CELL-DEVS WITH EXPLICIT DELAYS: PARALLEL SIMULATION

Gabriel A. Wainer Daniel Rodríguez
{gabrielw, drodrigu}@dc.uba.ar

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
(1428) Pabellón I. Ciudad Universitaria.
Buenos Aires. Argentina.

Norbert Giambiasi
Norbert.Giambiasi@ iuspim.u-3mrs.fr

DIAM-IUSPIM
Av. Escadrille Normandie Niemen
13397 Marseille
Cedex 20 France

Abstract

This work describes some of the extensions included in a tool to study, model and simulate cellular models. The environment is based on the DEVS and Cell-DEVS paradigms. Cell based systems can be built using a specification language, allowing reductions in the development, checking and maintenance times of the components. A mapping between the simulation mechanism and asynchronous PDES techniques is also presented, permitting to outline the implementation of executable models in parallel.

1. INTRODUCTION

In [Zei76], the **DEVS** (Discrete EVents Systems specifications) formalism was proposed to model discrete events systems. A DEVS model is built using a set of behavioral models called **Atomic**, which can be combined to form **Coupled** ones. **Cell-DEVS** [WG98] is a paradigm that has extended the DEVS formalism, allowing the implementation of cellular models. In this formalism, each cell is defined as an atomic model using transport or inertial delays [Gho96] [GIA76]. A coupled model that includes a group of these cells will form a cellular model. **CD++** [BBW98] is a tool that allows implementing the theoretical concepts specified by the DEVS and Cell-DEVS formalisms. A specification language permits the creation of coupled models, the initial configuration for the atomic models, and the creation of external events to be used during the simulation. The original version of CD++ permits the creation of bidimensional cellular automata, where the state of a cell has a binary or three-state value.

The goal of this work is to introduce a set of extensions done to the CD++ tool [DW98], and others being implemented at present, so as to allow the execution of the models in parallel.

2. CD++

CD++ was defined building a class hierarchy in C++, using the basic concepts defined in [Zei84, Zei90]. Two basic abstract classes were defined: *Models* and *Processors*. The first are used to represent the behavior of the atomic and coupled models, while the second implement the simulation mechanisms. The *Atomic* class implements the behavior of the atomic models. The *Coupled-Model* class implements the mechanisms of the coupled models. For the case of a cellular model, a special atomic model is used to represent to each cell. To do so, *AtomicCell* and *CoupledCell* are defined as subclasses of *Atomic* and *Coupled* respectively. *AtomicCell* extends the behavior of the atomic models, to define the functionality of the cell space. On the other hand, the *CoupledCell* class permits the management of a group of atomic cells.

The *Simulator* and *Coordinator* classes manage the atomic and coupled models respectively. The *Root-Coordinator* class manages the global aspects of the simulation. It is directly related with the coupled model that has the higher level within the hierarchy. It is in charge to maintain the global time, and to start and stop the simulation process. In addition, it gets the output results.

The simulation is based on the interchange of messages between the different processors. Each message contains information to identify the sender/receiver, the time of the event, and the content that consists in a port and a value for it. Different messages are used: *X* (which represents an external event), *Y* (represents the model output), * (represents an internal event), and *done* (indicating that the model has finished its task).

The CD++ tool includes a specification language that allows describing the behavior of each cell of a cellular model. In addition, it allows to define the size of the cell space and their connection with other DEVS models (if they exist), the type of delay, the neighborhood, the border and the initial state of each cell. To do so, the theoretical definitions of the Cell-DEVS formalism are used.

This work was partially supported by the UBACYT research project TX-004, "Concurrency in distributed systems".

The specification of the behavior for a cell is defined using a set of rules. Each rule indicates the value for the cell's state if a condition is satisfied. The output of the model should be delayed by using a specified time. The BNF for the specification language is shown in the *Appendix*.

The specification of the behavior for a cell is defined using a set of rules. Each rule indicates the value for the cell's state if a condition is satisfied. The output of the model should be delayed by using a specified time. If the condition is not valid, then the following rule will be evaluated (according to the order in that they were defined), repeating this process until a rule is satisfied, or until there are no more rules.

In the latter case, an error will be raised, indicating this situation to the modeller, and aborting the simulation process. The occurrence of this error indicates that the model has been specified in incomplete form. The tool could also detect the existence of two or more rules with same condition but with different state value or delay, avoiding the creation of ambiguous models. In this situation, the simulation will be aborted. Also, when two different rules are evaluated satisfactorily and their result is the same, the modeller will be warned.

3. AN APPLICATION EXAMPLE: A HEAT DIFFUSION MODEL

This example consists of a surface represented by cellular automaton, where each cell contains a temperature. In each stage of the simulation, the temperature of the cell is calculated as the average of the values of the neighborhood. In addition, heat generator is connected to the cells (2, 2) and (5, 5), and permit the creation of temperatures in the range [24, 40] with uniform distribution. On the other hand, a generator of cold allow to create values in the range [10, 15] also with uniform distribution, and is connected to the cells (2, 8) and (8, 8). Both generators create values after *x* seconds, where *x* follows an exponential distribution with mean 50 seconds.

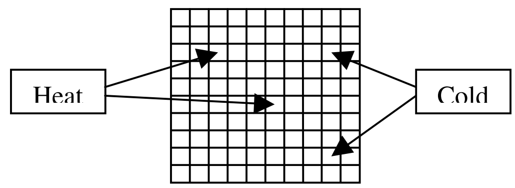

Figure 1. Coupling scheme of the heat diffusion model

```
[top]
components : surface generatorHeat@Generator generatorCold@Generator
link : out@generatorHeat  inputHeat@surface
link : out@generatorCold  inputCold@surface

[surface]
type : cell
width : 10
height : 10
delay : transport
defaultDelayTime  : 100
border : wrapped
neighbors : surface(-1,-1) surface(-1,0) surface(-1,1)
neighbors : surface(0,-1)  surface(0,0)  surface(0,1)
neighbors : surface(1,-1)  surface(1,0)  surface(1,1)
initialvalue : 24
in : inputHeat inputCold
link : inputHeat in@surface(5,5)
link : inputHeat in@surface(2,2)
link : inputCold in@surface(8,8)
link : inputCold in@surface(2,8)
localtransition : heat-rule
portInTransition : in@surface(5,5)  setHeat
portInTransition : in@surface(2,2)  setHeat
portInTransition : in@surface(8,8)  setCold
portInTransition : in@surface(2,8)  setCold

[heat-rule]
rule : { ((0,0) + (-1,-1) + (-1,0) + (-1,1) + (0,-1) + (0,1) + (1,-1) + (1,0) + (1,1)) / 9 } 10000 { t }

[setHeat]
rule : { uniform(24,40) } 1000 { t }

[setCold]
rule : { uniform(-10,15) } 1000 { t }

[generatorHeat]
distribution : exponential
mean : 50
initial : 1
increment : 0

[generatorCold]
distribution : exponential
mean : 50
initial : 1
increment : 0
```

Figure 2. Definition of the heat diffusion model

The definition of the model using the language provided by the tool is showed in the figure 2. The top model and its components are defined between lines 1 and 4. Between lines 6 and 26, the model representing the surface is defined. It is composed by a cellular automata of 10x10 cells, having an initial temperature of 24° C. In the lines 28 and 29 the local computation function is defined.

Lines 31 and 32 define the function creating a temperature in the range [24, 40] with uniform distribution. Similarly, lines 34 and 35 define the function to create temperatures in the range [10, 15] with uniform distribution. Finally, the generators of cold and heat are defined between the lines 37 and 47. Here, the values are generate each *x* seconds, where *x* follows an exponential distribution with a mean of 50 seconds.

The outputs generated by the simulation are given in appendix. In the time 00:00:01:000 the generators of cold and heat produce changes in the input cells. At the time 00:00:05:041, the generator of cold will produce a change in the state of the cells (2, 8) and (8, 8), establishing them the value 2.5 and -2.6 respectively.

4. ABSTRACT SIMULATION MECHANISM IN ASYNCHRONOUS PDES

To improve the execution times in Cell-DEVS model execution, it has been proposed that the coordinators must be implemented to execute in parallel. In this case, multiple processes will execute the simulation simultaneously. Here, each processor of an available set will have an associated flat coordinator. Asynchronous parallel discrete event simulation mechanisms will be used. The logical execution of processes associated with each processor will be synchronized by using optimist or pessimist approaches. In this way, the environment can be tailored to the application, achieving the best performance results for each case.

Each coordinator will be coded as a logical process including three different event lists: one for local events, and the others for input and output links. The main synchronization mechanisms will be executed by three basic methods: *Receive_message()*, *Transmit_Message()* and *Execute_Message()*. The behavior of these methods will differ depending on the chosen approach.

Each coordinator will be in charge to choose the imminent cells to simulate, using the flatten procedure analyzed previously. The q-messages arriving to a coordinator can have a local or remote source. When these messages are processed, new, Y and done messages are created (using the translation mechanism explained previously). In this case, the coordinators must be changed to manage the coupling with the parallel simulation environment.

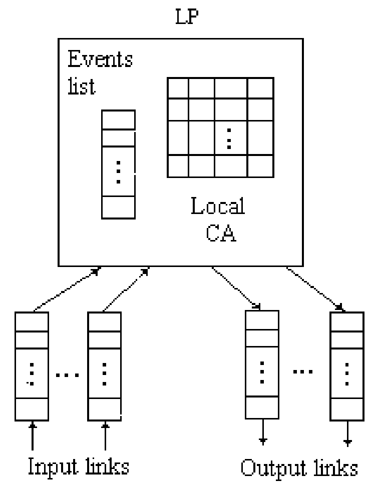

Figure 3. Structure for the logical process defined for cell spaces.

If a new output event belongs to the local processor, it will be added directly to the next event list as it was shown in the previous procedures. Instead, if the destination cell does not belong to the local processor, the message will be added into the output queue and transmitted to other processor, where it will be received as an input message.

The simulation can be splitted accordingly with the power of each processor, allowing the balance of the simulation load. The *Map_Tasks()* method will be in charge of this procedure, by executing standard load distribution algorithms. This method will be in charge of saving a submap of cells into each coordinator, indicating to which processor does each part of the cell space belong. This information is recorded into the *Processor_Mapping* instance variable, and will be used to build the input and output links. The following sections will be devoted to analyze the behavior of the simulators for each of the proposed methods.

4.1.1. Pessimist coordinators.

This section is devoted to study the behavior of each of the methods associated with the conservative coordinators for a Cell-DEVS environment. The coordination mechanism should execute the following methods:

```
Receive_Message() {
  Read a message arrived through an input link;
 Save it into the Event List;
 if the message correspond to the link with
          the lower timestamp then
              Unlock();
}

Execute_Message() {
 if the link with lower timestamp does not
     have a message then
         if Null Messages are used then
              Lookahead = LVT + d;
              Send a null message to the neighbors;
        endif
        Lock();      /* The LP should be blocked
        waiting the message with lower timestamp */
  endif

   LVT = time of the first message in the event list;
   For each message in the queue with
        timestamp = LVT do
           case kind of message do
                *: ReactionTo*Message();
                    Send a done message to the parent
                      coordinator. If it belongs to other
                        processor, put it into the
                          corresponding output link.
                Q: ReactionToQMessage();
                    Send a Y message to the parent
                      coordinator. If it belongs to other
                        processor, queue it into the
                          corresponding output link;
                    Translate(Y, Q);
           /* Using the Neighborhood relationship */
                    Insert Q into the local events list;
             endcase
    endfor
}

Transmit_Message() {
For each element into an output port do
  Send the message through the corresponding link;
}
```

Figure 4. Pessimist coordination functions for Cell-DEVS parallel simulators.

In this case, the method *Receive_Message()* must save a message into the input queue, recording its type, timestamp, origin/destination and value, according with the message type. When a message arrives, they are queued into the event list. The pair of Lock()/Unlock() methods are used to stop or reactivate the activity of the logical process. In this case, if a new message arrives with the lower timestamp, the logical process should be restarted. The method *Execute_Message()* must process all the events in the queue having timestamps lower than those of the local virtual time (LVT). This should be done only if there are messages into the input lists with timestamps higher than those of the LVT. If this is not the case, the logical process must be blocked. If the logical process is active, it must start a cycle activating the methods *ReactionToXMessage()*, and *ReactionTo*Message()*, that will start the routines of the flat coordinator.

According with the strategy chosen to manage deadlocks, the method *Recovery()* (that detects and recovers deadlocks) must be activated, or null messages must be sent. In this case, the lookahead computation is easy, due that as each cell has a fixed delay, the lookahead is equal to the delay. The method *Transmit_Message()* is activated when the coordinator, analyzing its mapping (by using the method *Translate()* that activates the Z function) decides that the message belongs to other processor. To do so, the instance variable *Processor_Mapping* is used.

4.1.2. Optimist coordinators

For these processors, the methods *Receive_Message()* and *Transmit_Message()* have the same functionality that of the pessimist ones, and the difference can be found in the *Execute_Message()* method.

```
Receive_Message() {
  Read a message arrived through an input link;
  Save it into the Event List;
}

Execute_Message() {
  if the first message in the event list has a
     timestamp lower than the LVT then
         Rollback();
  endif

  LVT = time of the first message;

  if LVT > GVT then  /* Fossil collection */
    Delete all the events with timestamps lower to
      the GVT to the Next-Events and Space lists;
  endif

  LVT = time of the first message in the event list;

  For each message in the queue whose
     timestamp = LVT do
       case type of message do
           *:  ReactionTo*Message();
             Send a done message to the parent
               coordinator. If it belongs to other
                 processor, queue it into the
                   corresponding output link;

     Q: ReactionToQMessage();
         Send a Y message to the parent coordinator.
```

```
        If it belongs to other processor, queue it
          into the corresponding output link;
        Translate(Y, Q);
         Insert Q into the local events queue;
   endcase

  if a message identical to the present exists, and
         LAZY_ CANCELLATION is used then
     if the message is the same except for
           the value then
        Generate an antimessage using the previous;
        Queue the present;
     endif
  endif
endfor

Update the Next-Events and Spaces lists with the
           new status;
}

Rollback() {

    LVT = time for the straggler;
    Search into the Next-events and Space lists
       the first element whose value is smaller
         than LVT;
     Cells = Space.Cells;

      Next_Events = Next-Events.list;

      case CANCELLATION do
            LAZY: nothing;
            AGGRESIVE: generate an antimessage
                            into the output queue;
       endcase

      Delete the elements whose timestamp
        is smaller than the LVT from the output
          queue;
}
```

Figure 5. Optimistic coordination functions for Cell-DEVS models.

In this case, the first messages of the events list are used, analyzing if the arrived event is a straggler or not. If this is the case, the event list should be restored to the instant for the straggler, as the cell space is. To do so, a list keeping all the Next-Events and other with all the cell spaces are kept. The straggler generates antimessages, and restores the previous state.

If aggressive cancellation is used, the *Transmit_Message()* method should be activated when the coordinator, analyzing the mapping, decides that the message belongs to a coordinator in a different processor. If lazy cancellation is used, the method deletes safe elements from the output queue, and antimessages are not transmitted for messages with the same contents.

5. CONCLUSION

This work introduced an extension to the tool CD++ used for the modeling and simulation of Cell-DEVS models. This formalism allows hierarchical construction of the models, which improves the development, checking and maintenance phases. The extensions introduced to the tool allow to represent new models in other domains for the sate variables. It also offers the possibility to use probabilistic functions, which permits the creation of stochastic models.

The parallel execution of these models is being considered at present. A simulation mechanism was presented for this kind of models. In addition, a new extension to the flat coordination mechanism was introduced. The formalism entitles the definition of complex cell-shaped models using a high level specification language. In this way, the construction of the simulators is improved, enhancing their safety and development costs. Besides, the parallel execution allows performance improvements without adding extra costs in development or maintenance.

REFERENCES

[BBW98] BARYLKO, A.; BEYOGLONIAN, J.; WAINER, G. "CD++: a tool to develop binary Cell-DEVS models" (in Spanish). *Proceedings of the XXII Latin-American Conference on Informatics*. Quito, Ecuador. 1998.

[Gar70] GARDNER, M. “The Fantastic Combinations of John Conway’s New Solitaire Game ‘Life’ ”. Scientific American, 23 (4), 1970, pp. 120-123.

[GM76] N.Giambiasi, A.Miara "SILOG: A practical tool for digital logic circuit simulation" 16th D.A.C San Diego , 1976

[GG96] GHOSH, S.; GIAMBIASI, N. "On the need for consistency between the VHDL language constructions and the underliying hardware design". SCS ESM '96. pp. 562-567.

[WG98] WAINER, G.; GIAMBIASI, N. "Specification, modeling and simulation of timed Cell-DEVS spaces". Technical Report n.: 98-007. Departamento de Computación. Facultad de Ciencias Exactas y Naturales. Universidad de Buenos Aires. Submitted for publication. 1998.

[RW99] RODRIGUEZ, D.; WAINER, G. "Extensions to the CD++ tool". Submitted for publication. 1999.

[Zei76] ZEIGLER, B. “*Theory of Modelling and Simulation*”. *Wiley*, N.Y. 1976.

[Zei84] ZEIGLER, B. "Multifaceted Modeling and discrete event simulation". *Academic Press*, 1984.

[Zei90] ZEIGLER, B. "Object-oriented simulation with hierarchical modular models". *Academic Press*, 1990.

APPENDIX. Simulation results

```
Time: 00:00:00:000
        0    1    2    3    4    5    6    7    8    9
   +-------------------------------------------------+
  0| 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0|
  1| 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0|
  2| 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0|
  3| 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0|
  4| 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0|
  5| 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0|
  6| 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0|
  7| 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0|
  8| 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0|
  9| 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0|
   +-------------------------------------------------+

Time: 00:00:01:000
        0    1    2    3    4    5    6    7    8    9
   +-------------------------------------------------+
  0| 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0|
  1| 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0|
  2| 24.0 24.0 35.8 24.0 24.0 24.0 24.0 24.0 -6.3 24.0|
  3| 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0|
  4| 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0|
  5| 24.0 24.0 24.0 24.0 24.0 39.5 24.0 24.0 24.0 24.0|
  6| 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0|
  7| 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0|
  8| 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 -4.0 24.0|
  9| 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0|
   +-------------------------------------------------+
                          . . .

Time: 00:00:02:000
        0    1    2    3    4    5    6    7    8    9
   +-------------------------------------------------+
  0| 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0|
  1| 24.0 25.3 25.3 25.3 24.0 24.0 24.0 20.6 20.6 20.6|
  2| 24.0 25.3 25.3 25.3 24.0 24.0 24.0 20.6 20.6 20.6|
  3| 24.0 25.3 25.3 25.3 24.0 24.0 24.0 20.6 20.6 20.6|
  4| 24.0 24.0 24.0 24.0 25.7 25.7 25.7 24.0 24.0 24.0|
  5| 24.0 24.0 24.0 24.0 25.7 25.7 25.7 24.0 24.0 24.0|
  6| 24.0 24.0 24.0 24.0 25.7 25.7 25.7 24.0 24.0 24.0|
  7| 24.0 24.0 24.0 24.0 24.0 24.0 24.0 20.9 20.9 20.9|
  8| 24.0 24.0 24.0 24.0 24.0 24.0 24.0 20.9 20.9 20.9|
  9| 24.0 24.0 24.0 24.0 24.0 24.0 24.0 20.9 20.9 20.9|
   +-------------------------------------------------+
                          . . .
Time: 00:00:05:041
        0    1    2    3    4    5    6    7    8    9
   +-------------------------------------------------+
  0| 23.3 23.9 24.3 24.3 24.1 23.7 23.1 22.6 22.3 22.6|
  1| 23.4 24.1 24.5 24.5 24.2 23.8 23.2 22.6 22.3 22.7|
  2| 23.4 24.2 24.6 24.6 24.4 23.9 23.3 22.6  2.5 22.7|
  3| 23.5 24.2 24.6 24.7 24.6 24.3 23.7 23.1 22.7 23.0|
  4| 23.7 24.1 24.4 24.7 24.7 24.6 24.1 23.6 23.2 23.3|
  5| 23.7 24.0 24.3 24.6 24.8 24.7 24.4 23.9 23.5 23.5|
  6| 23.5 23.9 24.1 24.4 24.6 24.5 24.1 23.6 23.3 23.3|
  7| 23.3 23.8 24.0 24.2 24.3 24.2 23.7 23.1 22.8 22.9|
  8| 23.2 23.7 24.0 24.1 24.1 23.8 23.3 22.7 -2.6 22.6|
  9| 23.2 23.8 24.1 24.1 24.0 23.7 23.2 22.6 22.4 22.6|
   +-------------------------------------------------+
```

Simulation results of the heat diffusion model

APPENDIX. CD++ SPECIFICATION LANGUAGE.

```
RULELIST   = RULE  |  RULELIST
RULE       = RESULT RESULT { BOOLEXP }
RESULT     = CONSTANT  |  { REALEXP }
BOOLEXP    = BOOL  |  (BOOLEXP)  |  REALRELEXP  | not BOOLEXP  |  BOOLEXP OP_BOOL BOOLEXP
OP_BOOL    = and  |  or  |  xor  |  imp  |  eqv
REALRELEXP = REALEXP OP_REL REALEXP  |  COND_REAL_FUNC(REALEXP)
REALEXP    = IDREF  |  (REALEXP)  |  REALEXP OPER REALEXP
IDREF      = CELLREF  |  CONSTANT  |  FUNCTION  |  portValue(PORTNAME)
CONSTANT   = INT  |  REAL  |  CONSTFUNC  |  ?
FUNCTION   = UNARY_FUNC(REALEXP) | WITHOUT_PARAM_FUNC  |  BINARY_FUNC(REALEXP, REALEXP)  |
             if(BOOLEXP, REALEXP, REALEXP)  |  ifu(BOOLEXP, REALEXP, REALEXP, REALEXP)
CELLREF    = (INT, INT)
BOOL       = t  |  f  | ?
OP_REL     = !=  |  =  |  >  |  <  |  >=  |  <=
OPER       = +  |  -  |  *  |  /
INT        = [SIGN] DIGIT {DIGIT}
REAL       = INT  |  [SIGN] {DIGIT}.DIGIT {DIGIT}
SIGN       = +  |  -
DIGIT      = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
PORTNAME   = thisPort  |  STRING
STRING     = LETTER {LETTER}
LETTER     = a | b | c |...| z | A | B | C |...| Z
CONSTFUNC  = pi | e | inf | grav | accel | light | planck | avogadro | faraday | rydberg |
euler_gamma | bohr_radius | boltzmann | bohr_magneton | golden | catalan | amu |
electron_charge | pem | ideal_gas | stefan_boltzmann | proton_mass | electron_mass |
neutron_mass
WITHOUT_PARAM_FUNC = truecount | falsecount | undefcount | time | random | randomSign
UNARY_FUNC = abs | acos | acosh | asin | asinh | atan | atanh | cos | sec | sech | exp | cosh
| fact | fractional | ln | log | round | cotan | cosec | cosech | sign | sin | sinh |
statecount | sqrt | tan | tanh | trunc | truncUpper | poisson | exponential | randInt | chi |
asec | acotan | asech | acosech | nextPrime | radToDeg | degToRad | nth_prime | acotanh |
CtoF | CtoK | KtoC | KtoF | FtoC | FtoK
BINARY_FUNC = comb | logn | max | min | power | remainder | root | beta | gamma | lcm | gcd |
normal | f | uniform | binomial | rectToPolar_r | rectToPolar_angle | polarToRect_x | hip |
    polarToRect_y
COND_REAL_FUNC = even | odd | isInt | isPrime | isUndefined
```