Introduction to the DigiHive environment

Rafal Sienkiewicz and Wojciech Jedruch

Gdansk University of Technology, Gdansk, Poland,
Rafal.Sienkiewicz@eti.pg.gda.pl Wojciech.Jedruch@eti.pg.gda.pl

1 The DigiHive environment

The environment is defined over two dimensional continuous space with periodic boundary condition.

2 Particles and complexes

Basic universe constituent objects are called *particles*. Particles are persistent – they can not be created nor annihilated during simulation. Particles are represented by hexagons. Particles are of 256 types and are characterized by velocity, position, internal energy. Each type is related with set of constant attributes: mass, bond energy (energy needed to disrupt bond between particle of given types), activation energy (energy needed to initiate any transformation of bonds)

Particles can bond together forming a *complex* of particles. The permanence of complex and its ability to react, depends on its constituent particles bond energies. Particles can bond vertically on the directions up (U) and down (D) forming stacks. The particles on the bottom of the stack can bond horizontally in the directions north (N), north-west (NW), south-west (SW), south (S), south-east (SE), and north-east (NE). An example of stack of particles and complex formed by horizontal bonds are shown in Fig.1.

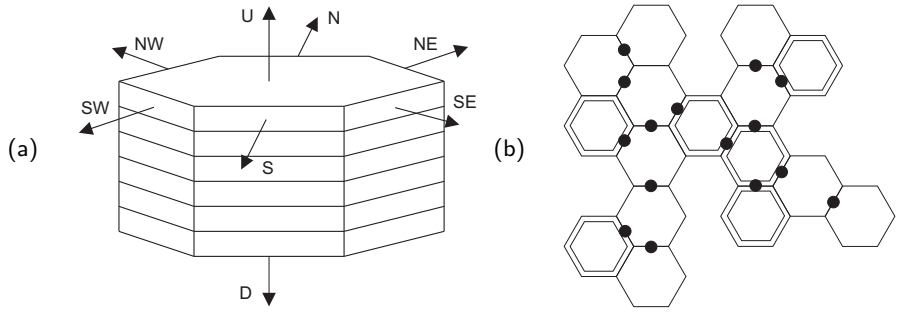

Fig. 1. Examples of complexes: (a) horizontal view of single stack of particles with directions shown, and (b) vertical view of complex formed by horizontal bonds, where hexagons drawn by double lines represent stacks of particles and black dots marks bonds between stacks

All processes are synchronized by a discrete clock. At the end of time step each particle's position is updated according to its velocity and the length of the time step.

There are two types of collisions of particles and complexes, elastic and inelastic one, where the type is chosen randomly according to a preset probability. During elastic collision, the resulting velocities are calculated according to the rules of classical mechanics resolving the collision of two disks (circles surrounding hexagons) – conservation of momentum and of kinetic energy is observed. After inelastic collision the velocities of both particles are equalized – conservation of momentum is observed but the resulting decrease of kinetic energy of particles is compensated by emission of a *photon*.

3 Photons

In addition to permanent particles the universe contains temporary entities called *photons* which transport energy. They have no mass or momentum and move with constant velocity. Photons are characterized by the following attributes: energy, position and direction angle .

Photons are created in the following situations:

1. After inelastic collision of particles or complexes,
2. After bonding of particles (bond energy is converted into photon)
3. Emitted spontaneously by particles – in each time step a particle may (with preset probability) convert part of its internal energy into a new photon.

Photons may collide with particles. There are two type of collisions: elastic and inelastic. Elastic collisions change photon direction only (direction angle has new random value), after inelastic collision one of the following event is randomly selected (each event has its own preset probability):

1. Rebounding of the particle hit by the photon from an adjoining particle. Energy of photon is transformed into the kinetic energy of rebounded particles.
2. Creating horizontal or vertical bond between the hit particle and adjoining particle. The photon energy should be greater than the bond activation energy. After the reaction a new photon is created having energy equal to the sum of: energy of hitting photon, energy of created bond and decrease of kinetic energy of bonded particles.
3. Removing horizontal or vertical bond between the hit particle and bound particles. The photon energy should be greater than both: bond activation energy and bond energy. After the reaction the photon is created having energy equal to the energy of hitting photon minus the energy of removed bond.
4. Photon absorption. Photon is absorbed by the particle, and is converted into its internal energy.

If selected event cannot be completed, e.g. because of insufficient photon energy, no other action is performed (photon skip the particle).

The above constitutes the first level of the system. When there is no photons and the probability of inelastic collisions of particles is set to zero the system behave like classical Newtonian one. Particles and complexes of particles move and collide. Otherwise, the possible inelastic collision of particles produce photons which in consequence can randomly change structures of particles by creating and removing bonds between them.

4 Functions

Besides the reactions resulting from the collision of particles with photons, there exists additional class of interactions in which complexes of particles are capable to recognize and manipulate particular structures of particles in the space around them. The description of the function encoded in the complex is contained in types and locations of particles in the complex, which is interpreted as a program written in a specially defined, described below language.

5 Syntax

Program syntax is similar to Prolog language using the following predicates only: **program**, **search**, **action**, **structure**, **exists**, **bind**, **unbind**, **move**, **not**. Similarity to Prolog is clearly visible in the overall structure and execution algorithm. Specific syntax of above predicates (especially **exists**) disallows however easy transformation of program into well-formed Prolog. Language syntax in BNF notation is shown in the Fig. 2.

An example of simple program is presented in the Fig. 3. The program recognizes the structure shown in Fig. 4, and then binds the particle 11111111 the unbond particle 00000000.

6 Semantics

Program execution is based on Prolog backtracking algorithm. Each predicate returns one of the following status: *OK** (means that unification succeed but not every possibility has been yet checked), *OK* (means that unification succeed and every possibility has been already checked) and *FAIL* (means that unification failed).

The predicates of the language may be divided into the following groups:

6.1 program, search, action, structure

Each program consists of single, main predicate **program**. This predicate always consists of the two following predicates: **search** and **action**. The first one calls searching predicates, while the second one calls execution predicates that can

Fig. 2. Language syntax

```
program ::= program_header program_body
program_header ::= program() :- search(), action().
program_body ::= search action definitions
search ::= search() :- header .
action ::= action() :- row_action {, row_action } .
definitions ::= row_definition {row_definition }
row_definition ::= header :- body.
header ::= structure( integer )
body ::= exists( exists ) {,exists( exists )} {,not( header )}
     |not( header ) {,not( header )}
short ::= 0|1|2|3|4|5 |6|7| 8|9
integer ::= short {short}
exists ::= [not] c [[[not] bound [to f] [on d]] | [adjacent [to f] [on d] ]], [mark f]
row_action ::= bind( action_spec ) | unbind( unbind_spec ) | move( action_spec )
action_spec ::= f to f on d
unbind_spec ::= f [from f] [on d]
c ::= [0|1|×, 0|1|×, 0|1|×, 0|1|×, 0|1|×, 0|1|×, 0|1|×, 0|1|×]
f ::= V short
d ::= N|NE|SE|S|SW|NW |U|D
```

affect particles in its surrounding (by moving particles or creating and removing bonds between them). It is easy to notice that "actions" are performed only if searching succeeded, i.e. particular structures were recognized.

Searching commands are grouped in the **structure** predicates, which describes some particular structures. These predicates groups both predicate **exists** and other predicates **structure**. Predicate **structure** embedded in other **structure** is always in its negative form, **not(structure())**, i.e. structure description is formed by the sequence of **exists** predicates and sequence of some negative condition. The information flow in embedded structures is possible only downwards, i.e. embedded structures can check some additional condition, but they cannot send any information upwards (besides simple answer – structure exists or structure does not exists).

exists

Predicate **exists** is the basic command for structure searching. It recognizes a particle (or empty place adjoining a particle) of particular type, with particular bond structure. It is possible to check the basic states of the particle and relations between two particles.

The following states are checked:

- [**not**] **is bound** – checks if there exists (or not exists) any bond between checked particle and some other particle
- [**not**] **is bound on d** – checks if there exists (or not exists) any bond between checked particle and some other particle on direction **d**

Fig. 3. Example of program recognizing the structure shown in Fig. 4

```
program():–
        search(),
        action().
search():–
        structure(0).
structure(0):–
        exists ([0,0,0,0,0,0,×,×], mark V1),
        exists ([1,1,1,1,1,1,1,1] bound to V1 on N, mark V2),
        exists ([0,0,0,0,0,0,0,0], mark V5),
        not(structure(1)),
        not(structure(2)).
structure(1):–
        exists ([1,1,1,1,0,0,0,0] bound to V2 on NW, mark V3),
        exists ([1,1,1,1,0,0,0,0] bound to V3 on SW, mark V4),
        not(structure(3)).
structure(3):–
        exists ([0,0,0,0,1,1,1,1] bound to V4 on S).
structure(2):–
        exists ([1,0,1,0,1,0,1,0]).
action():–
        bind (V2 to V5 on SW).
```

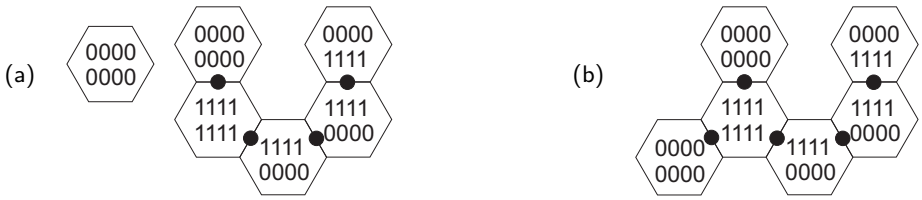

Fig. 4. Single particle and a complex of particles recognized by program listed in the Fig. 3 (a), and the structure after action of the program (b)

The following relations can be also checked:

- **is adjacent to V** – checks if the particle is adjacent to particle **V**. Any adjacent particles must belong to the same complex
- **is adjacent to V on d** – checks if the particle is adjacent to particle **V** on direction **d**. Any adjacent particles must belong to the same complex
- [**not**] **is bound to V** – checks if the particle is bound to particle **V**
- [**not**] **is bound to on d** – checks if the particle is bound to particle **V** on direction **d**

bind, unbind, move

Predicates of the group affect other particles. If any of the predicate fails the whole program fails (backtracking is not performed). In case of program fail, any partial changes in space are reverted – "everything or nothing ".

- **move V1 to V2 on d** – moves particle **V1** to place adjacent to particle **V2** on direction **d**. The movement does not change the mass center of collection: complex encoding program, complex into which belongs **V1** and complex into which belongs **V2**.
- **bind V1 to V2 on d** – moves particle **V1** to place adjacent to particle **V2** on direction **d** and then creates bond between them on **d** direction.
- **unbind V1 from V2** – removes bond between particles: **V1** and **V2**.

not Negates the single **structure** predicate.

7 Interpreter

Programs encoded in structures of particles are translated into Prolog and run by built-in simplified interpreter. During the translation, predicates discussed in 6 are replaced by sequences of Prolog predicates.

After the photons movements, list of complexes in environment is processed in random order. If complex encodes a program, it is passed to interpreter then translated into internal representation and run. Just before execution, interpreter creates list of facts about particles which are visible to program. Those particles are contained in a circle of fixed radius surrounding the program (called Ω).

After the database creation execution is performed. If both search and action part of the program succeeded interpreter tries to apply changes into the environment (note that until that moment, program only affect fact list and has no influence on space). Next the condition is checked that energy balance of changes must be positive and not less than activation energy. If this is not fulfilled interpreter gain some energy by lowering participating particles internal energy.

If the above conditions are fulfilled, interpreter applies changes to environment. Otherwise, program is rotated and executed again, i.e. before another execution, every direction related argument in predicates is changed according to rules: N $\rightarrow$ NE, NE $\rightarrow$ SE, SE $\rightarrow$ S, S $\rightarrow$ SW, SW $\rightarrow$ NW, NW $\rightarrow$ N. Only if all possibilities were tried and failed, execution of program is cancelled. The summary of execution algorithm is presented in Fig. 5.

8 Encoding

A complex of particles may encode a program. Each predicate **structure** is represented by the single stack of particles.

Every stack encodes a list of predicates **exists**. Stack which encodes **structure**(0) also encodes predicates **bind**, **unbind** and/or **move**. Adjoining stacks encode negative form of predicates **structure**. Program listed in the Fig. 3 is represented by the structure of particles as shown in the Fig. 6.

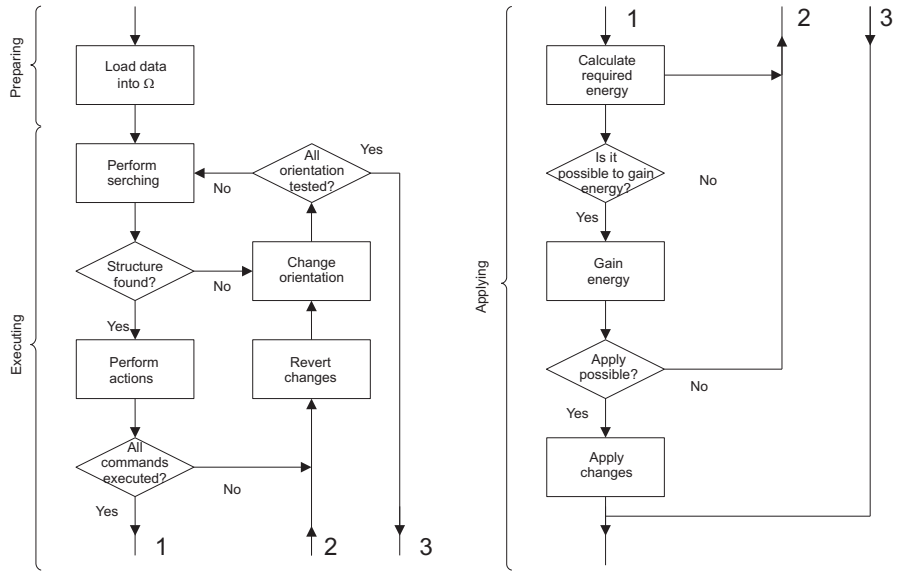

Fig. 5. Program execution algorithm

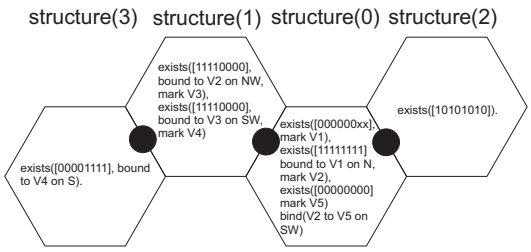

Fig. 6. Program listed in the Fig. 3 encoded in complex of particles. Note that the predicates **not(structure)** are in stacks of particles which adjoin the **structure**(0) stack

Stacks encoding predicates **structure** are labelled by specific particles at the bottom of the stacks. Also inside the stacks, particles of specific types labels the beginning of predicates, e.g. 00110000 encodes the beginning of predicate **exists**.

The main idea was to develop a language with property that small changes in code of a program (i.e. in particles in which the program is encoded) should usually lead to small changes in the recognition and action parts of the program. Such property of the language is crucial while using the system to simulate the evolutionary, spontaneous development of complex structures.

In the presented language usually small changes of code lead to removing or changing some predicates which often reduces only the recognizing capabilities of the program, e.g. reduce its precision.