INTERCELLAS
an interactive cellular space simulation language

By T. LEGENDI

Abstract

Intercellas is an integral member of a family of languages for cellular space simulation and programming. It has been designed for interactive use on mini-computers and gives possibilities for simulation of heterogeneus spaces, too.

I. Design goals

The main goals of the research and the reasons to define a new family of cellular space simulation and programming languages [8, 10, 12] rather than using the existing ones [1, 2, 3, 4] — are described in detail in [11].

The CELLAS cellular space simulation and programming language family has been designed for medium size computers and mainly for use under batch operating systems. In many cases it seems more favourable and economic to use a smaller but interactive system.

The instruction set of INTERCELLAS is compatible with CELLAS except dinamic program editing and interrupt possibilities.

It is natural that after detecting syntactic or semantic errors control is passed to the programmer.

Some instructions of the language are more simple, than their equivalents in CELLAS, but combined with interactive facilities. For example: instruction ON, that monitors continuosly the space, in the original language has wider condition description part and has action part, too. In INTERCELLAS it has no action part, preprogrammed automatic reaction is inpossible, the only action is that control is passed to the programmer at the consol (display).

The instructions of the language are command-type. They are interpreted and executed as they enter the processor. The structure of the language is very simple and natural:

a) to begin a simulation the programmer first needs tools to define topology, neighbourhood, transition function(s), size of the space and initial configuration should be formed;

b) control of the simulation steps is the central goal of the language;
c) the display of the results should be flexible to support obtaining as much information as possible but in compact form through controlling the time, mode, origin and destination of the results to display;
d) interactive intervention into the simulation process;
e) other utilities;
f) as minicomputer configurations are very different (peripherals, mass storage, core memory) and the structure of the language is simple, it is very natural to associate a system generation feature (actual peripherals, instructions, limits for tables — including transition function table, the maximum size of the cellular space and other internal tables, etc — may determine actual processors).

In the following the instruction set of the language will be briefly discussed and shown that it meets the above basic requirements.

The instruction format is simple, assembly type. The instructions begin with instruction code (may be abbreviated or full) which is followed by parameters. The instructions may be labeled in the usual way.

II. Groups of the instruction set

1. Definition of the space. The *topology* of the space may be defined through assigning to each cell its neighbours' relative coordinates.

Transition functions may be defined by a list of terms, by tables, by definition of terms in a tree form, or by feature definition and new state assignment instructions.

Examples: *Table form 1* 0110 1001 1001 0110 ... (table of summa mod 2). Here $n_0+2n_1+4n_2+\ldots$ defines an address in the table where the new state should be found ($n_0, n_1, n_2 \ldots$ are the current states of the cell and its neighbours).

Table form 2 (Summa mod 4)

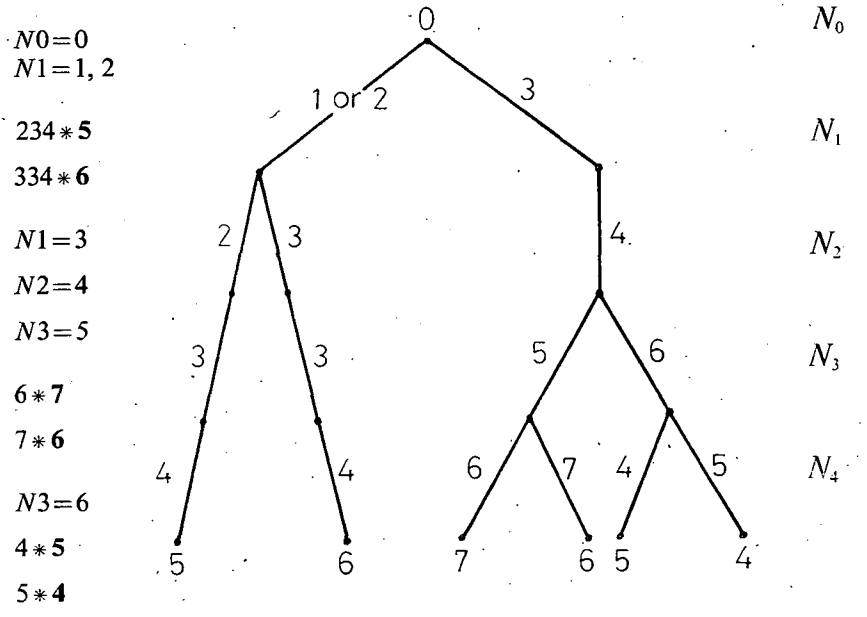
T	0	1	2	3
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

Here $T(T(T(T(n_0, n_1), n_2), n_3), n_4)$ assigns the next state.

List of *terms*

old state	neighbours states	new state
0	1234	5
0	1334	6
0	2234	5
0	2334	6
0	3456	7
0	3457	6
0	3464	5
0	3465	4

Tree form (for the above terms)



Another *term form:*

0	−0+1+2−3−4	*	5
old state	neighbourhood description		new state

where in the simplest case + means there is at least one neighbour in the state following the plus sign, and — means that there is no neighbour in the state following the minus sign.

A more sophisticated interpretation may be used by *feature definition instructions*, their execution assigns a feature to i and after it, + i means that the neighbourhood has the ith feature, −i means the opposite.

Examples for possible features:

a) the third neighbour is in state 3 or
the third neighbour is in state 4 or
the second neighbour is in state 3
b) there are no neighbours in state 2
c) the number of neighbours in state 2 and state 4 is even
or the number of neighbours in state 3 is 2

Transtion functions may be read from files or libraries, too. (See I/0 instructions).

Dimension and actual *size* of the space may be directly defined, a space may have *dummy* cells on the *boundary* or may be *closed* in form of a circle in one dimension (or *torus* in two dimension).

To *different parts* of the space *different* (predefined) *transition functions* may be assigned.

Initial configuration. Initial configurations may be defined by a set of simple geometrical instructions: rectangles may be filled with the same value (state), lines (of the same value) may be specified, configurations may be copied from parts of the space to other parts of it and configurations may be shifted from outside of the space. I/0 instructions (see below) may also define configurations in the space.

2. Input-output. Under I/0 we mean here the flow of main (characteristic) data structures e.g. transition functions and configurations.

The instructions control the flow (and implicitly the conversion) of data among files, core memory and libraries.

On a *file* data are in a usual line form and may be accessed only sequentially.

In the *memory* data are in internal representation.

In a *library* data are in compressed formats, they may be accessed in associative way.

Simple library handling instructions are also included (library initiation/purge, listing, deleting objects from the library etc.).

3. Simulation. The central instruction of the language effects execution of n steps of simulation (the cellular space should be defined prior to the execution of the simulation). As the simulation is sequential and therefore slow a look ahead algorithm is applied to speed up processing. Cells are grouped in two classes:

a) *closed cells* (neither the cell nor its neighbour cells had changed their previous states during the last step of simulation)

b) *open cells* (either the cell itself or any of its neighbour cells had changed its state during the last step of simulation)

Naturally, the transition function is computed only for open cells. The cells are made open or closed during each simulation step. The presumed status is closed and it is made open only if a change occurs in the state of the cell or in the states of its neighbours. The status is stored in two independent flag-bits (open/closed, next open/next closed) which alternate during the consecutive simulation steps. The main characteristic of the algorithm is that no operation is performed on the closed cells—neither on their states nor on their flag—bits.

4. Display of the results. It is possible to define

a) *when* — in which steps of the simulation

b) *from where* — from what parts of the cellular space

c) *how* — which characters correspond to the states of the cells (conversion)

to display the results. Steps for display are indicated by a set of stored display/do not display instructions. (They are stored in the order of execution.) They effect in parallel — each of them defines a set of steps (non-negative integers) in form $a+bx+c^y$ $x, y=1, 2, 3, \ldots$ If an actual simulation step does not fall in any set or it is a member of at least one do not display set — no display will take place. The result will be displayed if the actual step occurs (only) in a display set.

For example

	a	b	c
DISPLAY	1	0	0
DISPLAY	2	0	0
DISPLAY	0	0	3
DO NOT DISPLAY	27		

Results will be displayed in the first, second and each $3n$, $n=1, 2, 3, \ldots$ steps excluding 27. (1, 2, 3, 9, 81, ...)

DISPLAY	0	1	0
DO NOT DISPLAY	0	3	0

Results will be displayed in each step excluding each third step. (1, 2, 4, 5, 7, 8, 10, ...)

Destination of the results may also be programmed, this is treated in the next section.

5. Flow of information. It is possible to designate files (including peripherial equipments) for
a) the program input file,
b) program output files,
c) result output files.

The program input file should contain an INTERCELLAS program or part of it.

The program output file(s) will contain the executed and (during execution) skipped instructions. This file will contain only syntactically correct instructions.

The result output file(s) will contain the result of the run.

All these files may be dinamically designated. (For example the source program may be read from two or more files).

Files may be designated for more than one purpose (in a meaningful way).

A natural way is to read program from the card reader or the console and to print the program and the results in an intermixed form e.g. each instruction is followed by its result (if any).

Another possibility is to make selective output: for example
a) to print results and send them to a mass storage file, not to print program;
b) to read program from console, to punch program, to print results and program and so forth.

6. Flow of control. Normally instructions are executed sequentially. It is possible to skip instructions conditionally or unconditionally with an optional input file change.

(There are two main reasons to combine skipping with input file change:
a) skip instructions typed in at the main periphery have no meaning without a change of the input file;
b) in some situations at the time of a change of the input file skipping may be needed. For example a prepared program on cards or tape during its run passes control to the programmer. After executing the needed intervention the programmer wants to pass control back to the program — but may be not to the point where it was interrupted).

It is possible to stop the work of the processor finally, or to begin a new simulation.

Continuos monitoring of the cellular space against simple conditions (whether the state of the (i, j) cell is S) may be programmed. When the condition is met, control is passed to the programmer. Conditions may be cleared, too.

7. Interactivity. There is a simple editing possibility: characters are specified for deleting consecutive characters or parameters or a whole line typed in previously. (The read process is character oriented rather then line oriented.)

A longer (many steps) simulation process may be interrupted by sending a special interrupt character. The actual simulation step is finished and control is passed to the main periphery. After the execution of arbitrary instructions it is possible to return to the previous activity and input periphery. Two special characters ensure the return. One of them selects to continue the interrupted simulation, while the other skips the remaining steps and execution continues by interpreting the next instruction on the previous input periphery.

Interrupt may be used in case of any problem with an actual (when it is not the main) periphery — the change to the input periphery ensures intervention preserving the previous state of the system.

8. System generation instructions. The list of *actual peripherals* defines the needed/unneeded handlers and tables.

The *main peripheral* (which should be a read/write peripheral, it has priority and interrupt may me initiated only from it) may be designated. The actual *set of instructions* may be defined including the definition of the names of the instructions. In connection with this there is an implicit possibility for *selecting* transition function evaluating *table search procedures* and this selection may be done explicitly, too.

There is a possibility to limit some explicit and implicit data structures namely the *maximum size* of the cellular *space*, and *transition function* tables; the *maximum number* of parallel *ON conditions*, *DISPLAY/DO NOT DISPLAY conditions* etc.

Error messages may be defined too.

An automatic user's manual generation and autotest generation will also be incorporated into this group of instructions which is implemented in FORTRAN IV as cross-software for the simulator.

III. Implementation, emulators

INTERCELLAS has been implemented on minicomputers CII—10010 (subset only), TPA (equivalent to PDP—8) and R—10 (equivalent to MITRA—15). The processors are coded in assembly and FORTRAN—IV languages.

For speeding up the simulation special firmware (cellular space emulator) has been designed for 2 state and 16 state spaces.

The main goal of the project is to design cellular processors: emulators will serve partly as their working models. INTERCELLAS has been designed as a software tool for simulation, which may be interpreted as machine code level programming of cellular processors (with added utilities which are important and useful but do not increase the machine code level). In this way INTERCELLAS *may be used for testing cellular processors and developing higher level languages for them.*

RESEARCH GROUP ON MATHEMATICAL LOGIC
AND THEORY OF AUTOMATA OF THE
HUNGARIAN ACADEMY OF SCIENCES
H-6720 SZEGED, HUNGARY
SOMOGYI U. 7.

References

[1] CODD, E. F., *Cellular automata*, Academic Press, Inc. New York, London, 1968.
[2] VOLLMAR, R., Über einen Interpretierer zur Simulation Zelluraren Automaten, *Angewandte Informatik*, v. 6, 1973, pp. 249—256.
[3] BRENDER, R. F., *A programming system for the simulation of cellular spaces*, Ph. D. Thesis, The University of Michigan, Ann Arbor, 1970.
[4] BAKER, R., G. T. HERMAN, CELIA — a cellular linear iterative array simulator, *Proceedings of the Fourth Conference on Applications of Simulation*, 1970, pp. 64—73.
[5] WU-HUNG LIU, CELIA, *Users manual*, Dept. of Computer Science, State University of New York at Buffalo, October, 1972.
[6] *Cellular spaces, homogeneous structures*, Institute of Mathematical Machines, Warsaw, 1973 (in Russian).
[7] LEGENDI, T., Simulation and synthesis of cellular automata, *Conference on Programming Systems'75*, Szeged, 1975, pp. 210—217 (in Hungarian).
[8] LEGENDI, T., Simulation of cellular automata, the simulation language CELLAS, *Conference on Simulation in medical, technical and economy sciences*, Pécs, 1975, pp. 100—106 (in Hungarian).
[9] CZIBIK, I., T. LEGENDI, User's manual CELLAS 1.0, 1976 (in Hungarian).
[10] LEGENDI, T., GY. HEGEDŰS, L. PÁLVÖLGYI, User's manual INTERCELLAS, 1976 (in Hungarian).
[11] LEGENDI, T., Cellprocessors in computer architecture, *Computational Linguistics and Computer Languages XI.*, 1976, pp. 147—167.
[12] LEGENDI, T., TRANSCELL — a cellular space transition function definition and minimization language, to appear in *Computational Linguistics and Computer Languages XII.*

(Received Oct. 13, 1976)