On Computability of Data Word Functions Defined by Transducers[⋆]

Léo Exibard[1,2]

[1] Université Libre de Bruxelles, Brussels, Belgium
leo.exibard@ulb.ac.be
[2] Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France

1 Program Synthesis

Program synthesis aims at deriving, in an automatic way, a program that fulfils a given specification. This setting is very appealing when for instance the specification describes, in some abstract formalism (an automaton or ideally a logic), important properties that the program must satisfy. The synthesised program is then *correct-by-construction* with regards to those properties. It is particularly important and desirable for the design of safety-critical systems with hard dependability constraints, which are notoriously hard to design correctly.

Program synthesis is hard to realise for general-purpose programming languages but important progress has been made in the automatic synthesis of *reactive systems*. Those systems continuously receive input signals to which they must react by producing output signals, and are not assumed to terminate. Their executions are modelled as infinite words over the alphabets of input and output signals. A specification is thus a set of pairs (in,out), where in and out are infinite words, such that out is a legitimate output for in. Most methods for reactive system synthesis only work for *synchronous* systems over *finite* input and output alphabets Σ and Γ. In this synchronous setting, input and output alternate, and *implementations* of such a specification are defined by means of *synchronous* transducers, which are Büchi automata with transitions of the form (q, σ, γ, q'), expressing that in state q, when getting input $\sigma \in \Sigma$, the machine outputs $\gamma \in \Gamma$ and moves to state q'. We aim at building *deterministic* implementations, in the sense that the output γ and state q' uniquely depend on q and σ. The realisability problem of specifications given as synchronous non-deterministic transducers, by implementations defined by synchronous deterministic transducers is known to be decidable [8,12]. In this paper, we are interested in the *asynchronous* setting, in which transducers can produce none or several outputs at once every time some input is read, i.e., transitions are of the form (q, σ, w, q') where $w \in \Gamma^*$. However, this generalisation makes the realisability problem undecidable [1,6].

⋆ This presentation is about the results contained in the eponym article, co-written with Emmanuel Filiot and Pierre-Alain Reynier, which has been accepted at FoSSaCS 2020. The present abstract is taken from its introduction.

2 Synthesis of Transducers with Registers

In this setting, the set of signals is considered to be finite. This assumption is not realistic in general, as signals may come with unbounded information (e.g. process ids) that we call here *data*. To address this limitation, recent works have considered the synthesis of reactive systems processing *data words* [10,4,9,5]. Data words are infinite words over an alphabet $\Sigma \times \mathcal{D}$, where Σ is a finite set and $\mathcal{D}$ is a possibly infinite countable set. To handle data words, just as automata have been extended to *register automata*, transducers have been extended to *register transducers*. Those transducers are equipped with a finite set of registers in which they can store data and with which they can compare data for equality or inequality. While the realisability problem of specifications given as synchronous non-deterministic register transducers ($\mathsf{NRT}_{\mathsf{syn}}$) by implementation defined by synchronous deterministic register transducers ($\mathsf{DRT}_{\mathsf{syn}}$) is undecidable, decidability is recovered for specifications defined by universal register transducers and by giving as input the number of registers the implementation must have [5,10].

3 Computable Implementations

In the previously mentioned works, both for finite or infinite alphabets, target implementations are deterministic transducers. By definition, they use only a constant amount of memory (assuming data have size $O(1)$). While this makes sense with regards to memory-efficiency, some problems turn out to be undecidable: realisability of $\mathsf{NRT}_{\mathsf{syn}}$ specifications by $\mathsf{DRT}_{\mathsf{syn}}$, or, in the finite alphabet setting, when both the specification and implementation are asynchronous.

Here, we study computable implementations, in the sense of (partial) functions f of data ω-words computable by some Turing machine M that has an infinite input $x \in \mathrm{dom}(f)$, and produces longer and longer prefixes of the output $f(x)$ as it reads longer and longer prefixes of the input x. Therefore, such a machine produces the output $f(x)$ in the limit. We denote by TM the class of Turing machines computing functions in this sense. As an example, consider the function f that takes as input any data ω-word $u = (\sigma_1, d_1)(\sigma_2, d_2)\dots$ and outputs $(\sigma_1, d_1)^\omega$ if d_1 occurs at least twice in u, and otherwise outputs u. This function is not computable, as an hypothetic machine could not output anything as long as d_1 is not met a second time. However, the following function g is computable. It is defined only on words $(\sigma_1, d_1)(\sigma_2, d_2)\dots$ such that $\sigma_1\sigma_2\cdots \in ((a+b)c^*)^\omega$, and transforms any (σ_i, d_i) by (σ_i, d_1) if the next symbol in $\{a, b\}$ is an a, otherwise it keeps (σ_i, d_i) unchanged. To compute it, a TM would need to store d_1, and then wait until the next symbol in $\{a, b\}$ is met before outputting something. Since the finite input labels are necessarily in $((a + b)c^*)^\omega$, this machine will produce the whole output in the limit. Note that g cannot be defined by any deterministic register transducer, as it needs unbounded memory to be implemented.

However, already in the finite alphabet setting, the problem of deciding if a specification given as some non-deterministic synchronous transducer is realisable by some computable function is open. The particular case of realisability by computable functions of universal domain (the set of all ω-words) is known to be decidable [7]. In the asynchronous setting, the undecidability proof of [1] can be easily adapted to show the undecidability of realisability of specifications given by non-deterministic (asynchronous) transducers by computable functions.

4 Continuity and Computability

We can equip the set of infinite words with the Cantor distance: for two infinite words u, v, we let $d(u, v) = 0$ if $u = v$ and $d(u, v) = 2^{-|u \wedge v|}$ otherwise, where $u \wedge v$ denotes the longest common prefix of u and v and $|w|$ denotes the length of w. This allows to define a notion of continuity for functions over infinite words. Note that the notion of continuity with regards to Cantor distance is not new, and for rational functions over finite alphabets, it was already known to be decidable [13]. Interestingly, continuity coincides with computability for functions defined by finite transducers; it can moreover be characterised by the exclusion of a pattern in the transducer [2]. We here show that this correspondance extends to the case of functions over data words defined by register transducers, and so does the corresponding pattern characterisation (cf Figure 1).

$$\begin{aligned}
&\exists u, v \in (\Sigma \times \mathcal{D})^*,\\
&\exists u', u'', v', v'' \in (\Gamma \times \mathcal{D})^*,\\
&\exists w \in (\Sigma \times \mathcal{D})^\omega,\\
&\exists w'' \in (\Gamma \times \mathcal{D})^\omega,\\
&\exists q \in Q, \exists q_f \in F,\\
&\exists \mu, \tau : R \to \mathcal{D},\\
&\mathsf{mismatch}(u', u'') \vee\\
&v'' = \varepsilon \wedge \mathsf{mismatch}(u', u''w'')
\end{aligned}$$

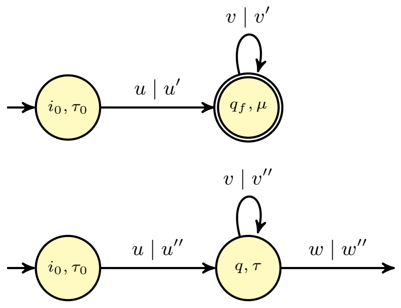

Fig. 1. A pattern characterising non-continuity of functions defined by a nondeterministic register transducer, where $\mathsf{mismatch}(u, v)$ holds whenever there exists a position $1 \leq i \leq \min(|u|, |v|)$ such that $u[i] \neq v[i]$.

Moreover, we show that there exists a bound on the number of distinct data appearing in the pattern, which allows us to reduce to the finite alphabet case. To this end, we use the fact that register transducers, as their automata counterpart, enjoy an "indistinguishability property": given a transducer T with r registers, for any data word, there exists a data word with at most $r + 1$ data on which T behaves the same. This is due to the fact that, when a transducer receives a new data and stores it, it has to erase the content of some of its registers. The above pattern hence yields a PSPACE algorithm to decide whether a function given as an NRT is computable; we then prove that such complexity is optimal by reducing to the emptiness problem of register automata.

5 Functional Specifications

A specification is in general a relation from inputs to outputs. If it is a function, we call it functional. Due to the negative results about the synthesis of computable functions from non-functional specifications, we focus on functional specifications and address the following question: given a functional specification over data ω-words, is this function "implementable", where "implementable" means "being computable by some Turing machine". Moreover, if it is implementable, how to automatically generate an algorithm that computes it? Besides, how to decide whether a specification is functional? Here, specifications are modelled as asynchronous register transducers (NRT). Asynchrony allows for more expressive power, but is a source of technical challenges. The above mentioned indistinguishability property allows us to reduce the functionality problem over data words to a functionality problem for finite transducers, modulo an exponential blowup in the number of registers; we provide a matching PSpace lower complexity bound. As a consequence, deciding whether two functions defined by NRT are equal on the intersection of their domains is PSpace-complete. We also show that the class of functions defined by NRT is effectively closed under composition.

We conclude by noticing that, as for the bounded synthesis problem studied in [5], the test-free restriction yields polynomial complexity bounds for checking functionality and continuity.

References

1. Carayol, A., Löding, C.: Uniformization in Automata Theory. In: Proceedings of the 14th Congress of Logic, Methodology and Philosophy of Science Nancy, July 19-26, 2011. pp. 153–178. London: College Publications (2014)
2. Dave, V., Filiot, E., Krishna, S.N., Lhote, N.: Deciding the computability of regular functions over infinite words. CoRR **abs/1906.04199** (2019), http://arxiv.org/abs/1906.04199
3. Durand-Gasselin, A., Habermehl, P.: Regular transformations of data words through origin information. In: Foundations of Software Science and Computation Structures - 19th International Conference, FOSSACS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings. pp. 285–300 (2016)
4. Ehlers, R., Seshia, S.A., Kress-Gazit, H.: Synthesis with identifiers. In: Proceedings of the 15th International Conference on Verification, Model Checking, and Abstract Interpretation - Volume 8318. pp. 415–433. VMCAI 2014 (2014)
5. Exibard, L., Filiot, E., Reynier, P.: Synthesis of data word transducers. In: 30th International Conference on Concurrency Theory, CONCUR 2019, August 27-30, 2019, Amsterdam, the Netherlands. pp. 24:1–24:15 (2019). https://doi.org/10.4230/LIPIcs.CONCUR.2019.24, https://doi.org/10.4230/LIPIcs.CONCUR.2019.24
6. Filiot, E., Jecker, I., Löding, C., Winter, S.: On equivalence and uniformisation problems for finite transducers. In: 43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, Rome, Italy. pp. 125:1–125:14 (2016), https://doi.org/10.4230/LIPIcs.ICALP.2016.125

7. Holtmann, M., Kaiser, L., Thomas, W.: Degrees of lookahead in regular infinite games. Logical Methods in Computer Science **8**(3) (2012). https://doi.org/10.2168/LMCS-8(3:24)2012, https://doi.org/10.2168/LMCS-8(3:24)2012
8. J.R. Büchi, L.H. Landweber: Solving sequential conditions finite-state strategies. Transactions of the American Mathematical Society **138**, 295–311 (1969)
9. Khalimov, A., Kupferman, O.: Register-bounded synthesis. In: 30th International Conference on Concurrency Theory, CONCUR 2019, August 27-30, 2019, Amsterdam, the Netherlands. pp. 25:1–25:16 (2019). https://doi.org/10.4230/LIPIcs.CONCUR.2019.25, https://doi.org/10.4230/LIPIcs.CONCUR.2019.25
10. Khalimov, A., Maderbacher, B., Bloem, R.: Bounded synthesis of register transducers. In: Automated Technology for Verification and Analysis, 16th International Symposium, ATVA 2018, Los Angeles, October 7-10, 2018. Proceedings (2018)
11. Neven, F., Schwentick, T., Vianu, V.: Finite state machines for strings over infinite alphabets. ACM Trans. Comput. Logic **5**(3), 403–435 (Jul 2004). https://doi.org/10.1145/1013560.1013562, http://doi.acm.org/10.1145/1013560.1013562
12. Pnueli, A., Rosner, R.: On the synthesis of a reactive module. In: ACM Symposium on Principles of Programming Languages, POPL. ACM (1989)
13. Prieur, C.: How to decide continuity of rational functions on infinite words. Theor. Comput. Sci. **276**(1-2), 445–447 (2002). https://doi.org/10.1016/S0304-3975(01)00307-3, https://doi.org/10.1016/S0304-3975(01)00307-3