On Succinctness of Hierarchical State Diagrams in Absence of Message Passing

Andrzej Wąsowski

Department of Innovation
IT University of Copenhagen, Denmark
Email: wasowski@itu.dk

Abstract

We show a subexponential but superpolynomial lower bound for flattening problem for statecharts. The result explains why common flattening algorithms explode, if the signal communication is excluded from the target language. This specifically affects flattening-based strategies for automatic model-based program synthesis.

Key words: statecharts, semantics, succinctness, code generation

1 Introduction

The formalism of hierarchical state diagrams underlies multiple modeling languages and tools, mostly variants of Harel's statecharts [8]. Flattening, or elimination of hierarchy, is an operation typically applied to hierarchical models both in theoretical and practical settings. It is used to give the semantics of hierarchical languages [9] and to provide algorithms for code generation [13], automatic testing [4] and model checking [6]. Flat models can be easily interpreted with very limited writable memory usage. They are also easier to analyze for worst-case execution time approximations, as they can be interpreted using a single loop. Finally they can be more easily translated to hardware circuits. Due to this multitude of applications, complexity of the flattening problem appears an important property of the language. Its impact has been studied previously, however only questions relevant to model-checking community have been addressed. Present paper belongs to the line of new developments discussing succinctness of hierarchical models from program synthesis perspective. We show a superpolynomial lower bound for flattening translations to languages without signal communication.

We proceed as follows. Section 2 introduces hierarchical and flat statecharts, defines flattening and formulates the main claim, which is then proved and discussed in section 3. Section 4 is devoted to related work and remaining open problems. We conclude in section 5.

This is a preliminary version. The final version will be published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

2 Problem definition

2.1 Source Language

Let *State* comprise two disjoint finite classes $State_{\mathsf{and}}$ and $State_{\mathsf{or}}$, of and-states and or-states, with a hierarchy ordering $\searrow \subseteq State \times State$ such that:

[and_root]	$root \in State_{\mathsf{and}}$
[and_leaves]	$\forall s \in State_{\mathsf{or}}.\ \exists s' \in State_{\mathsf{and}}.\ s \searrow s'$
[alternation]	$\forall s' \searrow s.\ (s{\in}State_{\mathsf{and}} \wedge s'{\in}State_{\mathsf{or}}) \vee (s{\in}State_{\mathsf{or}} \wedge s'{\in}State_{\mathsf{and}})$
[rooted]	$\forall s \in State.root \searrow^* s$
[acyclic]	$\forall s, s' \in State.\ \neg(s' \searrow^+ s \wedge s \searrow^+ s')$
[no_sharing]	$\forall s, s', s'' \in State.\ s' \searrow s \wedge s'' \searrow s \Rightarrow s' = s''$

If $s' \searrow s$ then s is a child of s', $s'{=}parent(s)$ and $s{\in}children(s')$. The relation defines a directed tree on states. The *root* and leaves are and-states. All children of and-states are or-states and vice-versa. Let *Event* and *Output* be finite sets of events and outputs and *Guard* be a set of synchronization conditions over states generated by the following grammar $g ::= state | g \wedge g | \neg g$. A transition is a tuple $(s, e, g, os, t) \in Trans \subseteq State_{\mathsf{and}} \times Event \times Guard \times Output^* \times State_{\mathsf{and}}$, where s is a source state, t—a target state, e—a triggering event and os—an output sequence. We write $s \xrightarrow{[e:g]/os} t$ instead of $(s, e, g, os, t) \in Trans$. All transitions are flat, so $parent(s){=}parent(t)$ or $s{=}t{=}root$. We assume that the source state is contained in the guard, so $g \Rightarrow s$.

Each or-state s has a distinguished initial child $ini(s)$, entered whenever s is entered. A sequence of exit outputs $ex(s)$ is assigned to every and-state s. Every exit of s generetes $ex(s)$. If any and-state is active, then all its children are active. If any or-state is active, then so is exactly one of its children. A statechart $\mathcal{S}$ is a tuple $\mathcal{S} = (State_{\mathsf{and}}, State_{\mathsf{or}}, \searrow, ini, ex, Event, Trans)$.

The set of active states $\sigma \subseteq State$ is called an active configuration. A transition is enabled if its triggering event e occurs and the guard is satisfied in σ. It fires by exiting the scope, producing the outputs and entering the target. Exit actions are generated in a bottom-up manner, while entry is performed top-down. The order of hierarchy traversal is implementation dependent, but fixed. We model this choice using an injection $\gamma : State \to \mathbb{N}$:

$$\frac{\sigma_1 \vDash g \quad \langle s, \sigma_1\rangle \xrightarrow[exit]{} \langle \sigma_2, o_1\rangle \quad \langle t, \sigma_2\rangle \xrightarrow[enter]{} \langle \sigma_3\rangle}{\langle s \xrightarrow{[e:g]/o} t, \sigma_1, e\rangle \xrightarrow[fire]{} \langle \sigma_3, o_1 \hat{} o\rangle}$$

$$\frac{s \in State_{\mathsf{and}} \quad \{s_1, \ldots, s_n\}{=}children(s) \quad \gamma(s_i){<}\gamma(s_{i+1}) \quad \langle s_i, \sigma_i\rangle \xrightarrow[exit]{} \langle \sigma_{i+1}, o_i\rangle}{\langle s, \sigma_1\rangle \xrightarrow[exit]{} \langle \sigma_{n+1} \setminus \{s\}, o_1 \hat{} \ldots \hat{} o_n \hat{} ex(s)\rangle}$$

$$\frac{s \in State_{\mathsf{or}} \quad s' {\in} children(s) \cap \sigma_1 \quad \langle s', \sigma_1\rangle \xrightarrow[exit]{} \langle \sigma_2, o\rangle}{\langle s, \sigma_1\rangle \xrightarrow[exit]{} \langle \sigma_2 \setminus \{s\}, o\rangle}$$

$$\frac{s \in State_{\mathsf{and}} \quad s_1, \ldots, s_n = children(s) \quad \langle ini(s_i), \sigma_i\rangle \xrightarrow[enter]{} \langle \sigma_{i+1}\rangle}{\langle s, \sigma_1\rangle \xrightarrow[enter]{} \langle \sigma_{n+1} \cup \{s, s_1, \ldots, s_n\}\rangle}$$

A statechart executes in steps, interpreting a stream of incoming events. Each step consists of firing all enabled transitions. The iteration order is implementation dependent, but fixed, which is modeled by the injection $\pi : \mathit{Trans} \to \mathbb{N}$:

$$\frac{e \in \mathit{Event} \quad \{t_1, \ldots, t_n\} = \mathit{Trans} \quad \pi(t_i) < \pi(t_{i+1}) \quad \langle t_i, \sigma_i, e\rangle \xrightarrow[\mathit{fire}]{} \langle \sigma_{i+1}, o_i\rangle}{\langle \sigma_1, e\rangle \xrightarrow[\mathit{macro}]{} \langle \sigma_{n+1}, o_1 \hat{} \ldots \hat{} o_n\rangle}$$

The initial configuration σ_0 is computed by $\langle \mathit{root}, \emptyset\rangle \xrightarrow[\mathit{enter}]{} \langle \sigma_0\rangle$.

A statechart is *conflictless* if for any two transitions enabled in the same step their source states are not related by $\searrow^*$. We only consider conflictless statecharts for now. Dynamic semantics is given by execution traces composed of input events and sequences of outputs:

$$[\![\mathcal{S}]\!]\gamma\pi = \mathit{traces}(\mathcal{S}, \gamma, \pi) \subseteq (\mathit{Event} \times \mathit{Output}^*)^*.$$

Traces are generated by feeding $\xrightarrow[\mathit{macro}]{}$ with all possible sequences of events. Each trace element comprises an event and a complete sequence of outputs—a *reaction*. Models are input enabled, so there is a trace for any sequence of input events, but some reactions are empty. Due to conflictlessness there is only one reaction for a given event in a given global state.

The model depth d is the number of states in the longest path from *root* to a basic state. An and-depth $\hat{d}$ excludes or-states from paths ($\hat{d}{=}\lceil \frac{d}{2} \rceil$). The size of the model depends on sizes of actions, guards, output sequences and states contained.

2.2 Target Language

A Mealy machine is a finite state machine with transitions labeled by triggering events, guard conditions and sequences of atomic outputs. A flat statechart (Fig. 1) is a set of concurrent Mealy machines operating in synchronous steps and communicating by guard synchronizations. In other words this is a hierarchical statechart such that its *ex* function constantly returns the empty sequence and its $\searrow$ relation forms a shallow tree ($d = 3$ and $\hat{d} = 2$). The semantics is defined in the same manner as for hierarchical statecharts.

2.3 Flattening

Neither the source nor the target language incorporate a message passing mechanism, also known as a signal communication. Many statechart variants, including UML and Harel's statecharts, provide a facility for generating local events as outputs. Generated events are normally not available in the same step, but stored for later interpretation in a queue. A single reaction step, or a macrostep, consists of multiple microsteps processing locally generated events as long as no more are available. Our claim is that this sequencing facility is a non-trivial extension, introducing a superpolynomial gain in size.

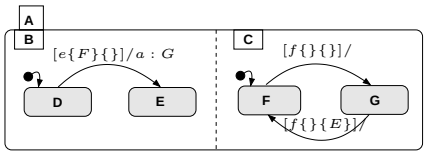

Fig. 1. An example of a flat statechart.

Definition 2.1 Statechart $\mathcal{S}'$ implements a statechart $\mathcal{S}$ iff every implementation of $\mathcal{S}'$ realizes legal executions of some implementation of $\mathcal{S}$:

$$\mathcal{S}' \lesssim \mathcal{S} \iff \forall\gamma_1\pi_1.\exists\gamma_2\pi_2.\ [\![\mathcal{S}']\!]\gamma_1\pi_1 \subseteq [\![\mathcal{S}]\!]\gamma_2\pi_2$$

Definition 2.2 Let F be an algorithm transforming statecharts. F is a flattening algorithm if for any hierarchical $\mathcal{S}$ it yields a flat $\mathcal{S}'$ such that $\mathcal{S}' \lesssim \mathcal{S}$.

Note that our definition is different from generation of a single product machine. Our understanding of hierarchy, concurrency and flattening is rather similar to that of [1,3,12,6] and substantially different than that of [10,11]. Also note that trace inclusion is a rather strong conformance requirement for input-enabled deterministic systems.

Theorem 2.3 *There exists a hierarchical statechart $\mathcal{S}$ such that for any flat statechart $\mathcal{S}'$ implementing it, $\mathcal{S}' \lesssim \mathcal{S}$:*

(i) *The size of $\mathcal{S}'$ is in $\Omega(2^{\sqrt{s}})$, where s represents the size of $\mathcal{S}$.*

(ii) *The previous claim holds if $\mathcal{S}$ is restricted to binary inputs and outputs.*

(iii) *The lower bound with growth rate arbitrarily close to the exponential, can be constructed by choosing $\mathcal{S}$ with sufficient amount of concurrency.*

3 Proof

The proof constructs a family of models such that each of its members has a superpolynomial reachable state space and each reachable configuration yields a unique sequence of exits. Such sequences cannot be represented in any flat model without equivalent superpolynomial expansion of transitions.

3.1 Family of (α, β)-models

Consider a family of statecharts with fixed number of children: α for nonbasic and-states and β for or-states ($\alpha, \beta \geq 2$). Each and-state has a unique exit output and a transition sourced in it, which is triggered by a unique event. The targets are selected in such a way that there is a transition cycle in every state machine at any level, so that every legal configuration is reachable.

A specific family member is indicated by its parameters and size, i.e. an (α, β)-model of and-depth $\hat{d}$ or an (α, β)-model of n states (where n has to be consistent with α and β). Fig. 2 presents a (2,3)-model of and-depth 3. The size of actions and guards is constant for (α, β)-models and the number of

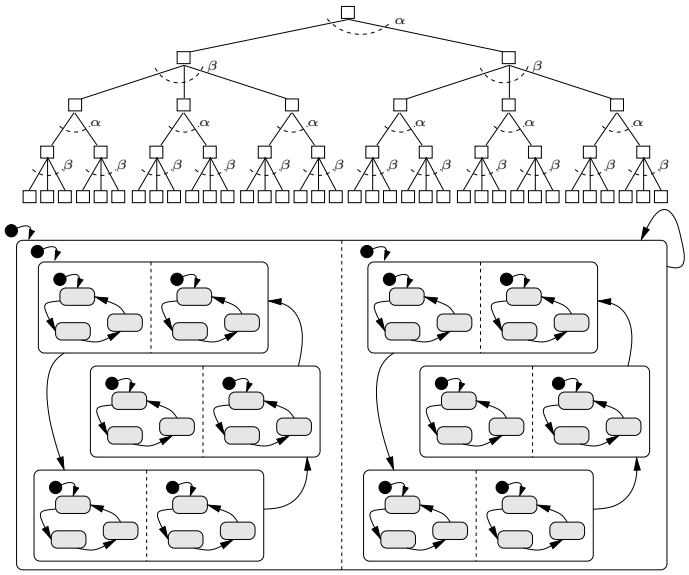

Fig. 2. A (2,3)-model of and-depth 3, also a (2,3)-model of 57 states.

transitions equals the number of states. Thus we can use the number of states n as a measure over (α, β)-models instead of a more general notion of size. The size of reachable state space for (α, β)-models can be calculated based on the fact that it equals all legal configurations:

$$R_{\hat{d}}^{(\alpha,\beta)} = \beta^{\frac{\alpha^{\hat{d}}-\alpha}{\alpha-1}} \quad .$$

This shows that the size of the reachable state space is double-exponential in the depth of the (α, β)-model and further for any legal choice of α, β and n:

$$R_n^{(\alpha,\beta)} = \beta^{\frac{\left[\frac{\beta}{\beta+1}(n-1)(\alpha\beta-1)+\alpha\beta\right]^{\log_{\alpha\beta}\alpha}-\alpha}{\alpha-1}} \quad .$$

The $R_n^{(\alpha,\beta)}$ function is $\Omega(2^{n^{\log_{\alpha\beta}\alpha}})$. Moreover if $\alpha = \beta$ then $R_n^{(\alpha,\alpha)} \in \Omega(2^{\sqrt{n}})$. The reachable state space size is superpolynomial and subexponential in n.

3.2 Succinctness

Consider the top level loop transition in any of (α, β)-models. This transition may be enabled in any reachable configuration. Each configuration yields a unique exit sequence as each state has a unique exit output assigned. Thus the top transition represents $R_{\hat{d}}^{(\alpha,\beta)}$ exit sequences. When flattening this transition, $R_{\hat{d}}^{(\alpha,\beta)}$ output reactions need to be expressed. As many other sequences are illegal, the only way to guarantee a sequence of actions to be generated in the flat models in a fixed order is to place outputs on a single transition. Otherwise, if the parts were split across various transitions, no guarantee can be given in which order outputs will be generated. Thus at least as many transitions as reachable state configurations are needed. Take $\alpha = \beta$ and the first claim of theorem 2.3 is achieved.

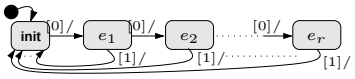

Fig. 3. An additional component decoding the binary input.

In order to prove the second claim it suffices to show a polynomial translation of statecharts over arbitrary alphabet to statecharts over binary alphabet.

Input encoding. Assume $Event = \{e_1, \ldots, e_r\}$. We will use $i + 1$ bit events to encode event e_i. First i *zero* symbols are sent, followed by a single *one* symbol. The translated model continuesly receives *zero* symbols, advancing the counter state. When *one* arrives, relevant transitions are fired (Fig. 3). A fresh concurrent component, is added to the translated model. The triggering event on every transition in the original model is changed to 1. If the original transition was fired by event e_i then an extra term is conjuncted to its guard, enforcing that e_i is active. A transition $s_1 \xrightarrow{[e_i:g]/os} s_2$ becomes $s_1 \xrightarrow{[1:g \wedge e_i]/os} s_2$.

The resulting model operates over binary input symbols, still presenting the same behavior and properties (modulo encoding). The size of the new model is linear in the number of events and transitions in the original model.

Output encoding. A similar encoding can be proposed for outputs. A unique binary sequence is generated instead of the original output and the model grows linearly in the number of outputs.

The entire proof can now be rephrased as the essential properties of models are not changed. The same superpolynomial order of growth is obtained, proving the second claim of theorem 2.3.

Finally note, that the innermost exponent in $\Omega(2^{n^{\log_{\alpha\beta}\alpha}})$ is a constant from the $(0; 1)$ interval. It approaches 1 as the amount of concurrency (α) grows (with fixed amount of sequentiality β). A lower bound with the growth rate arbitrary close to the exponential is constructed by selecting sufficiently big α, achieving the final claim of theorem 2.3. The influence of hierarchy on succinctness is strengthened by the presence of concurrency.

3.3 Robustness

Statecharts enjoy an abundance of variants [2]. Since our source language is a subset of typical dialects, we can state several direct conclusions on the lower bounds. Theorem 2.3 holds for source language of complete statecharts with do reactions, entry actions, join and fork transitions, cross-level transitions, signal communication, etc, given the same target language. Fortunately since our conflictless statecharts are just a special case of statecharts with conflict resolution (for example UML statecharts, or Harel's statecharts), the theorem holds also for such extended dialects. Similarly the introduction of nondeterminism in the order of processing of transitions, or in the conflict resolution, does not break the proof. One still has to use a single transition for each exit scenario to implement the top-level loop in a correct way.

The proof naturally suggests a flattening algorithm of the same asymptotic complexity. Consequently the bound is tight for (α, β)-models. Any (α, β)-model of n states can be flattened to a statechart with size in $\Theta(2^{n^{\log_{\alpha\beta}\alpha}})$.

In many practical settings, the target language allows signal communication, but for various reasons flattening algorithm cannot use it. For instance it is known that excessive use of signal communication increases the hardness of symbolic model checking, so signal-based flattening cannot be efficiently applied in model-checking tools. The lower bound of theorem 2.3 is then practically useful in explaining what succinctness can be expected in such applications. In fact the theorem discloses the reason for explosion in [6,5]. All algorithms mentioned there target languages with signal communication, but do not use signals in the flattening process.

Last but not least, the flattening problem becomes easier when the queue-based signal communication is permitted in the target language [13]. The essential difference is that a signal queue, being an ordered structure, can be used to enforce the order of output generation. The argument of our proof that each sequence of exits needs to be translated to a fresh transition does not hold then. Exits can be translated to locally triggered transitions and transitions can use sequences of signals to achieve relevant sequences of outputs. Let us restate the main theorem of [13]:

Theorem 3.1 *For any hierarchical UML statechart $\mathcal{S}$ there exists a flat statechart $\mathcal{S}'$ with queue-based signal communication such that $\mathcal{S}' \lesssim \mathcal{S}$ and the size of $\mathcal{S}'$ is at most polynomial in the size of $\mathcal{S}$.*

4 Related Work

David et al. [6] claim that flattening a hierarchical transition with their algorithm may lead to an exponential growth of the model in the depth of the structure, which confirms our general observation. Drusinsky and Harel [7] discuss the succinctness of cooperative concurrency without considering the influence of hierarchy on succinctness. Our result explores a new dimension of their statecharts succinctness space. Alur et al. [1] analyze the impact of hierarchy on model checking and succinctness, omitting the relation between concurrent hierarchical models and flat models in our sense. Their results cannot be directly used to state the hardness of flattening. Moreover they exploit sharing of subhierarchies, which is not commonly supported by tools.

5 Conclusion

Flattening has been formally defined as a semantics preserving translation of hierarchical statechart to a set of synchronized Mealy machines without signal communication. We have presented a subexponential, but superpolynomial, lower bound for this problem, and studied the applicability of the result for

various dialects, including UML statecharts. It has been shown that hierarchy and concurrency cooperate in increasing the hardness of the problem. The proof contained a general technique for translating results for statecharts over arbitrary alphabets to statecharts over binary alphabets.

We argue against flattening based code generation not using signal communication or any similar concept enforcing the execution order. Such techniques would be tempting otherwise, since lack of signals lowers the usage of writable memory—a scarce resource in many embedded systems applications.

References

[1] Alur, R., S. Kannan and M. Yannakakis, *Communicating hierarchical state machines*, in: J. Wiedermann, P. van Emde Boas and M. Nielsen, editors, *Proceedings of ICALP*, LNCS **1644** (1999), pp. 169–78.

[2] Beeck, M., *A comparison of statecharts variants*, LNCS **863** (1994), pp. 128–48.

[3] Behrmann, G., K. G. Larsen, H. R. Andersen, H. Hulgaard and J. Lind Nielsen, *Verification of hierarchical state/event systems using reusability and compositionality*, in: *Proceedings of TACAS*, LNCS **1579** (1999), pp. 163–77.

[4] Binder, R. V., *Testing Object-Oriented Systems. Models, Patterns and Tools,* Addison-Wesley, 2000.

[5] Bogdanov, K. and M. Holcombe, *Properties of concurrently taken transitions of Harel statecharts*, in: *Proceedings of SFEDL*, Grenoble, France, 2002.

[6] David, A., M. O. Möller and W. Yi, *Formal verification of UML statecharts with real-time extensions*, in: R.-D. Kutsche and H. Weber, editors, *Proceedings of FASE*, LNCS **2306** (2002), pp. 218–32.

[7] Drusinsky, D. and D. Harel, *On the power of bounded concurrency I: Finite automata*, Journal of ACM **41** (1994), pp. 517–39.

[8] Harel, D., *Statecharts: A visual formalism for complex systems*, Science of Computer Programming **8** (1987), pp. 231–74.

[9] Harel, D. and A. Naamad, *The STATEMATE semantics of statecharts*, ACM Transactions on Software Engineering and Methodology **5** (1996), pp. 293–333.

[10] Roubtsova, E. E., J. van Katwijk, R. C. M. de Rooij and H. Toetenel, *Transformation of UML specification to XTG*, in: D. Bjørner, M. Broy and A. V. Zamulin, editors, *Proceedings of PSI*, LNCS **2244** (2001), pp. 249–56.

[11] Simons, A. J. H., *The compositional properties of UML statechart diagrams*, in: C. J. van Rijsbergen, editor, *3rd Electronic Workshop on ROOM* (2000).

[12] Staunstrup, J., H. R. Andersen, H. Hulgaard, J. Lind-Nielsen, K. G. Larsen, G. Behrmann, K. J. Kristoffersen, A. Skou, H. Leerberg and N. B. Theilgaard, *Practical verification of embedded software*, IEEE Computer **5** (2000), pp.68–75.

[13] Wąsowski, A., *Flattening Statecharts without Explosions* (2004), submitted.