Guarded Recursion and Mathematical Operational Semantics

Mauro Jaskelioff and Neil Ghani

University of Nottingham

Operational semantics gives meaning to terms in a programming language by defining a transition relation which represents execution steps. In structural operational semantics (SOS) this transition relation is given by a set of rules defined on the structure of the terms of the language. But, when is a collection of rules satisfactory, in the sense that it defines a well-behaved operational semantics?

Before the introduction of Mathematical Operational Semantics by Turi [2], there were many attempts to conceive a theory of operational semantics in the form of syntactic rule formats. These results were specific to a particular type of transition relation, strongly syntactic, and therefore difficult to generalize and adapt to other settings. Turi stripped operational semantics to its bare bones, ignoring concrete syntax to focus on its structure and, as a result, giving us a clean categorical reformulation of SOS. Under this interpretation, the SOS of a language is given by a distributive law of syntax over behaviour. If the operations in the language are given by a signature functor Σ and the behaviour of the system by a behaviour functor B, the distributive law can be constructed from abstract operational rules, that is, natural transformations:

$$\Sigma(Id \times B) \to BT_\Sigma$$

where T_Σ is the free monad over Σ.

Apart from the elegant and language-independent formulation of SOS in this setting, Turi's approach has the benefit of automatically ensuring properties such as the congruence of bisimulation.

In SOS, recursive programs are usually described via a recursive specification, i.e. a system of equations:

$$\begin{aligned} x_1 &= t_1 \\ &\vdots \\ x_n &= t_n \end{aligned}$$

where each t_i is a term consisting of operations in a previously defined language and variables x_i.

In order to ensure that there exists a solution to such a system of equations, and that the solution is unique, these equations are required to be guarded. For process algebras like ACP [1], an equation is said to be guarded if each t_i is equivalent to a term of the form $a_1.p_1 + \ldots + a_n.p_n + b_1 + \ldots + b_m$, where x_i may occur freely in p_j, and a_j, b_k are atomic actions.

Turi gave an abstract solution to guarded equations, constructing a coalgebra $T_\Sigma X \to BT_\Sigma X$ where X is the set of recursively defined identifiers [3]. However, in this solution, no distributive law is obtained, so it is not clear how this solution fits with the rest of the framework. Here, we take a different approach and regard the recursive definitions as additional operations of the language. Instead of a set of identifiers, we will have a signature functor Ω for the operations defined by recursive equations. The advantage of this approach is that now we can obtain a distributive law for the complete language –ensuring that all the properties of the theory of mathematical operational semantics are still valid– and also, we can consider parameter-passing recursive programs.

Given a system of equations $e\colon \Omega \to BT_{\Sigma+\Omega}$ and the semantics of the non-recursive part of the language $\rho_\Sigma\colon \Sigma(Id\times B) \to BT_\Sigma$, we obtain the semantics of the combined language as a new operational rule involving the operations from both signatures:

$$\begin{tikzcd}
\Sigma(Id \times B) \arrow[r, "\mathsf{inl}"] \arrow[d, "\rho_\Sigma"'] & (\Sigma + \Omega)(Id \times B) \arrow[d] & \Omega(Id \times B) \arrow[l, "\mathsf{inr}"'] \arrow[d, "\pi_1"] \\
BT_\Sigma \arrow[r, hook] & BT_{\Sigma+\Omega} & \Omega \arrow[l, "e"]
\end{tikzcd}$$

We considered recursive specifications $\Omega \to BT_{\Sigma+\Omega}$ motivated by the usual concrete representation of guarded equations, but we could have considered more general natural transformations $\Omega(Id \times B) \to BT_{\Sigma+\Omega}$, which are not very different from operational rules. Hence, although we can view guarded equations as providing different concrete syntax for describing operational behaviour, they are essentially abstract operational rules in disguise.

In conclusion, we showed how to add recursively defined operations to a SOS using the distributive law approach of Turi which results in bisimulation as a congruence. Tantalizingly, this shows that recursive programs are a reflection in syntax of operationally defined infinitary behaviour.

References

1. Jan A. Bergstra and Jan Willem Klop. Process algebra for synchronous communication. *Information and Control*, 60(1-3):109–137, 1984.
2. D. Turi and G.D. Plotkin. Towards a mathematical operational semantics. In *Proc. 12th LICS Conf.*, pages 280–291. IEEE, Computer Society Press, 1997.
3. Daniele Turi. Categorical modelling of structural operational rules: Case studies. In *Category Theory and Computer Science*, pages 127–146, 1997.