Queens of the Hill

Artiom Alhazov[1], Sergiu Ivanov[2], David Orellana-Martín[3,4]

[1]Vladimir Andrunachievici Institute of Mathematics and Computer Science,
The State University of Moldova, Academiei 5, Chișinău, MD-2028, Moldova
artiom@math.md

[2]IBISC Laboratory, Université Paris-Saclay, Univ Évry
91020, Évry-Courcouronnes, France
E-mail : sergiu.ivanov@univ-evry.fr

[3]Research Group on Natural Computing,
Department of Computer Science and Artificial Intelligence,
Universidad de Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
E-mail: dorellana@us.es

[4]SCORE Laboratory, I3US, Universidad de Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain

Summary. Inspired by the programming game Core Wars, we propose in this work a framework and the organisation of king of the hill-style tournaments between P systems. We call these tournaments Queens of the Hill and the individual contestants valkyries. The goal of each valkyrie is to dissolve as many membranes of as many other valkyries as possible, while at the same time resisting the attacks. Valkyries are transition P systems with cooperative rules, target indication, and rudimentary matter–anti-matter annihilation rules. These ingredients are sufficient for computational completeness, but the context of Queens of the Hill reduces the relevance of this statement. We give some tentative examples of strategies and discuss their advantages and drawbacks. Finally, we describe how Queens of the Hill can be used as a teaching exercise, and also a tool to federate the students' creativity to push the frontiers of membrane computing.

Keywords: Core Wars, membrane dissolution, anti-matter, interaction.

1 Core Wars

To cite [11], *"Core War (or Core Wars) is a programming game where assembly programs try to destroy each other in the memory of a simulated computer."* In Core Wars, programmers design programs—called warriors—with two goals in mind:

1. kill as many other programs as possible,

2. survive for as long as possible against the attacks of the other programs.

In the most basic setup, all programs are loaded in the same shared memory space, and only feature instruction segments, i.e. their memory only contains code, and data is stored as part of some of the instructions. No memory protection is available for the instructions, so all programs can write anywhere, including to the instruction segments of competitors, which is the primary way of attacking. The simplest warrior is called the Imp and only consists of a single instruction in the special assembly language called Redcode:

```
MOV 0, 1
```

The numbers correspond to addresses in the memory space relative to the current instruction, so `0` refers to the current instruction slot, and `1` refers to the next one. This program copies its only instruction to the next memory slot, which then copies itself to the next one, etc. The Imp therefore ends up populating the whole memory with copies of itself.

As small and impressive as it is, the Imp will never actually win, because it just reproduces itself, possibly over the code of the competitors, but it never kills any competitor. To kill a process, Redcode features the special instruction `DAT`. When it is executed, the current process is killed. A simple winning code would throw `DAT` over the whole memory, while simultaneously avoiding to run this instruction in its own execution. This is what the warrior called the Dwarf does, whose detailed presentation is given in [11].

Multiple servers exist continuously running Code Wars tournaments in the king of the hill mode (see section "Climbing the hill" in [11]): 10 to 30 warriors are loaded in the same shared memory space and are run sequentially, on a single virtual processor, which interleaves the execution of the instructions of every warrior. The score of a warrior in a match roughly corresponds to the number of other warriors it has killed. The warrior with the highest score is the current king of the hill, and the warrior with the lowest score falls off the hill: it is replaced by a new warrior.

2 Queens of the Hill

In this submission we propose a framework for running king of the hill style tournaments between P systems. We refer to such tournaments as (P) *Queens of the Hill*, and we call individual contestants *valkyries*. In this section, we propose the formal framework to be used for the valkyries as well as the rules for Queens of the Hill tournaments.

2.1 Valkyries

Our choice of the P system variant for the valkyries is guided by the following two principles: ability to interact with the other contestants and ease of programming.

We choose here a variation on what is sometimes called transition P systems, which is partially inspired by P automata with matter–anti-matter annihilation rules as shown in [5] and by P colonies [2]. As a reminder, the original P automata rely on antiport rules exclusively [3].

We define a (valkyrie) P system as the following tuple:

$$\Pi = (O, \mu, w_1, \ldots, w_n, R_1, \ldots, R_n), \text{ where}$$

- $O = \Sigma \cup \Delta_k$ is a finite alphabet of objects,
- $\Delta_k = \{\delta_t, \bar{\delta}_t \mid 1 \leq t \leq k\} \cup \{\delta\}$ for some fixed $k \in \mathbb{N}$,
- μ is the hierarchical membrane structure bijectively labeled by the numbers from 1 to n and usually presented as a sequence of correctly nested brackets,
- w_i is the initial multiset in membrane i, $1 \leq i \leq n$,
- R_i is the finite set of rules in membrane i, $1 \leq i \leq n$.

The rules in R_i feature full cooperation and may use target indications. More precisely, a rule in R_i has the form $u \to v$, where $u \in \Sigma^\circ$, $u \neq \lambda$, is a non-empty multiset over Σ, and $v \in (O \times \mathit{Tar})^\circ$ is a multiset of symbols over O, each equipped with target indications $\mathit{Tar} = \{\mathit{in}, \mathit{here}, \mathit{out}\}$. A symbol appearing with the indication *in* in v will be sent into a non-deterministically chosen inner membrane of membrane i, a symbol with the indication *here* will remain in membrane i, and a symbol with the indication *out* will be sent to the parent membrane. If membrane i does not have any inner membranes, the symbols with target indication *in* will be kept in membrane i, i.e. the target indications *in* and *here* are equivalent in the case of elementary membranes. For readability, we will always omit the indication *here*, i.e. instead of writing $(a, \mathit{here})(a, \mathit{here})(b, \mathit{out})$ we will write $aa(b, \mathit{out})$.

The symbol $\delta \in \Delta_k$ has the special semantics of dissolving the membrane in which it appears. More formally, once δ is introduced into membrane i, all of its objects and inner membranes are moved to its parent membrane, and membrane i is removed from the system—non-elementary membrane dissolution is allowed. Membrane dissolution happens at the end of a computation step, and all introduced copies of δ are removed from the system after all dissolutions are performed. It follows incidentally that introducing any number of copies of δ in a membrane produces exactly the same as effect as introducing one copy of δ. Dissolution of the outermost (skin) membrane is forbidden, i.e. introducing a copy of δ into the skin membrane will have no effect and the symbol δ will be immediately removed.

All sets R_i, $1 \leq i \leq n$, also include the following rules:

$$\begin{aligned} R_k^\delta = {} & \{\delta_t \to \delta_{t-1} \mid 2 \leq t \leq k\} \cup \{\delta_1 \to \delta\} \\ & \cup \{\delta_t \bar{\delta}_t \to \lambda \mid 1 \leq t \leq k\}. \end{aligned}$$

Informally, the symbol δ_t is equipped with a timer which triggers the dissolution of the containing membrane after t steps. The rules $\delta_t \bar{\delta}_t \to \lambda$ have weak priority, meaning that if both a copy of δ_t and $\bar{\delta}_t$ are present, then they must be erased (annihilate), preempting the evolution rule for δ_t.

Since the left-hand sides of the rules in $R_i \setminus R_k^\delta$ are multisets over Σ, these rules cannot directly detect or rewrite the symbols in Δ_k. However, they can produce the anti-symbol $\bar{\delta}_t$ to force the annihilation of a symbol δ_t if it is present in the current membrane.

The rules are applied in the maximally parallel way, with weak priority of the annihilation rules $\delta_t\bar{\delta}_t \to \lambda$. A computation steps proceeds in the classical fashion, by first non-deterministically choosing a non-extendable multiset of rules to apply, applying it, and performing all the necessary dissolutions. A halting configuration is a configuration in which no more rules are applicable. We can consider halting computations of P systems, but due to the continual nature of the tournament, we will generally consider infinite or time-limited computations instead.

Example 1. Consider the following valkyrie P system:

$$\begin{aligned}
\Pi &= (O, [_1[_2[_3]_3]_2]_1, d, b\bar{\delta}_1, a, R_1, R_2, R_3),\\
O &= \{a, b, c, d\} \cup \Delta_2,\\
R_1 &= \{d \to d, d \to d(\delta_2, in)\} \cup R_2^\delta,\\
R_2 &= \{bc \to b\} \cup R_2^\delta,\\
R_3 &= \{a \to aa(c, out), a \to \delta\} \cup R_2^\delta.
\end{aligned}$$

As a reminder, $\Delta_2 = \{\delta_2, \delta_1, \delta\} \cup \{\bar{\delta}_1, \bar{\delta}_2\}$ and $R_2^\delta = \{\delta_2 \to \delta_1, \delta_1 \to \delta\} \cup \{\delta_2\bar{\delta}_2 \to \lambda, \delta_1\bar{\delta}_1 \to \lambda\}$. Figure 1 gives a graphical illustration of the P system above. For conciseness, we omit the rules in R_2^δ from such graphical illustrations.

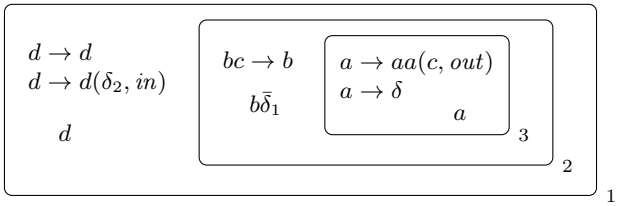

Fig. 1. A simple valkyrie P system. The rules from R_2^δ are not represented.

The rule $a \to aa(c, out)$ in membrane 3 doubles some of the a, and also ejects the corresponding number of c in membrane 2. The remaining copies of a are used to produce δ, which will dissolve membrane 3, copying all instances of a it contains into the parent membrane 2. The symbol b in membrane 2 will progressively erase all the copies of c ejected by membrane 1.

The symbol d in the skin may choose between simply maintaining itself, or also injecting δ_2 into membrane 2. The first copy of δ_2 injected into 2 will undergo the evolution rule $\delta_2 \to \delta_1$, and will afterwards annihilate with $\bar{\delta}_1$ already present there from the start. However, the second copy of δ_2 the rule $d \to d(\delta_2, in)$ will inject into membrane 2 will be free to produce δ in two steps thereby dissolving membrane 2. If by this time the rule $a \to \delta$ has not yet been applied in membrane 3,

membrane 3 will become the direct inner membrane of membrane 1, so the next application of the rule $d \to d(\delta_2, in)$ will send δ_2 in membrane 3, leading to its dissolution in two steps. Therefore, this valkyrie P system always converges to a cycle of configurations in which there is only the skin membrane containing a copy of d, a copy of b, possibly some copies of c, some copies of a, as well as a symbol from $\{\delta_2, \delta_1\}$, which always ticks down to δ without any effect, since the dissolution of the skin membrane is disallowed. □

2.2 Tournament Setup

The setup of Queens of the Hill tournaments is partially inspired by P colonies [2]: a set of valkyrie P systems is grouped together in a big skin membrane, which always sends back in whatever is sent out. More formally, we define an m-Queens of the Hill tournament as the following tuple:

$$\mathcal{Q} = (O, \Pi_1, \ldots, \Pi_m),$$

where $O = \Sigma \cup \Delta_k$ and Π_j is a valkyrie P system as defined in Section 2.1. All P systems Π_j share the same sets of symbols Σ and O. The tournament $\mathcal{Q}$ is a P system obtained by placing all Π_j into a common outer membrane 0 with the empty initial multiset and with the following set of rules:

$$R_0 = \{a \to (a, in) \mid a \in O\} \cup R_k^\delta.$$

In other words, R_0 always sends in whatever symbols are sent out from the individual valkyries, but due to non-determinism these symbols do not necessarily end up in the valkyrie which produced them. Note that R_0 contains 2 rules for symbols δ_t: such a symbol may be sent in, or it may evolve into δ_{t-1}. As before, if the corresponding anti-symbol $\bar{\delta}_t$ is also present, the annihilation rule $\delta_t \bar{\delta}_t \to \lambda$ will have to be applied. Finally note that R_0 is the only set of rules in which the left-hand sides are allowed to include δ_t.

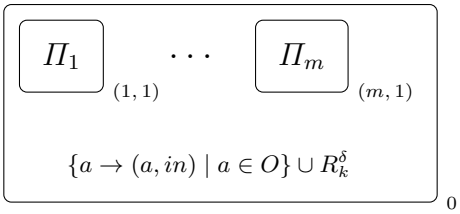

Fig. 2. An informal picture of an m-Queens of the Hill tournament $\mathcal{Q}$. The skin membranes of the valkyries Π_j are relabelled as $(j, 1)$.

A Queens of the Hill tournament obeys the same semantics as valkyrie P systems defined in Section 2.1. In particular, this means that the dissolution of the

skin membranes of a valkyrie Π_j *is allowed*, because at this time it is surrounded by the bigger skin membrane of the whole tournament $\mathcal{Q}$. To preserve consistent membrane labelling, a membrane i in the valkyrie P system Π_j is renamed into membrane (j, i) in the tournament $\mathcal{Q}$.

2.3 Tournament Organization

An m-Queens of the Hill tournament runs all the valkyries in the maximally parallel mode multiple times and for a limited number of steps. At the end, the score of each valkyrie is computed from the number of its membranes that was dissolved. Non-determinism in the computations is resolved probabilistically, as it is done in the P-Lingua framework [4, 6]: at every non-deterministic branching point, one of the branches is chosen under the uniform probability distribution.

More concretely, the tournament runs in the following way:

1. Run the computation for N steps, resolving non-determinism according to the uniform probability distribution.
2. Repeat Step 1 M times.

The score of a valkyrie is computed according to the following formula:

$$\text{score}(\Pi_j) = \frac{1}{|\Pi_j|}\left(|\Pi_j| - \frac{1}{M}\sum_{i=1}^{M}\text{diss}_i(\Pi_j)\right),$$

where $\text{diss}_i(\Pi_j)$ is the number of membranes of Π_j that were dissolved during the i-th computation (i-th run of Step 1 above), and $|\Pi_j|$ is the total number of membranes in Π_j.

Example 2. Suppose that Π_j has 5 membranes, $|\Pi_j| = 5$, and take $M = 3$. Further suppose that 2, 3, and 4 membranes of Π_j were dissolved respectively in the first, second, and third computations, i.e. $\text{diss}_1(\Pi_j) = 2$, $\text{diss}_2(\Pi_j) = 3$, and $\text{diss}_3(\Pi_j) = 4$. Then the score of Π_j in this tournament will be:

$$\frac{1}{5}\left(5 - \frac{2+3+4}{3}\right) = \frac{2}{5}.$$

Informally, the score of a valkyrie is how many membranes on average it retains by the end of a computation of the tournament, normalized by its total number of membranes. □

A valkyrie has the highest score of 1 if none of its membranes is ever dissolved in the tournament. It has the lowest score of 0 if all its membranes are always dissolved.

2.4 Tournament Parameters

Table 1 summarizes the parameters governing a Queens of the Hill tournament that were introduced in the previous sections. The values of these parameters may have a significant impact on the strategies adopted by the individual valkyries. Smaller values of $|\Sigma|$ reduce the richness of the behaviors of a valkyrie and make it less robust to perturbations coming from the skin membrane 0, i.e. from the other valkyries. Larger values of k mean more opportunities for the symbols δ_t to be captured. Larger values of m mean lower probability of receiving a symbol δ_t after emitting it into the skin membrane 0. Shorter computation lengths N mean that lightning attacks may be more feasible, while smaller values for M mean fewer computations in a tournament, which increases the contribution of randomness to the outcome.

$\|\Sigma\|$	10	The number of working symbols.
k	5	The maximal value of the index t in δ_t.
m	10–20	The number of entrants in the tournament.
N	1000	The length of a computation in the tournament.
M	50	The total number of computations in the tournament.

Table 1. A summary of the parameters governing a Queens of the Hill tournament, together with the possible values for these parameters.

3 A Note on Computational Complexity

Valkyrie P systems as defined in Section 2 are quite obviously computationally complete, even with a subset of the ingredients. In particular, full cooperation together with the maximally parallel mode suffice to simulate arbitrary register machines. We refer the reader to the first publication in membrane computing [15] for the very first proofs, as well as to the more recent [10, 16] for a sample of the wide variety of techniques for proving computational completeness of P system variants. For the record and for the sake of the discussion of the possible strategies in Queens of the Hill tournament, we briefly recall a proof of computational completeness of P systems as defined above.

A (deterministic) register machine is an abstract computational device consisting essentially of a finite set of registers and a program. The registers can contain natural numbers or zero. The program consists of the following two types of instructions:

- $(p, \mathrm{ADD}(r), q)$: in state p, increment register r and go to state q;
- $(p, \mathrm{SUB}(r), q, s)$: in state p, check the value of register r; if its value is strictly positive, decrement it and go to state q; otherwise go to state s.

Register machines are famously computationally complete. We refer the reader to [13] for a much more in-depth discussion.

One-membrane valkyrie P systems can simulate both types of register machine instructions, even without dissolution or anti-matter rules. Classically, the alphabet Σ will include one symbol per state p of the register machine, and the value of register r will be represented by the multiplicity of symbol a_r. The instruction $(p, \mathrm{ADD}(r), q)$ can be directly simulated by the rule $p \to q a_r$. The simulation of $(p, \mathrm{SUB}(r), q, s)$ is more intricate, as usual, and relies on non-determinism and maximal parallelism: the symbol p non-deterministically guesses whether the register is empty, and a trap symbol is produced if the guess is wrong. The following table lists the rules for both branches, arranged by steps:

	Decrement	*Zero test*
1.	$p \to \bar{p}_1 \hat{p}_1$	$p \to \tilde{p}_1 \dot{p}_1$
2.	$\bar{p}_1 a_r \to \bar{p}_2,\ \hat{p}_1 \to \hat{p}_2$	$\dot{p}_1 a_r \to \#,\ \tilde{p}_1 \to \tilde{p}_2$
3.	$\hat{p}_2 \bar{p}_2 \to q,\ \hat{p}_2 \bar{p}_1 \to \#$	$\tilde{p}_2 \dot{p}_1 \to s$

The decrement branch begins by splitting the state symbol p into $\bar{p}_1$ and $\hat{p}_1$. The symbol $\bar{p}_1$ erases a copy of a_r if it is present in the system and evolves into $\bar{p}_2$. It does not evolve if no copies of a_r are present. At the same time, $\hat{p}_1$ evolves into $\hat{p}_2$. In the third step, $\hat{p}_2$ evolves into q in the presence of $\bar{p}_2$, i.e. in the case in which the decrement was successful. If the decrement could not happen, $\hat{p}_2$ finds $\bar{p}_1$, which produces the trap symbol.

The zero test branch begins by splitting the state symbol p into $\tilde{p}_1$ and $\dot{p}_1$. The symbol $\dot{p}_1$ must evolve into the trap symbol # if it finds a copy of a_r, as $\dot{p}_1 a_r \to \#$ is the only rule which may transform $\dot{p}_1$. In the meantime, $\tilde{p}_1$ evolves into $\tilde{p}_2$. If in the third step $\dot{p}_1$ is still present in the system, this means that it did not find any copies of a_r, the register is empty, and the symbol s is produced. Otherwise $\tilde{p}_2$ cannot evolve, but this also means that a trap symbol was produced at step 2, meaning that the computation will never halt.

The argument above shows that the language of valkyries in Queens of the Hill tournaments is rich enough. However, note how this argument relies on two essential details which are partially relevant or even irrelevant in Queens of the Hill: non-determinism and halting. On the one hand, non-determinism is resolved probabilistically, meaning that not all possibilities will be explored, and that some of them may be explored multiple times. Furthermore, proofs of computational completeness in P systems classically consider the results produced at the end of halting computations, while in Queens of the Hill halting does not have a central role. What is important in Queens of the Hill is communicating with the other valkyries, i.e. attempting to dissolve as many of their membranes as possible, as soon as possible. From this standpoint, efficiency is important, while actual computational complexity is much less relevant, as long as the valkyrie manages to attain a relatively high score. Finally, note how $|\Sigma|$ is a powerful tool for modulating the complexity and the efficiency of individual valkyries.

4 Tentative Strategies

The main goal of Queens of the Hill is turning P system design into a game involving teams of students on the front line, backed by researchers collecting and systematizing the explicit and implicit knowledge produced by the teams designing the valkyries. In this section, we present several tentative strategies, whose efficiency or relevance will be the subject of immediate future work.

One of the first strategies one may think of when seeing the rules of Queens of the Hill is the Bomber: eject δ_t for some value of t into the skin membrane 0 and hope that none of those symbols is sent back into the same membrane. The

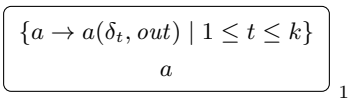

Fig. 3. The Bomber.

efficiency of the bomber decreases as the number of valkyries decreases. For example, when there is only one other valkyrie, the probability is quite high that the ejected δ_t lands back in the Bomber. Note that this probability is not exactly $\frac{1}{2}$, since the rule $\delta_t \to \delta_{t-1}$ can also be applied in the skin, potentially until the production of δ.

The Bomber can be made more robust by making it accumulate copies of $\bar{\delta}_t$ for some values of t, so that the symbols δ_t coming from the skin annihilate with the corresponding copies of $\bar{\delta}_t$. While this strategy can deal with an occasional $\bar{\delta}_t$, it will be quickly overwhelmed when sharing the tournament with a considerable population of bombers.

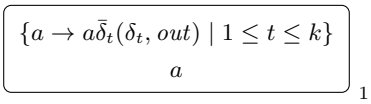

Fig. 4. The Bar Bomber.

Another variation of the Bomber is the strategy of ejecting $\bar{\delta}_t$ in the hope of neutralizing δ_t before it even gets into the valkyrie. This has the obvious disadvantage that it will also protect the other valkyries from δ_t.

In case the number of competing valkyries m—or an upper bound on m—is known, robustly dealing with such bomber strategies is in fact not very difficult: it suffices to ensure the presence of $r(m-1)$ copies of $\bar{\delta}_1$ at all times, where $r \in \mathbb{N} \setminus \{0\}$ is a natural factor which we discuss in the following paragraph. Indeed, it is not necessary to provide for $\bar{\delta}_t$ for $t > 1$, as these symbols will inextricably tick down to δ_1 and will have to annihilate with $\bar{\delta}_1$.

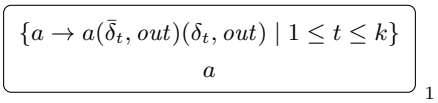

Fig. 5. The Anti-Bomber.

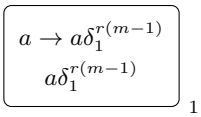

Fig. 6. The Delta Wall.

The idea behind the factor r is that other strategies may try to beat the Delta Wall by having rules emitting a large number of δ_t. However, the more such symbols are emitted, the lower the probability that they end up in the same valkyrie, meaning that the Delta Wall will have a high degree of resilience, even for smaller values of r, like 3 or even 2.

Another protective strategy consists in wrapping the valkyrie in a couple of additional membranes. In this way, the valkyrie can tolerate several membrane dissolutions without being thrown out of the game.

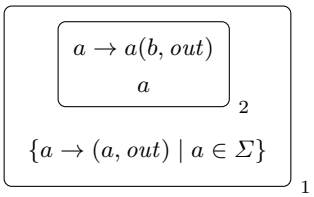

Fig. 7. The 2-layer Onion.

Remark that the Onion will have trouble emitting δ_t symbols. Firstly, the rules in the valkyrie are not allowed to contain δ_t in their left-hand sides, so the rules of the shape $\delta_t \to (\delta_t, out)$ are not allowed. If instead of emitting δ_t a different symbol d is used, then it is necessary to convert d into δ_t at some moment. If such conversion rules only appear in the outermost membrane of the Onion, then dissolving that membrane will remove those rules. On the other hand, including such rules in every layer of the Onion will create the possibility that an inner level inadvertently causes the dissolution of an outer level. Therefore, a strategy which may work best with the Onion would consist in relying on the relative scarcity of the symbols in Σ and in trying to destabilize the other valkyries by forcing some unexpected symbols into their membranes. For this to work, $|\Sigma|$ has to be sufficiently small.

Finally, the last tentative strategy we present in this section is the Bombshell. The idea is to have multiple inner membranes which are all released into the skin membrane of the tournament, therefore creating a family of cooperating agents belonging to the same team. This allows for exceeding the total number of valkyies m, but comes at the price of dissolving a membrane, which will be reflected in the final score.

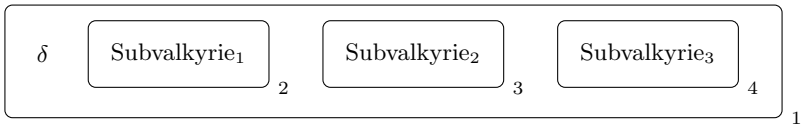

Fig. 8. The scheme of a 3-charge Bombshell.

5 Future Work and Perspectives

The immediate future work is setting up Queens of the Hill tournaments between valkyries designed by teams of students taking a course in formal languages or in natural computing. Queens of the Hill can be seen as a programming exercise in the language of an unconventional model of computing with a concrete goal: attacking all other contestants and surviving against their attacks for as long as possible. This context can also be used to introduce questions from theoretical biology about evolution and robustness, somewhat in the spirit of [17, 18]. We remark that such exercises are quite widespread in teaching of multi-agent systems and autonomic systems, as NetLogo-related resources illustrate [19].

To us as teachers and researches (enseignants-chercheurs as they say in French), Queens of the Hill is a great opportunity to employ our students' creativity to push the frontiers of what can be done with P systems. In the particular setup we describe in this paper we focus on transition P systems with non-elementary membrane dissolution and some *r*udimentary matter-antimatter annihilation rules, a model directly supported by P-Lingua. Obviously, other variants of P systems and the corresponding simulator engines can be used as the underlying formalism, thereby stimulating the students' interest in these other variants. Among the salient examples we cite kernel P systems [7, 12] and cP systems [8, 9, 14].

While valkyrie P systems are in principle computationally complete (Section 3), individual computational steps are less expressive than register machine instructions, meaning that designing valkyries *de facto* explores the capabilities of a less powerful language. Furthermore, good valkyrie design will require estimating the probabilities of different branches of computation, which will encourage the students to delve deeper into probability theory.

The setup we propose in this paper is at an early stage. We will most likely need to further tune the values of the parameters in Table 1, and probably also adjust

some aspects of the definitions of valkyrie P systems as well as of the tournament in order to avoid trivial edge cases and incite the design of complex strategies. An important question is the relevance of the scoring function score(Π_j) introduced in Section 2.3—other scoring functions may better capture the results of the competition. It is also possible to define scoring functions measuring the production of a certain set of symbols, thereby shifting the focus away from membrane dissolution entirely. One could also think about tracking the origins of the symbols, which could in principle allow saying which valkyrie dissolved which other valkyrie. This would require a rather fine analysis of the computations.

On a final note, we remark that while Queens of the Hill tournaments are directly inspired by Core Wars, the P system context shuffles things up quite a bit. In particular, data is secondary in Core Wars, and warriors interact by writing over each other's code. If we take the rules to be the program in P systems, then the programs of the valkyries are immutable in the sense that individual rules cannot be modified[1]. However, it is possible to instantly and entirely erase parts of their programs by dissolving the corresponding membranes. Furthermore, P systems are inherently non-deterministic, which we translate into a probabilistic framework, while warriors in Core Wars are deterministic. These remarks make us believe that Queens of the Hill tournaments have great potential waiting to be explored.

Acknowledgements

The authors would like to thank the Organizing Committee of the 19th Brainstorming Week on Membrane Computing[2] (BWMC 2023) for organizing this fruitful event.

References

1. Artiom Alhazov, Rudolf Freund, and Sergiu Ivanov. Polymorphic P systems: A survey. Technical report, Bulletin of the International Membrane Computing Society, December 2016.
2. Lucie Ciencialová, Erzsébet Csuhaj-Varjú, Ludek Cienciala, and Petr Sosík. P colonies. *J. Membr. Comput.*, 1(3):178–197, 2019.
3. Erzsébet Csuhaj-Varjú and György Vaszil. P automata or purely communicating accepting P systems. In Gheorghe Păun, Grzegorz Rozenberg, Arto Salomaa, and Claudio Zandron, editors, *Membrane Computing, International Workshop, WMC-CdeA 2002, Curtea de Arges, Romania, August 19-23, 2002, Revised Papers*, volume 2597 of *Lecture Notes in Computer Science*, pages 219–233. Springer, 2002.
4. Ignacio Pérez-Hurtado et al. The P-Lingua Website. `http://www.p-lingua.org/wiki/index.php/Main_Page`, Retrieved in May 2023.

[1] This may be an opportunity for plugging in polymorphic P systems and other P system variants with dynamic rules [1].

[2] `http://www.gcn.us.es/19bwmc`

5. Rudolf Freund, Sergiu Ivanov, and Ludwig Staiger. Going beyond turing with P automata: Regular observer ω-languages and partial adult halting. *Int. J. Unconv. Comput.*, 12(1):51–69, 2016.
6. Manuel García-Quismondo, Rosa Gutiérrez-Escudero, Miguel A. Martínez-del-Amor, Enrique Orejuela-Pinedo, and Ignacio Pérez-Hurtado. P-lingua 2.0: A software framework for cell-like P systems. *Int. J. Comput. Commun. Control*, 4(3):234–243, 2009.
7. Marian Gheorghe, Rodica Ceterchi, Florentin Ipate, Savas Konur, and Raluca Lefticaru. Kernel P systems: From modelling to verification and testing. *Theor. Comput. Sci.*, 724:45–60, 2018.
8. Alec Henderson and Radu Nicolescu. Actor-like cP systems. In Thomas Hinze, Grzegorz Rozenberg, Arto Salomaa, and Claudio Zandron, editors, *Membrane Computing - 19th International Conference, CMC 2018, Dresden, Germany, September 4-7, 2018, Revised Selected Papers*, volume 11399 of *Lecture Notes in Computer Science*, pages 160–187. Springer, 2018.
9. Alec Henderson, Radu Nicolescu, and Michael J. Dinneen. Solving a PSPACE-complete problem with cP systems. *J. Membr. Comput.*, 2(4):311–322, 2020.
10. Bulletin of the International Membrane Computing Society (IMCS). `http://membranecomputing.net/IMCSBulletin/index.php`.
11. Ilmari Karonen. The beginners' guide to Redcode. `https://vyznev.net/corewar/guide.html`, version 1.23, August 11, 2020.
12. Savas Konur, Laurentiu Mierla, Florentin Ipate, and Marian Gheorghe. kPWorkbench: A software suit for membrane systems. *SoftwareX*, 11:100407, 2020.
13. Ivan Korec. Small universal register machines. *Theor. Comput. Sci.*, 168(2):267–301, 1996.
14. Radu Nicolescu and Alec Henderson. An introduction to cP systems. In Carmen Graciani Díaz, Agustín Riscos-Núñez, Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa, editors, *Enjoying Natural Computing - Essays Dedicated to Mario de Jesús Pérez-Jiménez on the Occasion of His 70th Birthday*, volume 11270 of *Lecture Notes in Computer Science*, pages 204–227. Springer, 2018.
15. Gheorghe Păun. Computing with membranes. *Journal of Computer and System Sciences*, 61(1):108–143, 2000.
16. Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa, editors. *The Oxford Handbook of Membrane Computing*. Oxford University Press, 2010.
17. Rémi Segretain, Sergiu Ivanov, Laurent Trilling, and Nicolas Glade. A methodology for evaluating the extensibility of boolean networks' structure and function. In Rosa M. Benito, Chantal Cherifi, Hocine Cherifi, Esteban Moro, Luis Mateus Rocha, and Marta Sales-Pardo, editors, *Complex Networks & Their Applications IX - Volume 2, Proceedings of the Ninth International Conference on Complex Networks and Their Applications, COMPLEX NETWORKS 2020, 1-3 December 2020, Madrid, Spain*, volume 944 of *Studies in Computational Intelligence*, pages 372–385. Springer, 2020.
18. Rémi Segretain, Laurent Trilling, Nicolas Glade, and Sergiu Ivanov. Who plays complex music? On the correlations between structural and behavioral complexity measures in sign Boolean networks. In *21st IEEE International Conference on Bioinformatics and Bioengineering, BIBE 2021, Kragujevac, Serbia, October 25-27, 2021*, pages 1–6. IEEE, 2021.
19. Uri Wilensky. NetLogo. `http://ccl.northwestern.edu/netlogo/`. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL.