A Note on Controllability of Deterministic Context-Free Systems

Tomáš Masopust

Institute of Mathematics, Academy of Sciences of the Czech Republic, Žižkova 22, 616 62 Brno, Czech Republic

Abstract

In this paper, we prove that the most important concept of supervisory control of discrete-event systems, the *controllability* property, is undecidable for two deterministic context-free languages K and L, where L is prefix-closed, even though K is a subset of L. If K is not a subset of L, the undecidability follows from the work by Sreenivas. However, the case where K is a subset of L does not follow from that work because it is decidable whether K and L are equivalent as shown by Sénizergues. Thus, our result completes this study. The problem is also mentioned as open in the PhD thesis by Griffin, who extended the supervisory control framework so that the specification language is modelled as a deterministic context-free language (compared to the classical approach where the specification is regular) and the plant language is regular. This approach is of interest because it brings an opportunity for more concise representations of the specification (as discussed, e.g., in the work by Geffert et al.) and, therefore, in some sense it treats the most interesting problem of the current supervisory control theory, the state-space explosion problem.

Key words: Discrete-event systems; Controllability; Deterministic Context-Free Systems; Decidability.

1 Introduction

In 1993, Sreenivas [11] has shown that the controllability property is undecidable for any systems for which inclusion is undecidable. However, it turned out later that there exist systems, such as deterministic pushdown automata [9,10], for which inclusion is undecidable, whereas equivalence is decidable. For such systems, it is undecidable whether $K \subseteq L$ (in what follows, we implicitly assume that K denotes a specification language, and L denotes a plant language which is prefix-closed by definition), however, if we have some additional knowledge that the languages are given so that $K \subseteq L$, then it follows from [11] that if controllability is decidable, then so is decidable the question whether $L \subseteq K$. Since $K \subseteq L$ is known, to decide whether $L \subseteq K$ results in the equivalence problem $K = L$, which is decidable. Thus, we cannot use the results of [11] to solve the problem whether controllability is decidable for two deterministic context-free languages where the specification language K is included in the plant language L.

Note that the inclusion may be known for many reasons. For instance, the languages are of some special form and the human operator is able to decide it (as in the case of the languages from the proof of Theorem 1) or, more generally, we can assume that there exists an oracle that decides containment for two deterministic context-free languages.

The problem has not yet been discussed in the literature, and it is also formulated as open in [4]. Therefore, this paper completes the study discussed in [11] by solving this problem. More specifically, we prove that even though the equivalence problem is decidable for deterministic context-free languages, controllability is undecidable for two deterministic context-free languages even though the containment of the specification language in the plant language is known. This result means that the undecidability of controllability is not just a by-product of the undecidability of containment as one might infer from Theorem 2.1 and Corollary 2.1 of [11].

Because of these undecidability results, the only possibilities which deserve consideration in supervisory control of discrete-event systems are: (i) the specification language K is regular and the plant language L is (deterministic) context-free, or (ii) the specification language K is deterministic context-free and the plant language L is regular. The later case (ii) has recently been treated in [3–5] while, as far as the author knows, the former approach (i) has not yet been treated in the literature at all. For this reason, a brief discussion concerning controllability can be found in the conclusion.

These approaches are of interest because they present an opportunity to treat the most interesting problem of the current supervisory control theory of discrete-event systems,

⋆ Corresponding author: T. Masopust. Tel. +420222090784, Fax. +420541218657.

Email address: masopust@math.cas.cz (Tomáš Masopust).

the state-space explosion problem, so that we can describe the plant language or the specification language by a more concise representation which is up to exponentially smaller when using deterministic pushdown automata instead of finite-state machines as discussed, e.g., in [2].

2 Preliminaries

In this paper, we assume that the reader is familiar with the basic notions and concepts of supervisory control of discrete-event systems based on the Ramadge-Wonham automata framework [1,7,12], and with the theory of automata and formal languages [8].

Let Σ be an *alphabet* (a finite nonempty set of events). The set Σ^* consisting of all finite words over Σ denotes the free monoid generated by Σ. The unit of Σ^* (the *empty word*) is denoted by ε. Any set $L \subseteq \Sigma^*$ is a *language* over Σ. The prefix closure of a language L over Σ is defined as the set $\overline{L} = \{w \in \Sigma^* \mid \exists u \in \Sigma^*, wu \in L\}$ of all prefixes of all its words. The language L is *prefix-closed* if $L = \overline{L}$.

The notion of a generator denotes a deterministic finite-state machine with a partial transition function.

A *generator* G is a construct $G = (Q, \Sigma, f, q_0, Q_m)$, where Q is the finite set of states, Σ is the input alphabet, $f : Q \times \Sigma \to Q$ is a partial transition function, $q_0 \in Q$ is the initial state, and $Q_m \subseteq Q$ is the set of marked states. In the usual way, f can be extended to a function from $Q \times \Sigma^*$ to Q by induction. The behavior of G is described in terms of languages. The language *generated* by G is defined as the set $L(G) = \{s \in \Sigma^* \mid f(q_0, s) \in Q\}$, and the language *marked* by G as the set $L_m(G) = \{s \in \Sigma^* \mid f(q_0, s) \in Q_m\}$. By definition, $L(G)$ is always prefix-closed.

A language L is *regular* if there exists a generator G such that $L_m(G) = L$.

Let G be a generator over an alphabet Σ, and let $\emptyset \neq K \subseteq L_m(G)$ be a language. Let $\Sigma_u \subseteq \Sigma$ denote the set of *uncontrollable* events. Language K is *controllable* with respect to $L(G)$ and Σ_u if $\overline{K}\Sigma_u \cap L(G) \subseteq \overline{K}$.

Recall that the notion of controllability plays the central role in supervisory control of discrete-event systems because there exists a supervisor or controller if and only if the specification language K is controllable with respect to $L(G)$ and Σ_u (and is $L_m(G)$-closed if non-prefix-closed specifications are considered, cf. [1]).

A *projection* $P : \Sigma^* \to \Sigma_0^*$, for alphabets Σ_0 and Σ with $\Sigma_0 \subseteq \Sigma$, is a homomorphism defined so that $P(a) = \varepsilon$, for $a \in \Sigma \setminus \Sigma_0$, and $P(a) = a$, for $a \in \Sigma_0$. The *inverse image* of P is denoted by $P^{-1} : \Sigma_0^* \to 2^{\Sigma^*}$, and defined so that $P^{-1}(y) = \{x \in \Sigma^* \mid P(x) = y\}$. These definitions can naturally be extended to languages.

The *synchronous product* of two languages $L_1 \subseteq \Sigma_1^*$ and $L_2 \subseteq \Sigma_2^*$ is defined as $L_1 \parallel L_2 = P_1^{-1}(L_1) \cap P_2^{-1}(L_2) \subseteq (\Sigma_1 \cup \Sigma_2)^*$, where $P_i : (\Sigma_1 \cup \Sigma_2)^* \to \Sigma_i^*$, for $i = 1, 2$, are projections.

To define deterministic context-free languages, we first need the concept of a pushdown automaton, which is a finite-state machine with a potentially infinite stack memory.

A *pushdown automaton* $\mathscr{M}$ is a construct $\mathscr{M} = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, where Q is the finite set of states, Σ is the input alphabet, Γ is the pushdown alphabet, δ is a transition function from $Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma$ to the set of finite subsets of $Q \times \Gamma^*$, $q_0 \in Q$ is the initial state, $Z_0 \in \Gamma$ is the initial pushdown symbol, and $F \subseteq Q$ is the set of accepting states. A *configuration* of $\mathscr{M}$ is a triple (q, w, γ), where q is the current state of $\mathscr{M}$, w is the unread part of the input, and γ is the current content of the pushdown (the leftmost symbol of γ is the topmost pushdown symbol). If $p, q \in Q$, $a \in \Sigma \cup \{\varepsilon\}$, $w \in \Sigma^*$, $\gamma, \beta \in \Gamma^*$, $Z \in \Gamma$, and $(p, \beta) \in \delta(q, a, Z)$, then $\mathscr{M}$ makes a move from $(q, aw, Z\gamma)$ to $(p, w, \beta\gamma)$, formally $(q, aw, Z\gamma) \vdash_{\mathscr{M}} (p, w, \beta\gamma)$. For simplicity, the initial pushdown symbol Z_0 appears only at the bottom of the pushdown during any computation, that is, if $(p, \beta) \in \delta(q, a, Z)$, then either β does not contain Z_0, or $\beta = \beta' Z_0$, where β' does not contain Z_0 and $Z = Z_0$. As usual, the reflexive and transitive closure of the relation $\vdash_{\mathscr{M}}$ is denoted by $\vdash_{\mathscr{M}}^*$. The *language accepted* by $\mathscr{M}$ is defined as $T(\mathscr{M}) = \{w \in \Sigma^* : (q_0, w, Z_0) \vdash_{\mathscr{M}}^* (q, \varepsilon, \gamma)$ for some $q \in F$ and $\gamma \in \Gamma^*\}$.

A pushdown automaton $\mathscr{M} = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ is *deterministic* if there is no more than one move the automaton can make from any configuration, that is, the following two conditions hold:

(1) $|\delta(q, a, Z)| \leq 1$, for all $a \in \Sigma \cup \{\varepsilon\}$, $q \in Q$, and $Z \in \Gamma$, and
(2) for all $q \in Q$ and $Z \in \Gamma$, if $\delta(q, \varepsilon, Z) \neq \emptyset$, then $\delta(q, a, Z) = \emptyset$, for all $a \in \Sigma$.

In this case, we simply write $\delta(q, a, Z) = (p, \gamma)$ instead of $\delta(q, a, Z) = \{(p, \gamma)\}$.

A language L is *deterministic context-free* if there exists a deterministic pushdown automaton $\mathscr{M}$ such that $T(\mathscr{M}) = L$.

3 Main Result

This section presents the main result of this paper. It completes the investigation of [11] for the case of deterministic context-free languages where the containment of the specification language K in the plant language L is known.

Theorem 1. *Controllability is undecidable for two deterministic context-free languages K and L, where L is prefix-closed, even though $K \subseteq L$.*

PROOF. We prove the theorem by reduction of Post's Correspondence Problem (PCP) to the problem of controllability of two deterministic context-free languages $K \subseteq L$. First, recall that PCP is the problem whether, given two finite sets $A = \{w_1, w_2, \ldots, w_n\}$ and $B = \{u_1, u_2, \ldots, u_n\}$ of n words over an alphabet Σ, there exists a sequence of indices $i_1 i_2 \ldots i_k$, for $k \geq 1$, such that

$$w_{i_1} w_{i_2} \ldots w_{i_k} = u_{i_1} u_{i_2} \ldots u_{i_k}.$$

It is also well-known that PCP is undecidable [6].

Let $A = \{w_1, w_2, \ldots, w_n\}$ and $B = \{u_1, u_2, \ldots, u_n\}$ be an instance of PCP over an alphabet Σ such that for all $i = 1, 2, \ldots, n$, we have $w_i \neq u_i$. We define an alphabet $\Sigma_1 = \{c_i \mid i = 1, 2, \ldots, n\}$ of new symbols, that is, $c_i \neq c_j$, for $i \neq j$, and $\Sigma \cap \Sigma_1 = \emptyset$, and two languages K and L as follows. The specification language K is defined as the set of all inverse projections of words and their mirror images (the notation w^R denotes the mirror image of a word $w \in \Sigma^*$)

$$K = \bigcup_{w \in \Sigma^*} \#P^{-1}(w)\$w^R\#$$

where $\$, \# \notin \Sigma \cup \Sigma_1$, and $P : (\Sigma \cup \Sigma_1)^* \to \Sigma^*$ is a projection. The plant language L is defined as a prefix-closure of the language

$$\begin{aligned} L' = {} & (\Sigma \cup \Sigma_1 \cup \{\$, \#\})^* \cdot \\ & \{\#c_{i_1} u_{i_1} c_{i_2} u_{i_2} \ldots c_{i_k} u_{i_k} \$ w_{i_k}^R \ldots w_{i_2}^R w_{i_1}^R \#@ \mid \\ & \qquad k \geq 1, u_i \in B, \text{ and } w_i \in A\} \end{aligned}$$

defined as a concatenation of the regular language $(\Sigma \cup \Sigma_1 \cup \{\$, \#\})^*$ with the deterministic context-free language $\{\#c_{i_1} u_{i_1} c_{i_2} u_{i_2} \ldots c_{i_k} u_{i_k} \$ w_{i_k}^R \ldots w_{i_2}^R w_{i_1}^R \#@ \mid k \geq 1,\ u_i \in B$, and $w_i \in A\}$, where @ is a new symbol.

As, obviously, $K \subseteq (\Sigma \cup \Sigma_1 \cup \{\$, \#\})^*$, K is also included in the language $L = \overline{L'}$ as required.

It is not hard to construct a deterministic pushdown automaton for K. Considering a word $\#v\$u\#$, the pushdown automaton reads v symbol by symbol from the input and pushes each symbol from the alphabet Σ to the pushdown store, until it reads symbol \$. Then, it reads a symbol from the input, compares it with the symbol stored on the top of the pushdown store, and if they match, it pops the top of the pushdown and continues; otherwise, it rejects. Thus, the word stored in the pushdown store says the automaton what should be read from the input in the next computational step. Hence, the pushdown automaton is deterministic.

Similarly, we can construct a deterministic pushdown automaton for the plant language $L = \overline{L'}$. However, this construction is a bit tricky. The main idea is that the automaton always tries to verify that the word between two symbols # is of the form $c_{i_1} u_{i_1} c_{i_2} u_{i_2} \ldots c_{i_k} u_{i_k} \$ w_{i_k}^R \ldots w_{i_2}^R w_{i_1}^R$. If it fails to check this property, it changes its mind and considers the read part of the input as a prefix from the language $(\Sigma \cup \Sigma_1 \cup \{\$, \#\})^*$. More specifically:

(1) The automaton reads the input until it reads the first symbol #.

(2) Then, it verifies that the subword between two #'s is of the form $c_{i_1} u_{i_1} c_{i_2} u_{i_2} \ldots c_{i_k} u_{i_k} \$ w_{i_k}^R \ldots w_{i_2}^R w_{i_1}^R$. This is done so that (because # has already been read from the input in step 1)

    (*) the automaton must now read $c_i \in \Sigma_1$, for some i, from the input. If so, it stores c_i to the pushdown store, and verifies that u_i follows on the input. This verification can be done using only the finite state control (i.e., the pushdown store is not used). This is repeated until the symbol \$ is read. After that, it pops $c_k \in \Sigma_1$ from the top of the pushdown store and reads w_k^R from the input (again, this can be done using only the finite state control). This procedure is repeated until the next symbol # is read.

    If there is no inconsistency discovered in the procedure (*) and the pushdown store is empty (but the initial pushdown symbol), then if the automaton reads @ from the input, it halts if the whole input has been read; otherwise, it goes to a *rejecting* state (because @ is not the last symbol of the input word) and reads the rest of the input. If the automaton does not read @ after #, it empties its pushdown store so that only the initial pushdown symbol is left in the pushdown store, and continues as in the procedure (*). This corresponds to the situation where the read part of the input belongs to the prefix $(\Sigma \cup \Sigma_1 \cup \{\$, \#\})^*$.

(3) On the other hand, as soon as an inconsistency is discovered in the procedure (*), the automaton keeps reading (without checking any properties) the input symbols from the language $(\Sigma \cup \Sigma_1 \cup \{\$, \#\})^*$ until it reads the next symbol #. If the next symbol behind # on the input is @, the automaton halts and *rejects* because the suffix $\#\tilde{w}\#@$ has to be of the required form. Otherwise, as above, it empties its pushdown store (but the initial pushdown symbol) and goes to step 2. This again corresponds to the situation where the read part of the input belongs to the prefix $(\Sigma \cup \Sigma_1 \cup \{\$, \#\})^*$.

The only rejecting states of the pushdown automaton are those states the automaton goes to when the input contains the symbol @ which is not the last symbol of the input, or when the suffix in step 3 is not of the correct form. In all other cases, the automaton is in accepting states. Thus, the automaton rejects if and only if the input is not correct (it contains @ but not as the last symbol), or the suffix $\#\tilde{w}\#@ \notin \{\#c_{i_1} u_{i_1} c_{i_2} u_{i_2} \ldots c_{i_k} u_{i_k} \$ w_{i_k}^R \ldots w_{i_2}^R w_{i_1}^R \#@ \mid k \geq 1,\ u_i \in B$, and $w_i \in A\}$. Moreover, note that the pushdown automaton is deterministic.

Now, let $\Sigma_u = \{@\}$. We prove that $\overline{K}\{@\} \cap L \subseteq \overline{K}$ (that is, K is controllable with respect to L and Σ_u) if and only if the instance (A, B) of PCP has no solution.

If $\overline{K}\{@\} \cap L \subseteq \overline{K}$, then the instance of PCP has no solution. To prove this, assume, for the sake of contradiction, that the instance of PCP has a solution $i_1 i_2 \dots i_k$, for some $k \geq 1$, that is, $w_{i_1} w_{i_2} \dots w_{i_k} = u_{i_1} u_{i_2} \dots u_{i_k}$. But this means that the word $\#c_{i_1} u_{i_1} c_{i_2} u_{i_2} \dots c_{i_k} u_{i_k} \$ w_{i_k}^R \dots w_{i_2}^R w_{i_1}^R \# \in K$ because $P(c_{i_1} u_{i_1} c_{i_2} u_{i_2} \dots c_{i_k} u_{i_k}) = u_{i_1} u_{i_2} \dots u_{i_k} = w_{i_1} w_{i_2} \dots w_{i_k}$. Moreover, $\#c_{i_1} u_{i_1} c_{i_2} u_{i_2} \dots c_{i_k} u_{i_k} \$ w_{i_k}^R \dots w_{i_2}^R w_{i_1}^R \#@ \in L$, which then implies that

$$\#c_{i_1} u_{i_1} c_{i_2} u_{i_2} \dots c_{i_k} u_{i_k} \$ w_{i_k}^R \dots w_{i_2}^R w_{i_1}^R \#@ \in \overline{K}$$

by the controllability property. However, no word of $\overline{K}$ ends with (even contains) symbol @, which is a contradiction. Thus, the instance of PCP has no solution.

On the other hand, assume that the instance of PCP has no solution. We show that then $\overline{K}\{@\} \cap L \subseteq \overline{K}$. Again, for the sake of contradiction, assume that $\overline{K}\{@\} \cap L \not\subseteq \overline{K}$. Then, it means that there exists a word $w \in \overline{K}$ such that $w@ \in L$ and $w@ \notin \overline{K}$. However, any word $w@$ of L ending with @ is of the form

$$\Upsilon^* \#c_{i_1} u_{i_1} c_{i_2} u_{i_2} \dots c_{i_k} u_{i_k} \$ w_{i_k}^R \dots w_{i_2}^R w_{i_1}^R \#@$$

for $k \geq 1$, where $\Upsilon = \Sigma \cup \Sigma_1 \cup \{\$, \#\}$, which means that it ends with $\#@$. Moreover, because of this and the assumption that $w \in \overline{K}$, it begins and ends with # and, therefore, $w \in K$. This implies that $w@ = \#c_{i_1} u_{i_1} c_{i_2} u_{i_2} \dots c_{i_k} u_{i_k} \$ w_{i_k}^R \dots w_{i_2}^R w_{i_1}^R \#@ \in K\{@\} \cap L$. In addition, since we have that $w \in K$, we get that $P(c_{i_1} u_{i_1} c_{i_2} u_{i_2} \dots c_{i_k} u_{i_k}) = u_{i_1} u_{i_2} \dots u_{i_k} = w_{i_1} w_{i_2} \dots w_{i_k}$. This means that the instance (A, B) of PCP has a solution, namely $i_1 i_2 \dots i_k$, which is a contradiction. □

4 Conclusion

In this paper, we have shown that controllability is undecidable for two deterministic context-free languages even though the specification language is known to be included in the plant language. This result shows that the undecidability of controllability is not just a by-product of the undecidability of containment as one might infer from Theorem 2.1 and Corollary 2.1 of [11]. Moreover, this result also opens up the possibility that there might exist modeling paradigms where containment might be decidable, yet controllability remains undecidable.

In addition, note that the languages constructed in the proof of the main result are also linear [8], so the theorem proves that controllability is undecidable even for linear, deterministic context-free languages.

Recall that the case where one of the languages is represented by a deterministic pushdown automaton deserves more attention because it is able to treat the state-space explosion problem, which is the most interesting question of the current supervisory control theory. Unfortunately, as shown in this paper and in [11], it is not possible, in general, to represent both languages as deterministic pushdown automata because of the undecidability issues.

Finally, we briefly discuss the decidability of controllability for the case the specification language K is regular and the plant language L is (deterministic) context-free. By the closure properties of regular and context-free languages, the language $\overline{K}\Sigma_u$ is regular, and $\overline{K}\Sigma_u \cap L$ is context-free. Since the controllability property $\overline{K}\Sigma_u \cap L \subseteq \overline{K}$ is equivalent to the emptiness problem $(\overline{K}\Sigma_u \cap L) \cap (\Sigma^* \setminus \overline{K}) = \emptyset$, and the language $(\overline{K}\Sigma_u \cap L) \cap (\Sigma^* \setminus \overline{K})$ is context-free, the decidability of controllability follows from the decidability of the emptiness problem for context-free languages [8]. Note that if L is deterministic context-free, it is decidable whether $K \subseteq L$. However, if L is not deterministic, it is (in general) undecidable whether $K \subseteq L$.

Acknowledgements

The author gratefully acknowledges very useful suggestions and comments of the anonymous referees. The research was supported by the GAČR grant no. P202/11/P028, and by RVO: 67985840.

References

[1] C. G. Cassandras and S. Lafortune. *Introduction to discrete event systems*. Springer, second edition, 2008.

[2] V. Geffert, C. Mereghetti, and B. Palano. More concise representation of regular languages by automata and regular expressions. *Inform. and Comput.*, 208:385–394, 2010.

[3] C. Griffin. A note on deciding controllability in pushdown systems. *IEEE Trans. Automat. Control*, 51(2):334–337, 2006.

[4] C. Griffin. *Decidability and optimality in pushdown control systems: a new approach to discrete event control*. PhD thesis, Penn State University, 2007. [Online]. Available at http://etda.libraries.psu.edu/paper/7980/.

[5] C. Griffin. On partial observability in discrete event control with pushdown systems. In *Proc. of ACC 2010*, pages 2619–2622, Baltimore, MD, 2010.

[6] E. L. Post. A variant of a recursively unsolvable problem. *Bull. Amer. Math. Soc.*, 52, 1946.

[7] P. J. Ramadge and W. M. Wonham. The control of discrete event systems. *Proc. of IEEE*, 77(1):81–98, 1989.

[8] A. Salomaa. *Formal languages*. Academic Press, New York, 1973.

[9] G. Sénizergues. T(A)=T(B)? In *Proc. of ICALP 1999*, volume 1644 of *Lecture Notes in Comput. Sci.* Springer, Berlin, 1999.

[10] G. Sénizergues. L(A)=L(B)? decidability results from complete formal systems. *Theoret. Comput. Sci.*, 251(1-2):1–166, 2001.

[11] R.S. Sreenivas. On a weaker notion of controllability of a language K with respect to a language L. *IEEE Trans. Automat. Control*, 38(9):1446–1447, 1993.

[12] W. M. Wonham. Supervisory control of discrete-event systems. Lecture notes, University of Toronto. [Online]. Available at http://www.control.utoronto.ca/DES/, 2011.