Models of Computation: Automata and Processes

J.C.M. Baeten P.J.L. Cuijpers B. Luttik
P.J.A. van Tilburg*

July 21, 2009

1 Introduction

Back in the 1940s the computer was invented. However, these machines were quite different from the computers we use today. They accepted a fixed string as input and produced output by means of a specific batch process. Thus, input was separated from the output. The theory of automata and formal language was devised to provide models of these systems to reason about them and even turned out to provide powerful models of computation in general.

Nowadays, computers are systems that interact with us but also each other; they are reactive systems. They no longer operate using a given procedure and are systems where output is no longer separated from the input; a single click can be the input and the result can be non-deterministic. For example, a search query on Google leads to different results every time.

Concurrency theory provides a model of computation similar to the model given by the theory of automaton and formal language, but focuses more on interaction. Concurrency theory has been split of the classical automaton theory a few decades ago and began its on separate development and deals with system in a concurrent setting.

The goal of the project is to investigate the integration of the two theories. The attempt reveals the differences and similarities between them. We use analogies to make the integration explicit. For example we study the pushdown automaton and Turing machine in a process-theoretic setting. The attempt at integration hopefully increases the understanding of both theories. This also has a practical side. The result can be integrated into one course in the undergraduate curriculum, providing students with an increased understanding of concurrent, reactive systems.

In this work we study three levels in the Chomsky hierarchy: we study the notation of a regular process, a pushdown process and a computable process. We look at process-theoretic analogies of classic results from automata theory

*Division of Computer Science, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands,
e-mail: {j.c.m.baeten,p.j.l.cuijpers,s.p.luttik,p.j.a.v.tilburg}@tue.nl

(see also [10], [11]) and see if they still hold and if not, what extra conditions are needed to make it hold. In our work we consider the process algebra TCP_τ and several fragments; refer to [1] for details. We also use *structural operational semantics* [13] to associate labelled transition systems to the process expressions.

Given a set of a (possible infinite) labelled transition system, we can divide out different equivalence relations on this set. Dividing out language equivalence throws away too much information, as the moments where choices are made are totally lost, and behaviour that does not lead to a final state is ignored. An equivalence relation that keeps all relevant information, and has many good properties is *branching bisimulation* as proposed by Van Glabbeek and Weijland [9, 6]. For a detailed definition and motivations to use branching bisimulation as the preferred notation of equivalence, see [7]. Note that in between language equivalence and bisimulation equivalence, there are also several other equivalence relations (see [8]).

2 Regular Processes

A computer with a fixed-size, finite memory is just a finite control. This can be modelled by a *finite* automaton. In automata theory the finite automaton has a central place. Finite automata accept so-called *regular languages*: language equivalence classes of a finite automaton.

In our work, we relate a finite automaton to a (regular) process given by a finite recursive specification over a subtheory of TCP_τ, called BSP_τ, which only allows for deadlock, successful termination, action prefixing and choice.

This can be viewed as the process-theoretic counterpart of the result from the theory of automata and formal languages that states that every language accepted by a finite automaton is generated by a so-called *right-linear* grammar. We define a *regular process* as a bisimulation equivalence class of a finite, nondeterministic automaton.

Note that unlike in automata and formal language theory where every language can be given by a regular expression, not every regular process is given by a regular expression, see [3].

3 Pushdown and context-free processes

As an intermediate between the notions of Finite Automaton and Turing Machine, the theory of automata and formal languages treats the notion of pushdown automaton, which is a finite automaton with a stack as memory.

We consider the process-theoretic version of the standard result in the theory of automata and formal languages that the set of pushdown languages coincides with the set of languages generated by context-free grammars. As the process-theoretic counterparts of context-free grammars we consider recursive specifications in the subtheory TSP_τ of TCP_τ, which is obtained from BSP_τ by adding sequential composition.

That the notion of TSP_τ recursive specification naturally corresponds with with the notion of context-free grammar is confirmed a theorem. We define a *pushdown language* as the language of the transition system associated with a pushdown automaton, and a *pushdown process* as a branching bisimilarity class of labelled transition systems containing a labelled transition system associated with a pushdown automaton.

When considering language equivalence, the class of pushdown languages coincide with the class of *context-free languages*. This is not the case when considering branching bisimulation. In fact, we show that there are pushdown processes that are not recursively definable in TSP_τ, and that there are also TSP_τ recursive specifications that define non-pushdown processes can be defined. We present a restriction on pushdown automata and a restriction on TSP_τ recursive specifications that enable us to retrieve the desired equivalence: we prove that the set of so-called *popchoice-free* pushdown processes corresponds with the set of processes definable by a *transparency-restricted* TSP_τ recursive specification.

For a different approach that requires none of the above restrictions, see [4]. However, note that it does require to drop the branching bisimulation equivalence in favour of a weaker equivalence called contrasimulation [14].

4 Basic parallel processes

A similar class to the pushdown processes that we consider is the class of the *basic parallel processes*. Here the sequential composition replaced by the parallel operator that just allows for interleaving, no communication.

We have a result that a bag (or multiset) is required instead of a stack to reach an analogous result. We can now characterise a *parallel pushdown process* as a process that can be written as a regular processes communicating with a bag. See [5] for the details.

Moller has shown that there are basic parallel processes that are not context-free nor pushdown processes. He even shows that there are parallel pushdown processes that are not context-free nor pushdown nor basic parallel, making all classes really distinct. See [12] for the counter examples.

5 Computable processes

Finally, we discuss the largest class of processes considered in our work called computable processes. We define a *computable process* as a bisimulation equivalence class of a computable transition system.

We show the following characterisation of computable processes (a precursor of this result was presented in [2]): a process is computable iff it can be written as a regular process communicating with two stacks. The two stacks emulate the tape present in the classical notation of a Turing machine.

Acknowledgements

The research of Van Tilburg was supported by the project "Models of Computation: Automata and Processes" (nr. 612.000.630) of the Netherlands Organization for Scientific Research (NWO).

References

[1] J.C.M. Baeten, T. Basten, and M.A. Reniers. *Process Algebra (Equational Theories of Communicating Processes)*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2009.

[2] J.C.M. Baeten, J.A. Bergstra, and J.W. Klop. On the consistency of Koomen's fair abstraction rule. *Theoretical Computer Science*, 51:129–176, 1987.

[3] J.C.M. Baeten, F. Corradini, and C.A. Grabmayer. A characterization of regular expressions under bisimulation. *Journal of the ACM*, 54(2):6.1–28, 2007.

[4] J.C.M. Baeten, P.J.L. Cuijpers, and P.J.A. van Tilburg. A context-free process as a pushdown automaton. In F. van Breugel and M. Chechik, editors, *Proceedings CONCUR'08*, number 5201 in Lecture Notes in Computer Science, pages 98–113, 2008.

[5] J.C.M. Baeten, P.J.L. Cuijpers, and P.J.A. van Tilburg. A basic parallel process as a parallel pushdown automaton. In D. Gorla and T. Hildebrandt, editors, *Proceedings EXPRESS'08*, Electronic Notes in Theoretical Computer Science, 2009. To appear.

[6] T. Basten. Branching bisimilarity is an equivalence indeed! *Information Processing Letters*, 58(3):141–147, 1996.

[7] R.J. van Glabbeek. What is Branching Time Semantics and why to use it? *Bulletin of the EATCS*, 53:190–198, 1994.

[8] R.J. van Glabbeek. The Linear Time – Branching Time Spectrum I. In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of Process Algebra*, pages 3–99. Elsevier, 2001.

[9] R.J. van Glabbeek and W.P. Weijland. Branching time and abstraction in bisimulation semantics. *Journal of the ACM*, 43(3):555–600, 1996.

[10] J.E. Hopcroft, R. Motwani, and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Pearson, 2006.

[11] P. Linz. *An Introduction to Formal Languages and Automata*. Jones and Bartlett, 2001.

[12] F. Moller. Infinite results. In U. Montanari and V. Sassone, editors, *Proceedings CONCUR'96*, number 1119 in Lecture Notes in Computer Science, pages 195–216, 1996.

[13] G.D. Plotkin. The origins of structural operational semantics. *Journal of Logic and Algebraic Programming*, 60(1):3–16, 2004.

[14] M. Voorhoeve and S. Mauw. Impossible futures and determinism. *Information Processing Letters*, 80:51–58, 2001.