Higher-order Chemical Model of Computation

J.-P. Banâtre[1], P. Fradet[2] and Y. Radenac[1]

[1] IRISA, Campus de Beaulieu, 35042 Rennes Cedex, France
(jbanatre,yradenac)@irisa.fr
[2] INRIA Rhône-Alpes, 655 avenue de l'Europe, 38330 Montbonnot, France
Pascal.Fradet@inria.fr

The chemical reaction metaphor has been discussed in various occasions in the literature. This metaphor describes computation in terms of a chemical solution in which molecules (representing data) interact freely according to reaction rules. Chemical solutions are represented by multisets. Computation proceeds by rewritings of the multiset which consume and produce new elements according to reaction conditions and transformation rules.

To the best of our knowledge, the Gamma formalism was the first "chemical model of computation" proposed as early as in 1986 [4] and later extended in [5]. A Gamma program is a collection of reaction rules acting on a multiset of basic elements. A reaction rule is made of a condition and an action. Execution proceeds by replacing elements satisfying the reaction condition by the elements specified by the action. The result of a Gamma program is obtained when a stable state is reached, that is to say, when no reaction can take place anymore.

Let's illustrate this style of programming of Gamma with three small examples: the maximum, the prime numbers and the majority element. The reaction $max = \textbf{replace}\, x, y \,\textbf{by}\, x \,\textbf{if}\, x \geq y$ computes the maximum element of a non empty set. The reaction replaces any couple of elements x and y such that the reaction condition ($x \geq y$) holds by x. This process goes on till a stable state is reached, that is to say, when only the maximum element remains. The reaction $primes = \textbf{replace}\, x, y \,\textbf{by}\, y \,\textbf{if}\, multiple(x, y)$ computes the prime numbers lower or equal to a given number N when applied to the multiset of all numbers between 2 and N ($multiple(x, y)$ is true if and only if x is multiple of y). The majority element of a multiset M is an element which occurs more than $card(M)/2$ times in the multiset. Assuming that such an element exists, the reaction $maj = \textbf{replace}\, x, y \,\textbf{by}\, \{\} \,\textbf{if}\, x \neq y$ yields a multiset which only contains instances of the majority element just by removing pairs of distinct elements. Let us emphasize the conciseness and elegance of these programs. Nothing had to be said about the order of evaluation of the reactions. If several disjoint pairs of elements satisfy the condition, the reactions can be performed in parallel.

Gamma makes it possible to express programs without artificial sequentiality. By artificial, we mean sequentiality only imposed by the computation model and unrelated to the logic of the program. This allows the programmer to describe programs in a very abstract way. In some sense, one can say that Gamma programs express the very idea of an algorithm without any unnecessary linguistic idiosyncrasies. The interested reader may find in [5] a long series of examples (string processing problems, graph problems, geometry problems, *etc.*) illustrating the Gamma style of programming.

The chemical reaction model has served as the basis of a number of extensions. In [1], the reader may find a review of contributions related to the chemical reaction model. Let's mention here only two of them. The chemical abstract machine (or *cham*) [6] extends Gamma with the notions of membrane and airlock mechanism. The cham was proposed to describe the operational semantics of process calculi in a chemical way. Gamma has also been extended to handle data structures and types [8]. This work led to the definition of a richer type system for C named *shape types* [7]. Even if Gamma is a quite high-level and abstract language, the metaphor provides new points of view on problems and so it inspires new and original solutions to these problems.

In our recent work [3], we have extended Gamma to higher-order (programs may be parameters of any program). In this model, called the γ-calculus, reaction rules are themselves molecules that can be consumed like any other molecule. The model comes in two flavors: a minimal version called γ_0 and a more expressive version called γ_{cn}. In γ_0, reaction rules have no reaction condition and can only replace one molecule at a time (no atomic catch). Despite these limitations, it is Turing-complete. It shows that the principles of chemical models lie in the Brownian motion and the locality of reactions. However, γ_0 is not very convenient to write programs. The γ_{cn}

model includes the reaction condition and the atomic catch. Both extensions really increase the expressive power of the γ-calculus. Here is an example in γ_{cn} that computes the largest prime number lower than 10:

$$\langle\langle 2,3,4,5,6,7,8,9,10,prime\rangle,\ \mathbf{replace}\langle prime,\omega\rangle\ \mathbf{by}\langle\omega,max\rangle\rangle$$

where $\langle\ldots\rangle$ represents solutions and *prime* and *max* the previous reaction rules (which are now molecules in the solution). The reaction rule that matches the sub-solution (higher-order property) cannot react since the prime numbers are not computed. So first, reactions inside the sub-solution occur. When it is inert, the reaction rule spots the *prime* reaction rule in the solution and matches ω for the rest of the solution which corresponds to all the computed prime numbers. It then replaces the sub-solution by the prime numbers (ω) and the *max* reaction rule which starts computing the maximum.

The article [2] illustrates the adequacy of the chemical paradigm to the description of autonomic systems. The higher-order property of γ_{cn} is used to program coordination. Reactions may be generated or removed according to predefined situations. The autonomy property is based on the natural self-organization of a (higher-order) chemical solution. A reaction rule in a solution may be seen as an invariant for the solution: if it is not satisfied, the reaction rule transforms the solution such that it is satisfied (it self-organizes). The solution will reach a stable state that satisfies the invariant. If new molecules are added to the solution, it may break the invariant: it perturbs the stable state. The reaction rule will then react to reach a new stable state.

Let us point out another interesting extension. The multiplicity of a molecule is the number of copy of it in a solution. The extension consists in considering negative and infinite multiplicities. A solution is interpreted as a table (representing the characteristic function of the solution) which gives the multiplicity for any molecule in the solution. When molecules are added or removed from the solution, the table is updated accordingly. For example, if a solution contains a molecule m with a multiplicity –3, and a copy of M is added to the solution, then the new multiplicity of M is –2. Thus negative multiplicities are viewed as destroyers. A molecule may also have an infinite multiplicity: whatever the number of copies of this molecule react, the multiplicity of the molecule is still infinite. Thus molecules with infinite multiplicity are viewed as persistent molecules.

This higher-order chemical model can be used as a coordination language. It allows to express coordination in a simple way for the application programmer through the chemical metaphor. It provides also a useful model to program operating systems which can exploit the independence of reactions, the non-determinism and higher-order properties to adapt the execution of applications in complex systems.

References

1. Jean-Pierre Banâtre, Pascal Fradet, and Daniel Le Métayer. Gamma and the chemical reaction model: Fifteen years after. In *Multiset Processing*, volume 2235 of *LNCS*, pages 17–44. Springer-Verlag, 2001.
2. Jean-Pierre Banâtre, Pascal Fradet, and Yann Radenac. Chemical specification of autonomic systems. In *Proc. of the 13th Int. Conf. on Intelligent and Adaptive Systems and Software Engineering (IASSE'04)*, 2004.
3. Jean-Pierre Banâtre, Pascal Fradet, and Yann Radenac. Principles of chemical programming. In *Fifth International Workshop on Rule-Based Programming (RULE'04)*. Electronic Notes in Theoretical Computer Science 2005, June 2004.
4. Jean-Pierre Banâtre and Daniel Le Métayer. A new computational model and its discipline of programming. Technical Report RR0566, INRIA, September 1986.
5. Jean-Pierre Banâtre and Daniel Le Métayer. Programming by multiset transformation. *Communications of the ACM (CACM)*, 36(1):98–111, January 1993.
6. Gérard Berry and Gérard Boudol. The chemical abstract machine. *Theoretical Computer Science*, 96:217–248, 1992.
7. Pascal Fradet and Daniel Le Métayer. Shape types. In *Proc. of Principles of Programming Languages, POPL'97*, pages 27–39. ACM Press, January 1997.
8. Pascal Fradet and Daniel Le Métayer. Structured gamma. *Science of Computer Programming*, 31(2–3):263–289, 1998.