniquely defined line through the points Q_1, Q_2 if $S = \{Q_1, Q_2\}$ with $Q_1 \neq Q_2$, and $undef$ otherwise.
4. $intersectC(C_1, C_2)$ is the set of points common to C_1 and C_2. If $C_1 = C_2$ then $intersectC(C_1, C_2)$ is $undef$.

A structure $\mathcal{U} \in \mathcal{S}(A_{3C}) = (U, I)$ is an initial structure of A_{3C}, i.e. $\mathcal{U} \in \mathcal{I}(A_{3C})$ if

1. If $I(P_1)$, $I(P_2)$, $I(P_3)$ are points not lying on a line.
2. $I(M)$ is undefined.

For structures $\mathcal{U} \in \mathcal{I}(A_{3C})$ the one-step transformation $\tau_{3C}(\mathcal{U}) = \mathcal{W}$, where the only change in $\mathcal{W}$ with respect to $\mathcal{U}$ is in the interpretation of the constant

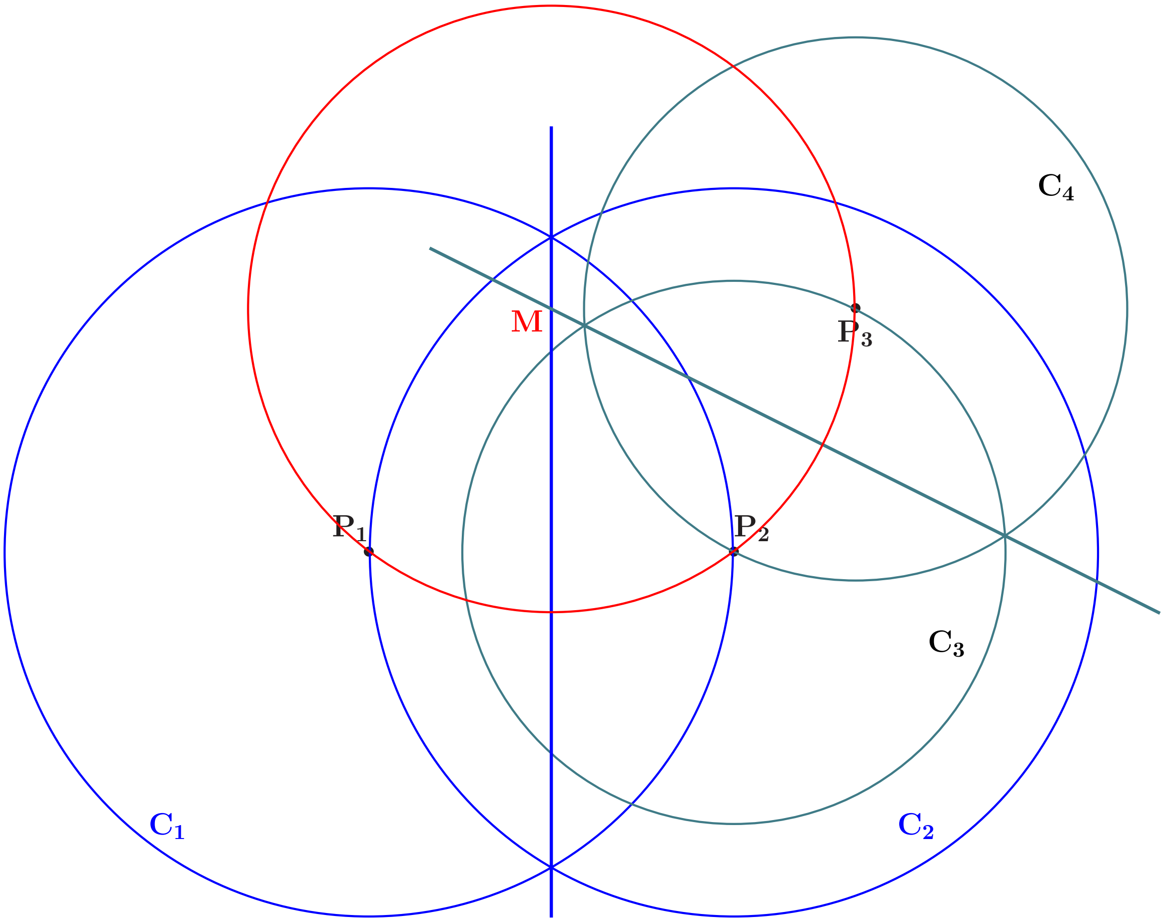

Figure 8.2: The circle touching three given points

M. This is given by:

$$I^W(M) = intersectL^U(l_1, l_2)$$

with

$$\begin{array}{ll} l_1 & = line^U(intersectC^U(C_1, C_2)) \\ l_2 & = line^U(intersectC^U(C_3, C_4)) \\ C_1 & = circle^U(P_1, P_1, P_2) \\ C_2 & = circle^U(P_2, P_1, P_2) \\ C_3 & = circle^U(P_2, P_2, P_3) \\ C_4 & = circle^U(P_3, P_2, P_3) \end{array}$$

τ_{3C} is not defined for non-initial structures. Thus runs for the algorithm A_{3C} are very simple: they start with an initial structure $\mathcal{U}_0 \in \mathcal{I}(A_{3C})$, proceed with $\mathcal{U}_1 = \tau_{3C}(\mathcal{U}_0)$, and that is the end. Some simple reasoning shows that $I^{U_1}(M)$ is defined. Thus $\tau_{3C}(\mathcal{U}_1)$ is not defined, the run is completed.

8.1.5 What Is A Single Step?

One of the constituent features of Definition 19 is the one-step transformation τ. But what exactly is *one step*? Most steps can be broken down into simpler steps. Is there a notion of atomic steps? We want to shed some light on this issue by the following example.

The vocabulary Σ_{sq} of algorithm A_{sq} contains only one sort Int and a constant symbol *const*. The typical structure in $\mathcal{S}(A_{sq})$ has as it universe U the set of integers There is no restriction on $I^{\mathcal{U}}(const)$. All of $\mathcal{S}(A_{sq})$ is obtained by taking isomorphic copies. We further stipulate $\mathcal{I}(A_{sq}) = \mathcal{S}(A_{sq})$. For $\mathcal{U} \in \mathcal{S}(A_{sq})$ the one-step transformation $\tau_{sq}(\mathcal{U}) = \mathcal{W}$ is obtained by

$$I^{\mathcal{W}}(const) = I^{\mathcal{U}}(const)^2$$

This makes sense immediately for the typical structures with universe equal to the set of integers. In the remaining cases the effect of the squaring operation is mimicked by isomorphic transfer. Is A_{sq} given by $(\mathcal{S}(A_{sq}), \mathcal{I}(A_{sq}), \tau_{sq})$ a sequential algorithm? The structure $\mathcal{U}$ with universe the set of integers and $I^{\mathcal{U}}(const) = 2.$ is certainly in $\mathcal{S}(A_{sq})$ and $\Delta(A_{sq}, \mathcal{U}) = \{(const, \langle\rangle, 4)\}$. The set of critical terms T can at most be $T = \{const\}$. This would contradict Lemma 3. Thus A_{sq} is not a sequential algorithm.

Let the vocabulary Σ_{sq}^{+} be $\Sigma_{sq} \cup \{+\}$. In structures $\mathcal{U} \in \mathcal{S}A_{sq}^{+}$ the symbol $+$ is interpreted as addition. Otherwise the algorithm A_{sq}^{+} coincides with A_{sq}. By the same argument A_{sq}^{+} still does not satisfy the requirements of a sequential algorithm. Only when we extend the vocabulary further to include also multiplication $*$ will $A_{sq}^{+,*}$ satisfy the requirement of Definition 19.

This seems to contradict our experience, that we can write an algorithm computing the square function by using addition only. The crucial point to observe here, is the fact that the notion of one-step transformation does exlude loops. Here is the correct set-up of an algorithm A_{square} computing the square function by using addition only.

The vocabulary Σ_{square} contains the two sorts Int and $Bool$ and the function symbols

$$
\begin{array}{ll}
input & : \to Int \\
output & : \to Int \\
counter & : \to Int \\
0 & : \to Int \\
+ & : Int \times Int \to Int \\
= & : Int \times Int \to Bool \\
< & : Int \times Int \to Bool \\
- & : Int \to Int \\
minus1 & : Int \to Int \\
\& & : Bool \times Bool \to Bool
\end{array}
$$

The universe of all structures $\mathcal{U}$ in $\mathcal{S}(A_{square})$ is the disjoint union of the set of all integers and the Boolean values $\{\mathbf{true}, \mathbf{false}\}$. The symbols 0, $+$, $=$, $<$, $-$, $\&$ are interpreted as usual and also $minus1$ will be as expected, i.e. $I^{\mathcal{U}}(minus1)(n) = n - 1$. $I(input)$ may be any number. For initial structures $\mathcal{U}$ we require $I^{\mathcal{U}}(output) = I^{\mathcal{U}}(counter) = 0$. For $\mathcal{U} \in \mathcal{S}(A_{square})$ the one-step transformation function $\tau_{square}(\mathcal{U})$ equals $\mathcal{W}$. The only symbols whose interrpetation may possibly change in passing from $\mathcal{U}$ to $\mathcal{W}$ are $input$, $counter$ and $output$.

$$
\begin{array}{lll}
\text{if } input < 0 & \text{then} & I^{\mathcal{W}}(input) = I^{\mathcal{U}}(-(input)) \\
\text{if } input > 0 \ \& \ counter = 0 & \text{then} & I^{\mathcal{W}}(counter) = I^{\mathcal{U}}(input) \\
\text{if } input > 0 \ \& \ counter > 0 & \text{then} & I^{\mathcal{W}}(counter) = I^{\mathcal{U}}(minus1(counter)) \\
 & & I^{\mathcal{W}}(output) = I^{\mathcal{U}}(output + input)
\end{array}
$$

It can be easily checked that for every initial structure $\mathcal{U} \in \mathcal{I}(A_{square})$ there is only one run. This run is finite and for its final state $\mathcal{W}$ we get $I^{\mathcal{W}}(output)$

is the square of $I^{\mathcal{U}}(input)$.

8.1.6 Example: A Graph Algorithm

We set out to describe an algorithm A_{G0} for testing reachability within graphs. On our way to an eventual solution we will discover some subtleties of definition 19.

The states of A_{G0} will be two-sorted first-order structures, containting the sorts *Bool* and *Node*. The universe U of any structure in $\mathcal{S}(A_{G0})$ will be the disjoint union $U = U_{Bool} \cup U_{Node}$. Where U_{Bool} always equals $\{\mathbf{true}, \mathbf{false}\}$, while U_{Node} may be any non-empty set.

The vocabulary Σ_{G0} contains the following function symbols

$$\begin{array}{ll} edge & : Node \times Node \to Bool \\ start & : \to Node \\ finish & : \to Node \\ true & : \to Bool \\ false & : \to Bool \\ reachable & : Node \to Bool \end{array}$$

$\mathcal{S}(A_{G0})$ consists of all Σ_{G0}-structures $\mathcal{U} = (U, I)$ such that $U_{Bool} = \{\mathbf{true}, \mathbf{false}\}$, $I(true) = \mathbf{true}$, $I(false) = \mathbf{false}$, and $U_{Node} \neq \emptyset$.

$\mathcal{I}(A_{G0})$ consists of all $\mathcal{U} \in \mathcal{S}(A_{G0})$ with

1. $\{n \in U_{Node} \mid I(reachable)(n) = \mathbf{true}\} = \{I(start)\}$, i.e. in an inital state the start node is the only node marked *reachable*.

Finally for $\mathcal{U} = (U, I_U) \in \mathcal{S}(A_{G0})$ the one-step transformation τ_{G0} of A_{G0} has to be defined. If $I_U(reachable(finish)) = \mathbf{true}$, then $\tau_{G0}(\mathcal{U})$ is not defined. Otherwise $\tau_{G0}(\mathcal{U}) = \mathcal{W} = (W, I_W)$ with

for all $n \in U_{Node}$ $I_W(reachable)(n) = \mathbf{true}$ iff $I_U(reachable)(n) = \mathbf{true}$ or there is $n_0 \in U_{Node}$ with $I_u(edge)(n_0, n) = \mathbf{true}$ and $I_U(reachable)(n_0) = \mathbf{true}$.

Is A_{G0} thus defined an algorithm? If yes, then there exists by the postulate 18 a set of critical terms T. The terms in T are of no particular interest to us at this moment, we are only interested in the number k of elements in T.

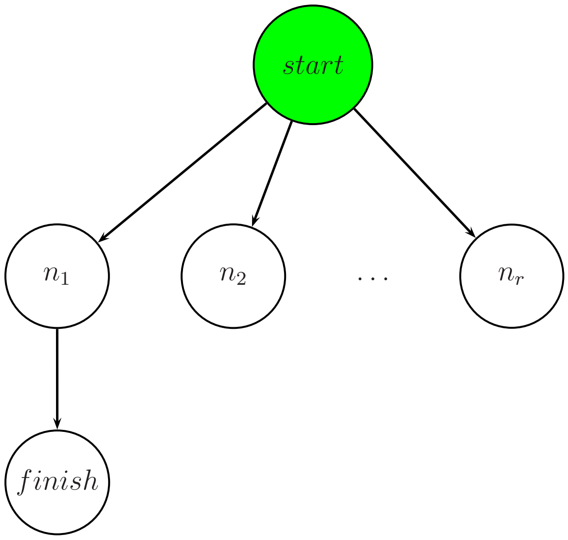

Figure 8.3: Example of a reachability problem

Look at the graph in Figure 8.3 where we have chosen r (the number of successor nodes of start) to be greater than k. Let $\mathcal{U}_0$ be the structure in $\mathcal{I}(A_{G0})$ whose graph part is given by the graph in Figure 8.3 and $\mathcal{U}_1 = \tau_{G0}(\mathcal{U}_0)$. Then $\Delta(A_{G0}, \mathcal{U}_0)$ contains at least the locations

$$\{(\mathit{reachable}, \langle n_1 \rangle, \mathbf{true}), \ldots, (\mathit{reachable}, \langle n_r \rangle, \mathbf{true})\}$$

By Lemma 3 there exists a term $t_i \in T$ such that $t_i^{\mathcal{U}_0} = n_i$ for all $1 \leq i \leq r$. Since all the nodes n_i are different, $r > k$ and there are only k elements is T, this is a contradiction. A_{G0} is not a sequential algorithm?

The problem here is that A_{G0} does too many updates in parallel.

`muss ab hier noch ergänzt werden`

8.2 ASM Programs

In the previous section we have introduced a precise and abstract notion of algorithm (see Definition 19). Part of this definition was the one-step transformation τ. There was no restriction on how τ should be defined, as long as this was done unambigously. In this section we will turn to an approach that is closer to main stream computer science. We will describe an abstract, programing language. Programs in this language go by the name of *rule* or more precisely *Abstract State Machine rule* (ASM rule, for short).

We then consider the same set-up as in Definition 12, with the only, but notable difference, that the transformation if effected by a program or ASM rule.

8.2.1 Definition

Definition 20 (ASM Rule) *Let Σ be a fixed vocabulary. We assume that Bool is a sort of Σ. A term in Σ is called a* Boolean term *if it is of sort Bool.*

1. *If ϕ is a Boolean term and R_1, R_2 are rules then*

   $$\textbf{if } \phi \textbf{ then } R_1 \textbf{ else } R_2 \textbf{ endif}$$

   is a rule,

2. *If $R_1, \ldots R_k$ are rules then*

   par
   R_1
   $\vdots$
   R_k
   endpar

   is a rule,

3. *If f is an n-place function symbol in Σ, $t_0, t_1, \ldots t_n$ are Σ-terms then*

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

   is a rule.

For any ASM rule R and any Σ-structure $\mathcal{U}$ in $\mathcal{S}$ the result of applying R in the state $\mathcal{U}$, denoted by $\tau_R(\mathcal{U})$, will be defined as $\mathcal{U} + R(\mathcal{U})$ for a set of updates $R(\mathcal{U})$.

Definition 21 (Update of Rules) *Let R be an ASM rule of vocabulary Σ and $\mathcal{U}$ a Σ-structure.*

1. *If R is of the form $f(t_1, \ldots t_n) := t_0$ then*

$$R(\mathcal{U}) = \{(f, \langle a_1, \ldots , a_n\rangle, a_0)\}$$

where $a_i = t_i^{\mathcal{U}}$. Here $(f, \bar{a}, a_0)$ is a location, see Definition 15.

2. *If R is of the form*

par
R_1
$\vdots$
R_k
endpar

then

$$R(\mathcal{U}) = R_1(\mathcal{U}) \cup \ldots \cup R_k(\mathcal{U})$$

3. *If R is of the form* **if** ϕ **then** R_1 **else** R_2 **endif** *then*

$$R(\mathcal{U}) = \begin{cases} R_1(\mathcal{U}) & \textit{if } \phi^{\mathcal{U}} = \textbf{true} \\ R_2(\mathcal{U}) & \textit{otherwise} \end{cases}$$

Definition 22 (Transformation of an ASM Rule) *Let R be an ASM rule and $\mathcal{U}$ an eligible structure then*

$$\tau_R(\mathcal{U}) = \mathcal{U} + R(\mathcal{U})$$

For an explanation of of "+", see Definition 16. If $R\mathcal{U})$ is inconsistent then $\tau_R(\mathcal{U})$ is not defined.

Definition 23 (Sequential Abstract State Machine) *A* sequential Abstract State Machine *B of vocabulary Σ is given by:*

1. *A ASM program R of vocabulary Σ,*
2. *A set $\mathcal{S}(B)$ of Σ-structures closed under isomporphism,*
3. *A subset $\mathcal{I}(B)$ of $\mathcal{S}(B)$, called the initial states of B,*

Lemma 5 *Let B be a sequential Abstract State Machine of vocabulary Σ given by $(\mathcal{S}(B), \mathcal{I}(B), R)$ then $(\mathcal{S}(B), \mathcal{I}(B), \tau_R)$ is an sequential algorithm (see Definition 19).*

Proof: Easily checked.

∎

8.2.2 Examples

As a first example let us look again at the one-step transformation τ_{square} from Subsection 8.1.5.

par
  if $(input < 0)$ **then** $input := -(input)$
  endif
  if $(input > 0 \ \& \ counter = 0)$ **then** $counter := input$
    else par
      counter := minus1(counter)
      output := output + input
      endpar
  endif
endpar

As a convention let us agree that rules, or updates that are written vertically one below the other are implicitely understood to be executed in parallel. Then the above ASM rule looks:

if $(input < 0)$ **then** $input := -(input)$ **endif**
if $(input > 0 \ \& \ counter = 0)$ **then** $counter := input$
  else $counter := minus1(counter)$
    $output := output + input$
endif

8.2.3 Universality of Abstract State Machines

The purpose of this subsection is to prove:

Theorem 6 *Let A be a sequential algorithm of vocabulary Σ_A given by $(\mathcal{S}(A), \mathcal{I}(A), \tau_A)$ then there is a ASM rule R of vocabulary Σ_A such that τ_R coincides with τ_A on $\mathcal{S}(A)$, i.e.*

$$(\mathcal{S}(A), \mathcal{I}(A), \tau_A) = (\mathcal{S}(A), \mathcal{I}(A), \tau_R^*)$$

if τ_R^* denotes the restriction of τ_R to $\mathcal{S}(A)$.

This theorem has first been formulated and proved in [Gurevich, 2000].

Lemma 7 *For any state X of A there is a ASM rule R^X such that $R^X = \Delta(X)$.*

Proof: This is easy. Assume $\Delta(X) = \{(f_i, \langle a_1^i, \ldots, a_{n_i}^i \rangle, b^i) \mid i = 1 \ldots k\}$ By the accessability lemma 3 there are terms s_j^i and t^i such that $(s_j^i)^X = a_j^i$ and $(t^i)^X = b^i$. Thus R^X:

par
$f_1(s_1^1, \ldots, s_{n_1}^1) := t^1$
$\vdots$
$\vdots$
$f_k(s_1^k, \ldots, s_{n_k}^k) := t^k$
endpar

satisfies $R^X(X) = \Delta(X)$

∎

Note, that $R^X(Y) = \Delta(Y)$ does not hold for arbitrary Y.

Definition 24 (T-similar states) *Let T be a set of ground terms. Two states X,Y are called T-similar if for ayn pair $t_1, t_2 \in T$*

$$t_1^X = t_2^X \Leftrightarrow t_1^Y = t_2^Y$$

Lemma 8 *If X, Y are T_A-similar states of the algorithm A, then*

$$R^X(Y) = \Delta(Y)$$

Proof First assume that X and Y are disjoint. Let Z be the structure isomorphic to Y, where the universe of Z is obtained by replacing every element t^Y by t^X for all $t \in T$. By the assumption of the lemma the function $F : Y \rightarrow Z$:

$$F(y) = \begin{cases} y & \text{if } y \neq t^Y \text{ for all } t \in T \\ t^X & \text{if } y = t^Y \end{cases}$$

is surjective and injective and may thus be extended to an isomorphism from Y onto Z, as intended. Since for all $t \in T$ we have $t^X = t^Z$ we get by the defining property of the set of critical terms $\Delta(Z) = \Delta(X) = R^X(X)$. From this we see immediately $\Delta(Y) = R^X(Y)$.

∎