Department of Computer Science

Computers and Logic Research Group

University College of Swansea

Memorandum No. 17: April 1971

These are informal working papers

PROGRAM SIMULATION:

AN EXTENDED FORMAL NOTION

(A sequel to Memo 14 in this series)

R. Milner

This work was carried out under a grant from the Science Research Council

PROGRAM SIMULATION :

AN EXTENDED FORMAL NOTION

(A sequel to Memo 14 in this series)

Sections . 1. Introduction. 2. Simulation : an extended formal treatment. 3. Application to Flowchart programs. 4. Partitioned simulation .

Robin Milner

---

1. INTRODUCTION

In Memo 14 we defined a notion of simulation between programs which was confined to programs having the same 'shape', in the sense that their flowcharts are identical when the contents of the boxes are removed. Thus we concentrated on simulation between programs with different data representations.

We now want to relax the restriction of 'same shape', and so extend the notion of simulation. There is a clear intuitive sense in which the programs of figure 1 do the same job in the same way; we might say loosely that they only differ in their methods of controlling the 'important' part of the computation (that concerned with x).

So we want to say that they simulate each other; but we cannot base a definition of simulation on such a vague idea as, in a given program, separating those variables and instructions concerned with control from those which 'do the real work', because this separation is often a difficult or arbitrary matter.

It turns out that there is a natural notion of simulation with the following properties:

(i) It applies in situations in which the programs differ only in their data representations — i.e. it includes the notion of Memo 14 as a special case.

(ii) It applies in situations, such as in figure 1, where the programs do the same 'real work' in the same sequence but control it differently.

(iii) It applies in situations which combine (i) and (ii)

(iv) It is not restricted to non-recursive programs. Indeed simulation may hold between programs which use different evaluation mechanisms — an example is between a recursive and an iterative realisation of the factorial function.

Many proofs below are omitted: they are all fairly easy.

2. SIMULATION : AN EXTENDED FORMAL TREATMENT

Definition A program $\mathcal{A}$ is a quadruple $\langle D_{in}, D_{comp}, D_{out}, F \rangle$ where $D_{in}, D_{comp}, D_{out}$ are disjoint domains and

$$F : D \to D$$

is a total function (where $D = D_{in} \cup D_{comp} \cup D_{out}$) with the restrictions

(i) $F(D_{in} \cup D_{comp}) \subseteq D_{comp} \cup D_{out}$

(ii) The restriction of F to D_{out} is the identity on D_{out} - i.e. $F \restriction D_{out} = I_{D_{out}}$.

We call $D_{in}, D_{comp}, D_{out}$ the input, computation and output domains of $\mathcal{A}$.

Remarks. Conditions (i) and (ii) ensure that, starting with a member of D_{in} and applying F repeatedly we get either an infinite sequence in D_{comp}, or a finite sequence in D_{comp} followed by a single repeated member of D_{out}. We have (ii) merely to keep F total, which the following theory needs.

The apparently gratuitous novelty of this definition of a program is justified by the succinctness with which simulation properties may be stated and proved.

Why must the domains be disjoint? What about a program which takes an integer in and gives one out? In

this case one might argue that $D_{in} = D_{out} = \{\text{integers}\}$. But we get into no trouble having two formally disjoint domains with, for example, an injection or bijection existing between them. One can even claim that D_{in}, D_{out} should be disjoint; integers go in on punch cards and come out on the line printer — or perhaps they go in on the card reader and come out on the card punch. We are concerned with a level of abstraction lower than that in which a program denotes a function from integers to integers.

Definition. A computation sequence of $\mathcal{A}$ is either

$d_0, d_1, \ldots\ldots\ldots$ where $d_0 \in D_{in}$, otherwise $d_i \in D_{comp}$, or

$d_0, d_1, \ldots\ldots d_k$ where $d_0 \in D_{in}$, $d_1, d_2 \cdots d_{k-1} \in D_{comp}$, and $d_k \in D_{out}$

provided that everywhere $d_{i+1} = F(d_i)$

Definition A program $\mathcal{A}$ determines its associated partial function

$$\hat{\mathcal{A}} : D_{in} \to D_{out}$$

in an obvious way.

Remark. In the case of non-recursive (flowchart) programs we have $D_{comp} = N \times E$ where N is the finite set of nodes in the flowchart and E is the set of possible state-vector values;

in the case of recursive programs N would be the infinite set of possible states of a pushdown store.

Before dealing with simulation, let us look at the results of Floyd and Manna in this algebraic setting.

Definitions $\mathcal{A}$ is <u>partially correct</u> w.r.t $S_{in} \subseteq D_{in}$, $S_{out} \subseteq D_{out}$ if $\hat{\mathcal{A}}(S_{in}) \subseteq S_{out}$.

$\mathcal{A}$ is <u>totally correct</u> w.r.t. S_{in}, S_{out} if in addition $\hat{\mathcal{A}}$ is total on S_{in}.

S is a <u>verification</u> of $\mathcal{A}$ if $S \subseteq D$ and $F(S) \subseteq S$.

We now have three theorems.

<u>Theorem</u> For $S_{in} \subseteq D_{in}$, $S_{out} \subseteq D_{out}$

$$[\mathcal{A} \text{ partially correct w.r.t } S_{in}, S_{out}] \iff [\text{There is a verification } S_{in} \cup S_{comp} \cup S_{out} \text{ of } \mathcal{A}, \text{ where } S_{comp} \subseteq D_{comp}]$$ ∎

<u>Theorem</u> For $S_{in} \subseteq D_{in}$

$$[\mathcal{A} \text{ total on } S_{in}] \iff [\text{For every verification } S'_{in} \cup S'_{comp} \cup S'_{out} \text{ of } \mathcal{A}, \; S_{in} \cap S'_{in} \neq \emptyset \Rightarrow S'_{out} \neq \emptyset]$$ ∎

Theorem For $S_{in} \subseteq D_{in}$, $S_{out} \subseteq D_{out}$

$[\mathcal{A}$ totally coweer w.r.t $S_{in}, S_{out}] \iff [$For every verification $S'_{in} \cup S'_{comp} \cup S'_{out}$ of $\mathcal{A}$, $S_{in} \cap S'_{in} \neq \emptyset \Rightarrow S_{out} \cap S'_{out} \neq \emptyset]$ ∎

We now define two types of simulation between the programs $\mathcal{A} = \langle D_{in}, D_{comp}, D_{out}, F \rangle$ and $\mathcal{A}' = \langle D'_{in}, D'_{comp}, D'_{out}, F' \rangle$

Definition. Let $R \subseteq D \times D'$. Then R is a weak simulation of $\mathcal{A}$ by $\mathcal{A}'$ if (i) $R \subseteq D_{in} \times D'_{in} \cup D_{comp} \times D'_{comp} \cup D_{out} \times D'_{out}$

(ii) $RF' \subseteq FR$

(We denote relational composition by juxtaposition, and for a function, e.g. F, we confuse it with the relation $\{\langle d_1, d_2 \rangle \mid d_2 = F(d_1)\}$)

Now denote $R \cap (D_{in} \times D'_{in})$ by R_{in}, and R_{comp}, R_{out} similarly, so that $R = R_{in} \cup R_{comp} \cup R_{out}$ and these parts are disjoint. (R_{out} was inverted in Memo 14).

Theorem. If R is a weak simulation of $\mathcal{A}$ by $\mathcal{A}'$ then

(i) $R_{in} \hat{\mathcal{A}}' \subseteq \hat{\mathcal{A}} R_{out}$

(ii) R^{-1} is a weak simulation of $\mathcal{A}'$ by $\mathcal{A}$

(iii) $R_{in}^{-1} \hat{\mathcal{A}} \subseteq \hat{\mathcal{A}}' R_{out}^{-1}$ ∎

Remarks Part (i) of the Theorem says that the diagram

$$\begin{array}{ccc} D_{in} & \xrightarrow{\hat{A}} & D_{out} \\ R_{in}\downarrow & & \downarrow R_{out} \\ D'_{in} & \xrightarrow[\hat{A}']{} & D'_{out} \end{array}$$

Semi-commutes (i.e. we have $\subseteq$ not $=$). Now if we wish to be able to use A' to do the job of A we need more than this: we need the following to commute

$$\begin{array}{ccc} D_{in} & \xrightarrow{\hat{A}} & D_{out} \\ R_{in}\downarrow & & \uparrow R_{out}^{-1} \\ D'_{in} & \xrightarrow{\hat{A}'} & D'_{out} \end{array}$$

(note the changed arrow)

i.e. we require $\hat{A} = R_{in}\hat{A}'R_{out}^{-1}$. For this it is sufficient to require that R is a strong simulation of A by A', where

Definition. A weak simulation R of A by A' is a strong simulation if in addition R_{in}, R_{out}^{-1} are total and single valued.

Remark R^{-1} is not necessarily a strong simulation of A' by A.

Theorem If R is a strong simulation of A by A' then $\hat{A} = R_{in}\hat{A}'R_{out}^{-1}$ - i.e. A covers A' w.r.t. R_{in}, R_{out}. ∎

(The proof requires only that R_{in} is total and R_{out}^{-1} single valued)

3. APPLICATION TO FLOWCHART PROGRAMS

Suppose now we have given a flowchart program, with input domain D_{in}, state vector domain E, output domain D_{out}, perhaps with a single entry node and single exit node, and node-set N, and given also an input function $f_{in}: D_{in} \to E$ and $f_{out}: E \to D_{out}$. Then it is a simple matter to formalize it as a program according to our definition, with $D_{comp} = N \times E$ and $D = D_{in} \cup D_{comp} \cup D_{out}$, and $F: D \to D$ defined in terms of the tests and assignments in the boxes and also f_{in}, f_{out}.

Alternatively, we may formalize the same flowchart program by selecting a subset $M \subseteq N$ so that every cycle in the flowchart contains a member of M (we call such an M a <u>cycle-breaking</u> set) and define D_{comp} instead as $M \times E$, and again $D = D_{in} \cup D_{comp} \cup D_{out}$. The cycle breaking property ensures that F may again be defined as a <u>total</u> function $D \to D$.

Now suppose in $\mathcal{A}$ and $\mathcal{A}'$ (as previously defined) we have $D_{comp} = M \times E$, $D'_{comp} = M' \times E'$ — assuming that $\mathcal{A}$, $\mathcal{A}'$ are flowchart programs not necessarily of the same shape. If R is a simulation of $\mathcal{A}$ by $\mathcal{A}'$, we have $R_{comp} \subseteq (M \times E) \times (M' \times E')$ and to exhibit R_{comp}

it is sufficient to exhibit $R_{mm'}$ for each $m \in M, m' \in M'$, where

$$R_{mm'} = \{ \langle e,e' \rangle \mid \langle \langle m,e \rangle, \langle m',e' \rangle \rangle \in R_{comp} \} .$$

Then given also R_{in}, R_{out} and of course F, F' it is a routine matter to prove $RF' \subseteq FR$. We outline an example, which indicates that the $R_{mm'}$ may often be arrived at intuitively.

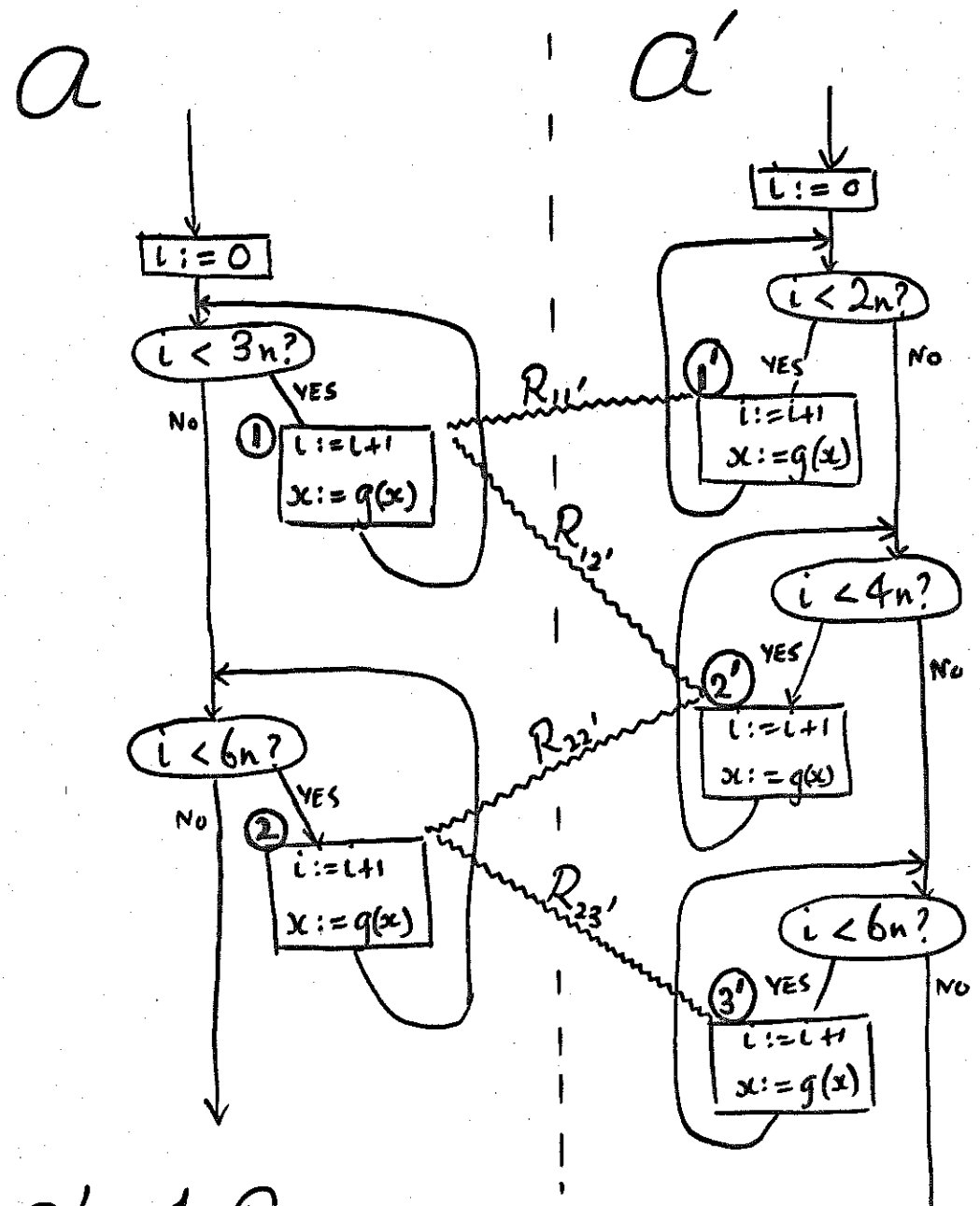
In Figure 1, $\mathcal{I}$ and $\mathcal{R}$ denote the integers and the reals. We assume that inputs to each program are pairs $\langle n,x \rangle$, their state vectors are triples $\langle i,n,x \rangle$ and that only x is output.

The node set $\{1,2\}$ has been chosen to formalise Q, and $\{1',2',3'\}$ to formalise Q'.

A simulation R is postulated; it is exhibited by giving R_{in}, R_{out} and the $R_{mm'}$.

For example, we may regard $R_{12'}$ as containing all state-vector-pairs attained at the node-pair $\langle 1,2' \rangle$ when Q, Q' are obeyed synchronously, starting from an input-pair in R_{in} (i.e. in this example from identical inputs). $R_{13'}$ is empty, because the node-pair $\langle 1,3' \rangle$ is never reached.

To prove $RF' \subseteq FR$ is equivalent to showing $(\forall d,d')\ d R d' \Rightarrow F(d)\, R\, F(d')$



$D_{in} = D'_{in} = \mathcal{I} \times \mathcal{R}$;

$E = E' = \mathcal{I} \times \mathcal{I} \times \mathcal{R}$; $D_{comp} = \{1,2\} \times E$; $D'_{comp} = \{1',2',3'\} \times E'$

$D_{out} = D'_{out} = \mathcal{R}$.

$R_{in} = I_{D_{in}}$; $R_{out} = I_{D_{out}}$;

$R_{13'} = R_{21'} = \emptyset$;

$$R_{11'} = \{ \langle \langle i,n,x \rangle \langle i,n,x \rangle \rangle \mid i < 2n \} \quad ;$$

$$R_{12'} = \{ \quad " \quad " \quad \mid 2n \le i < 3n \} \; ;$$

$$R_{22'} = \{ \quad " \quad " \quad \mid 3n \le i < 4n \} \; ;$$

$$R_{23'} = \{ \quad " \quad " \quad \mid 4n \le i < 6n \} \; .$$

Figure 1

and this must be done by cases:

Case 1: $\langle d,d'\rangle \in R_{in}$

Case 2: $\langle d,d'\rangle \in R_{out}$

Case 3: $\langle d,d'\rangle = \langle\langle m,e\rangle\langle m',e'\rangle\rangle$ where $\langle e,e'\rangle \in R_{mm'}$

(there is a subcase for each possible $\langle m,m'\rangle$)

using the definitions of F, F'.

Now since R is a strong simulation, and in fact R_{in}, R_{out} are identities, the last theorem of section 2 entitles us to conclude $\hat{a} = \hat{a}'$.

As with Floyd's technique for proving programs partially correct, once the $R_{mm'}$ have been guessed it should be - or should soon be - within the power of mechanical theorem provers to verify $RF' \subseteq FR$.

The above example has programs with identical data representation, but different control; however the technique works equally when the data representations are different — but then the $R_{mm'}$ will not be subidentities, as they are in the example. If control is the same in both programs — i.e. they have the same shape, and follow the same execution path on corresponding inputs — then we are back with the case studied in Memo 14, and will expect to have $R_{mm'} = \phi$, $m \neq m'$.

4. PARTITIONED SIMULATION

In Memo 14 the Simulation Theorem stated that, for given R_{in}, R_{out}^{-1} both single-valued and total, a necessary and sufficient condition for the existence of a simulation $R_{in} \cup R_{comp} \cup R_{out}$ of $\mathcal{A}$ by $\mathcal{A}'$ is that the following are satisfied:

(i) $\mathcal{A}'$ covers $\mathcal{A}$ w.r.t R_{in}, R_{out}, i.e. $\hat{\mathcal{A}} = R_{in} \hat{\mathcal{A}}' R_{out}^{-1}$

(ii) The computation sequences of $\mathcal{A}, \mathcal{A}'$ from inputs x, x' follow the same (node) path provided $x R_{in} x'$.

Condition (ii) had meaning because simulation was there defined only for programs with identical flowcharts. It has no meaning yet for our present definitions of program and simulation, but with the help of <u>partition pairs</u> (defined below) we get a general result of which the former Simulation Theorem is a special case.

<u>Definition</u>. If J is any indexing set and $\pi_J = \{C_j \mid j \in J\}$, $\pi'_J = \{C'_j \mid j \in J\}$ are partitions of D_{comp}, D'_{comp} respectively, then $\langle \pi_J, \pi'_J \rangle$ is a <u>partition pair</u> for D_{comp}, D'_{comp}.

(Of course any two domains can have a partition pair; we are only concerned with computation domains)

Definition. Computation sequences $d_0, d_1, \ldots$ in $\mathcal{A}$ and $d_0', d_1', \ldots$ in $\mathcal{A}'$ agree for $\langle \pi_J, \pi_J' \rangle$ if

$$\forall j \, \forall i \left[d_i \in C_j \iff d_i' \in C_j' \right]$$

Definition A simulation R respects $\langle \pi_J, \pi_J' \rangle$ if

$$R_{comp} \subseteq \bigcup_{j \in J} (C_j \times C_j')$$

We can now state two Theorems, one concerning weak simulation, the other strong simulation.

Weak simulation Theorem. Given $R_{in} \subseteq D_{in} \times D_{in}'$ the following two statements are equivalent (assume also a partition pair $\langle \pi, \pi' \rangle$ of D_{comp}, D_{comp}'):

(a) All computation sequences $\{d_i\}$ of $\mathcal{A}$, $\{d_i'\}$ of $\mathcal{A}'$ for which $d_0 R_{in} d_0'$, agree for $\langle \pi, \pi' \rangle$.

(b) There is a weak simulation $R_{in} \cup R_{comp} \cup R_{out}$ of $\mathcal{A}$ by $\mathcal{A}'$ which respects $\langle \pi, \pi' \rangle$. ∎

Strong simulation Theorem. Given $R_{in} \subseteq D_{in} \times D_{in}'$, $R_{out}' \subseteq D_{out}' \times D_{out}$ both single valued and total, and $\langle \pi, \pi' \rangle$ as in the previous Theorem, the following two statements are equivalent:

(a) A' covers A w.r.t. R_{in}, R_{out}, and all computation sequences $\{d_i\}$, $\{d_i'\}$ of A, A', for which $d_0 R_{in} d_0'$, agree for $\langle \bar{\pi}, \bar{\pi}' \rangle$

(b) There is a strong simulation $R_{in} \cup R_{comp} \cup R_{out}$ of A by A' which respects $\langle \bar{\pi}, \bar{\pi}' \rangle$. □

(The proof requires R_{in} single valued and total, R_{out}^{-1} single valued but not necessarily total)

Corollary. Suppose $D_{comp} = N \times E$, $D'_{comp} = N \times E'$, and that N is a finite set of nodes. Thus A, A' have the same node set, and in particular their flowcharts may be the same shape. Let partition $\bar{\pi}_N = \{\{n\} \times E \mid n \in N\}$ and $\bar{\pi}'_N = \{\{n\} \times E' \mid n \in N\}$. Then for R_{in}, R_{out}^{-1} single valued and total the following are equivalent:

(a) A' covers A w.r.t R_{in}, R_{out}, and the computation sequences $\{d_i\}$ in A, $\{d_i'\}$ in A' follow the same node path provided $d_0 R_{in} d_0'$.

(b) There is a strong simulation $R_{in} \cup R_{comp} \cup R_{out}$ of A by A' which respects $\langle \bar{\pi}_N, \bar{\pi}'_N \rangle$. □

This corollary is effectively the Simulation Theorem of Memo 14, and there is a corresponding corollary to the Weak Simulation Theorem

above, which we omit.

Acknowledgments.

I am greatly indebted to Peter Landin, Rod Burstall and John Laski for discussions on this topic, mainly in relation to generalizing the notion of simulation. The algebraic approach to topics in programming stems largely from Landin: see his paper "A Program Machine Symmetric Automata Theory", pp99-120, Machine Intelligence 5 (1969). For other references consult Memo 14.