Succinctness of Hierarchical State Diagrams in Absence of Message Passing

Andrzej Wąsowski

IT University Technical Report Series

TR-2004-42

ISSN 1600–6100 **February 2004**

ISSN 1600–6100

ISBN 87-7949-063-8

Copies may be obtained by contacting:

IT University of Copenhagen
Glentevej 67
DK-2400 Copenhagen NV
Denmark

Telephone: +45 38 16 88 88
Telefax: +45 38 16 88 99
Web www.it-c.dk

On Succinctness of Hierarchical State Diagrams in Absence of Message Passing

Andrzej Wąsowski (wasowski@itu.dk)

Abstract

We show a subexponential but superpolynomial lower bound for flattening problem for statecharts. The result presented is resistant to many variations in statecharts semantics. The immediate consequence is that usage of common flattening algorithms in the implementations of tools should be carefully examined, taking into account presence of signal communication in the target language. This specifically affects flattening-based strategies for automatic model-based program synthesis.

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 [15], automatic testing [3] 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. This makes flattening specifically attractive for code generation targeting constraint embedded systems. Due to this multitude of applications, complexity of the flattening problem appears an important property of the language.

The impact of introducing hierarchy in statecharts has been studied previously [1, 2], 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. In a parallel paper [15] a polynomial algorithm is shown for the same problem under the relaxed condition that message passing communication is allowed in the target language.

The paper proceeds as follows. Section 2 introduces hierarchical and flat statecharts, defines the flattening problem and formulates the main claim of the paper. Section 3 elaborates on development of the proof, while section 4 attempts to discuss applicability of the result to typical variants of statecharts. Section 5, mainly devoted to related work, also aims to indicate a range of open problems in discussion of succinctness of statecharts. We conclude in section 6.

2 Problem definition

2.1 Source Language

Consider a subset of the statechart language [8]. Let $State$ be a finite set divided into two disjoint classes $State_{\mathsf{and}}$ and $State_{\mathsf{or}}$. Members of the classes are called and-states and or-states correspondingly. States are ordered by a hierarchy relation $\searrow \subseteq State \times State$ 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]	$\forall s, s' \in State.\ s' \searrow s \Rightarrow$ $s \in State_{\mathsf{and}} \wedge s' \in State_{\mathsf{or}} \vee$ $\vee\ s \in State_{\mathsf{or}} \wedge s' \in State_{\mathsf{and}}$
[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 we say that s is a child of s', $s' = parent(s)$, and $s \in children(s')$. The relation imposes a directed tree structure on states: the *root*

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

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

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

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

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

Figure 1: Operational semantics rules for statecharts

node and all leaves are and-states, while all children of and-states are or-states and vice versa. Let *Event* and *Output* denote finite sets of events and outputs and *Guard* be the set of possible synchronization conditions over activity of states generated by the grammar $g ::= s \mid g \wedge g \mid \neg g$. 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 is a target state, e is a triggering event and os is the sequence of outputs. We will write $s \xrightarrow{[e:g]/os} t$ instead of $(s, e, g, os, t) \in Trans$. Only flat transitions are allowed, thus $parent(s) = parent(t)$ or $s = t = root$. We assume that for all transitions the source state is also contained in the guard condition, so $g \implies s$.

Each or-state s has a distinguished child called an initial state, denoted $ini(s)$, which is entered whenever s is entered, if further targets are not specified. Each and-state s has an assigned sequence $ex(s)$ of exit outputs, which are generated whenever s is exited.

$$ini : State_{\mathsf{or}} \to State_{\mathsf{and}} \qquad ex : State_{\mathsf{and}} \to Output^*$$

A statechart is a tuple:

$$\mathcal{S} = (State_{\mathsf{and}}, State_{\mathsf{or}}, \searrow, ini, ex, Event, Trans).$$

The *root* state is permanently active during execution. If any and-state is active, then all its children are active. If any or-state is active, then exactly one child is active. The set of active states $\sigma \in State$ is called an active configuration. A transition is enabled if its triggering event occurs and σ satisfies the guard:

$$enabled(s \xrightarrow{[e:g]/o} t, \sigma, e') \text{ iff } e = e' \text{ and } \sigma \vDash g$$

Transition fires by exiting its scope, producing outputs and entering the target. See figure 1 for a formalization of the dynamic semantics.

Exit actions are generated in a bottom up manner and entry is performed in a top down manner. The order of traversing hierarchy is fixed, so the behavior is deterministic, but the precise choice is tool implementation dependent. We parameterize the semantics with an injection $\gamma : State \to \mathbb{N}$ to model this choice.

A statechart executes in steps, interpreting a sequence of incoming events. Each such step consists of an iteration of firing enabled transitions. Again the order of the iteration is fixed but implementation dependent, which is reflected in the semantics by the priority injection function $\pi : Trans \to \mathbb{N}$.

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

A statechart is *conflictless* if for any two transitions enabled in the same step their source states are not related by $\searrow^*$, the transitive closure of $\searrow$. In other words every two transitions, which may be enabled at the same time have non overlapping scopes. We shall only consider conflictless statecharts (until section 4).

The dynamic semantics of given implementation is described by execution traces composed of input events and sequences of outputs:

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

Traces are generated by feeding the *macro* relation with all possible sequences of input events. In each

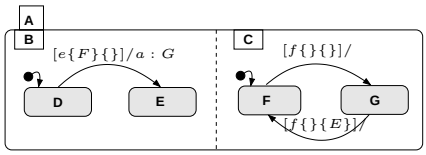

Figure 2: An example of flat statechart.

element of the trace, an environment event is accompanied by a sequence of outputs, called a *reaction*. Models are input enabled, so there is a trace for any sequence of environment events, but some reactions may be empty. There is only one reaction for a given event in a given state. This follows from conflictlessness requirement.

The *out-degree* of a state is the number of its children (i.e. the out-degree of the node in the hierarchy tree). The *depth* of the model, denoted d, is the number of states in the longest path in the hierarchy tree leading from *root* to a leaf. All models always have odd depth (as the *root* is an and-state and so are all leaves). A variant of depth—called and-*depth*, denoted $\hat{d}$, only reflects number of and-states in the paths: $\hat{d} = \lceil \frac{d}{2} \rceil$ We will denote number of all states in the model by $n = |State|$. Finally the size of the model is defined as the size of all its guards, actions output sequences and the number of states.

2.2 Target Language

A Mealy machine is a finite state machine with transitions between states and sequences of atomic actions executed when a transition is fired. Each transition is labeled by a triggering event and a guard condition.

A flat statechart (see figure 2) is defined as a set of Mealy machines, operating concurrently in synchronous steps. The machines communicate by synchronization on active states. In other words this is a statechart with one level of or-states and without exit actions. The *ex* function is a constant, always returning an empty sequence of outputs and the $\searrow$ relation forms a shallow tree: all and-states are basic states (except for the *root*). Depth of flat statecharts is $d = 3$ and $\hat{d} = 2$ in this model. The semantics of flat machines is defined in the same manner as semantics of hierarchical machines. Flat statecharts are also required to be conflictless and input-enabled.

Neither the source nor the target language incorporate a message passing mechanism, also called signal communication. Many statechart variants, including UML and Harel's statecharts, provide a facility for generating local events as outputs in states and on transitions. Generated events are normally not available in the same step, but stored for interpretation in a set, multiset or a queue. A single reaction step, or a macrostep, consists of multiple microsteps processing locally generated events as long as no more local events are available. My main claim is that this extension is non-trivial, thus in the absence of message passing communication hierarchy leads to superpolynomial succinctness gains. For this reason message passing has been excluded from the language definition.

2.3 Flattening

Implementation relation.

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

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

Note that trace inclusion is a sufficient implementation criterion for input-enabled and deterministic systems.

Flattening.

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

Literature mentions a multitude of meanings for flattening and related concepts. Let me stress that the meaning given above is different from generation of single product machine for all concurrent components. Our understanding of hierarchy, concurrency and flattening is rather similar to that of [1, 2, 13, 6] and substantially different than that of [5, 10, 12].

Theorem 1. *There exists a hierarchical statechart $\mathcal{S}$ such that for any flat statechart $\mathcal{S}'$ implementing it, $\mathcal{S}' \lesssim \mathcal{S}$:*

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

2. *Previous claim holds even if $\mathcal{S}$ is restricted to binary inputs and outputs.*

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

Note that the second claim of the above theorem is stronger than the initial one. It says that the lower bound holds even for a subset of hierarchical statecharts over binary alphabet (decreasing the set from which $\mathcal{S}$ can be chosen). So the first claim is a special case of the second claim. The third claim is even stronger saying that the lower bound can be increased arbitrary close to the exponential function. We will show how to construct $\mathcal{S}$ so that the degree of the root in the exponent approaches one, as the amount of concurrency in $\mathcal{S}$ increases.

3 Proof

The proof proceeds by identifying an infinite family of (α, β)-models such that each of the members in the family has a superpolynomial reachable state space and each state configuration yields a different sequence of exit outputs. Then the observation is made that such sequences cannot be represented in any flat model without equivalent multiplication of transitions. Finally I show that the family of (α, β)-models contains statecharts for which hardness of flattening problem is arbitrary close to an exponential of model size.

3.1 Family of (α, β)-models

Consider a family of statecharts with fixed out-degree α for nonbasic and-states and fixed out-degree β for or-states ($\alpha \geq 2, \beta \geq 2$). Each and-state has a unique exit action assigned. For each and-state in the model there is a transition sourced in that state. Each transition has a unique event triggering it. The targets of transitions are selected in such a way that there is a cycle over and-states in any particular state machine at any level. Note that this way every statically legal configuration in the statechart is reachable.

We will indicate a specific model in the family by giving its parameters and size, calling it an (α, β)-model of and-depth $\hat{d}$ or an (α, β)-model of n states. In the latter case n has to be consistent with α and β. Figure 3 presents a (2,3)-model of and-depth 3.

Note that the size of any model depends on size of actions and guards, the number of transitions and number of states. Size of actions and guards is constant for (α, β)-models and the number of transitions is the same as number of states. Thus, from now on, we will use the number of state n as a measure over (α, β)-models instead of general size s.

3.2 Reachable State Space

We shall now be concerned with the size of the reachable state space of an (α, β)-model. Let $width_{\hat{k}}$ denote the number of states on and-depth $\hat{k}$ (i.e. on $\hat{k}$th level of and-states) in an (α, β)-model:

$$width_{\hat{k}}^{(\alpha,\beta)} = (\alpha\beta)^{\hat{k}-1}$$

In particular $width_{\hat{d}}$ denotes the number of basic states in a given family member. The number of active states of an (α, β)-model at and-depth $\hat{k}$ is given by:

$$active_{\hat{k}}^{(\alpha,\beta)} = \alpha^{\hat{k}-1}$$

The total number of states as a function of and-depth can be described with the following recurrence:

$$\begin{aligned} n_1^{(\alpha,\beta)} &= 1 \\ n_{\hat{d}}^{(\alpha,\beta)} &= n_{\hat{d}-1}^{(\alpha,\beta)} + width_{\hat{d}-1}^{(\alpha,\beta)} \cdot (\alpha + \alpha\beta). \end{aligned} \quad (1)$$

The recurrence, solved and inverted, gives the and-depth of the model as a function of the number of states n:

$$\hat{d}^{(\alpha,\beta)} = \log_{\alpha\beta}\left[\frac{\beta}{\beta+1}(n-1)(\alpha\beta-1) + \alpha\beta\right] \quad (2)$$

for legal combinations of values of α, β and n. Formula (2) gives a translation from functions over and-depth to functions over model size.

Recall that all statically legal configurations are reachable in (α, β)-models, so the number of reachable states in a model of depth $\hat{k}$ given by $R_{\hat{k}}^{(\alpha,\beta)}$, is equal to the number of possibilities in which active sets of states can be selected according to semantics of statecharts.

$$\begin{aligned} R_1^{(\alpha,\beta)} &= 1 \\ R_{\hat{k}}^{(\alpha,\beta)} &= \sum_{i=1}^{R_{\hat{k}-1}^{(\alpha,\beta)}} (\beta^{\alpha})^{active(\hat{k}-1)} \end{aligned}$$

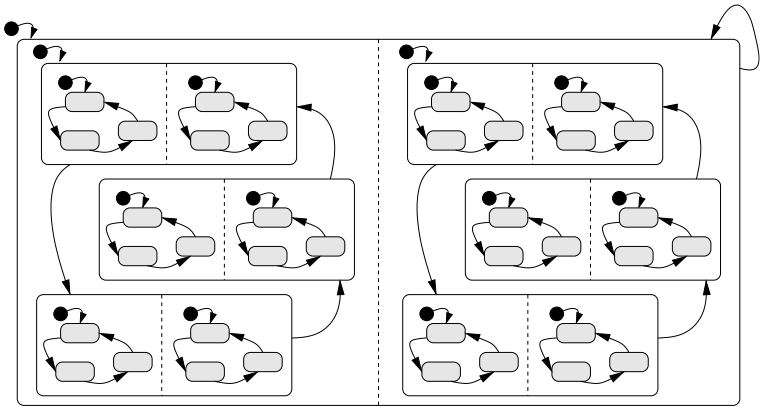

Figure 3: (2,3)-model of and-depth 3, also a (2,3)-model of 57 states.

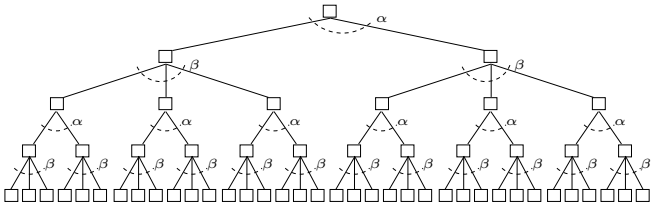

Figure 4: Hierarchy tree of (2,3)-model of figure 3

The term under summation is independent of sum index, because of the high regularity of the model. Each configuration at level $\hat{k}-1$ can be refined to exactly the same number of configurations on level $\hat{k}$. Thus the recurrence simplifies to:

$$R_1^{(\alpha,\beta)} = 1$$
$$R_{\hat{k}}^{(\alpha,\beta)} = R_{\hat{k}-1}^{(\alpha,\beta)} \beta^{\,\alpha active(\hat{k}-1)},$$

which can be solved giving:

$$R_{\hat{d}}^{(\alpha,\beta)} = \beta^{\frac{\alpha^{\hat{d}}-\alpha}{\alpha-1}} \tag{3}$$

This shows that the size of the reachable state space is double exponential in the depth of the (α,β)-model. Substitute (2) to obtain the size of reachable state space as a function of model size:

$$R_n^{(\alpha,\beta)} = \beta^{\frac{\left[\frac{\beta}{\beta+1}(n-1)(\alpha\beta-1)+\alpha\beta\right]^{\log_{\alpha\beta}\alpha}-\alpha}{\alpha-1}}, \tag{4}$$

for any legal choice of α, β and n. 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 size of reachable state space for (α,β)-models is double exponential in the model depth, but superpolynomial and subexponential in the model size.

3.3 Succinctness

Let us deal with the unlimited case first, when arbitrary many input and output symbols are allowed in our statecharts. We shall see that (α,β)-models cannot be translated to flat statecharts without superpolynomial growth of size. 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 the same transition. Otherwise, if the parts were split across various transitions, no guarantee can be given in which order outputs will be generated (the order of processing transitions is fixed, but unknown and implementation dependent). Thus at least as many transitions as reachable state configurations are needed. Take $\alpha = \beta$ and the first claim of theorem 1 is reached.

3.4 Input output alphabet

In order to prove the second claim of theorem 1, namely that the same lower bound holds for binary input output alphabets, it suffices to show a polynomial translation of statecharts over arbitrary alphabet to statechart over binary alphabet. The translation should preserve the semantics of original statechart allowing triggering non binary events by encoding in binary sequences and similar observation of non-binary effects with distinguishable binary words.

Input encoding

We assume a finite number of events in input alphabet, indexed from 1 to r. We will use $i+1$ bits to encode the firing of event e_i. First i *zero* symbols are sent, followed by a single *one* symbol. The translated model continues to receive *zero* symbols, advancing the counter of arriving event, and fires relevant transitions, when *one* arrives. A fresh component, illustrated on figure 5, is added to the model being translated by means of concurrent composition.

Then the triggering event on every transition in the old model is changed to 1. If the original transition was fired by event e_i then an extra term is conjuncted to transition's guard enforcing that state e_i is active. A transition (s_1, e_i, g, os, s_2) becomes $(s_1, 1, g \wedge e_i, os, s_2)$. The size of each transition has been increased by a constant factor.

The resulting model operates over binary input symbols, still presenting the same behavior and properties (modulo encoding). Moreover the size of the new model is linear in the number of transitions in original model, as in the worst case as many new states and new transitions have been added as there were transitions in the original model (if each transition was fired by unique event).

Output encoding

The translation from models over arbitrary output alphabet to models over binary output alphabet is even easier. It suffices to use any isomorphic encoding of natural numbers in binary alphabet and instead of every output generate a corresponding sequence of binary outputs.

The above encodings are generally useful whenever complexity proofs for statecharts need to be generalized to models over binary alphabets. In our case we notice that (α, β)-models can be translated within polynomial bounds to a corresponding family over binary alphabets. The whole proof can be rephrased in this framework – the properties of models are not changed. All configurations are reachable and each configuration gives rise to a unique set of exit sequences. One still obtains the same superpolynomial order of growth, which finishes the proof of second claim in theorem 1.

3.5 Improving the lower bound

Let us return to the lower bound on number of configurations $\Omega(2^{n^{\log_{\alpha\beta}\alpha}})$. Note that the innermost exponent in the lower bound function is a constant from the interval $(0;1)$. Moreover if one extends the amount of concurrency in the model (controlled by α, keeping the amount of sequentiality (controlled by β) constant, the exponent approaches 1. Thus one can give a lower bound of the size being arbitrarily close to exponential in the sense of growth rate.

This shows that the third claim of theorem 1 is true. It is harder to flatten more concurrent models, despite the fact that concurrency is preserved by flattening. Hierarchy is strengthened by concurrency. □

4 Robustness

Statecharts enjoy an abundance of variants [14]. For this reason it is always important to evaluate applicability of statecharts related results at least with respect to major dialects. Since our source language is a subset of typical statechart dialects we can directly state several conclusions.

Corollary 1. *Theorem 1 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.*

In section 2.1 we have described a deterministic semantics for conflictless statecharts. Most of the statechart variants, however, allow some conflicts, carefully specifying deterministic rules for resolution. Fortunately our conflictless statecharts are still just a special subset of the language, so:

Corollary 2. *Theorem 1 holds also for statecharts with conflict resolution, given the same target language.*

Some statecharts variants underspecify the order of traversing children and processing transitions. This can be achieved in our model by disregarding

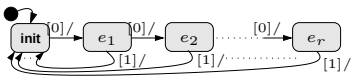

Figure 5: An extra component decoding the binary input.

semantic conditions on priority orders γ and π. In such case multiple reactions are possible for a single input event, and the set of traces is richer. However, because relaxing γ introduces a non empty partial order on exiting (instead of a total order) one still has to implement at least $R_{\hat{d}}^{(\alpha,\beta)}$ sequences to implement the top level transition. Otherwise the correct semantics of exits cannot be guaranteed.

Corollary 3. *Theorem 1 holds also for deterministic statecharts with nondeterministic semantics of processing transitions and exits, given the same target language.*

Going further, some dialects, permit non-resolvable conflicts, namely nondeterministic selection of conflicting transitions. However such nondeterministic statecharts are still including our (α, β)-models:

Corollary 4. *Theorem 1 holds also for flattening fully nondeterministic statecharts, given the same target language.*

Author is not aware of any major statechart variant, for which the result presented would not hold.

The proof naturally suggests an algorithm of the same asymptotic complexity for flattening of statecharts. Thus for our (α, β)-models, which are a kind of regular statecharts (they have regular tree structure) the bound given is tight:

Observation 1. *Any (α, β)-model of n states can be flattened to a statechart with size in $\Theta(2^{n^{\log_{\alpha\beta}\alpha}})$.*

This also means that for regular statecharts in this sense an exponential limit on the lower bound cannot possibly be reached.

In many practical settings, the target language allows signal communication, but for various reasons flattening algorithm cannot take any significant advantage of it. For instance it is known that excessive use of signal communication increases the hardness of model checking, so signal-based flattening cannot be efficiently applied in model-checking tools. The lower bound of theorem 1 is then practically useful in explaining what succinctness can be expected in such applications. In fact it explains formally the reason for size explosion in [6, 4], even if their target languages are richer, precisely because these algorithms do not use signals in the flattening process.

Until now we have considered variations in the input language. It should be emphasized that the problem becomes easier when the queue-based signal communication is permitted in the target language [15]. Our (α, β)-models can now be flattened in polynomial space. The essential difference in this new formulation of the problem is that a signal queue, being an ordered structure, can be used to enforce the order of executing actions, so the argument of our proof that each sequence of exits needs to be translated to a fresh transition does not hold any more. Exit actions can be translated to locally triggered transitions and transitions can use sequences of signals to achieve relevant sequences of outputs. The resulting statechart is linear in the size of (α, β)-model. See [15] for details of the complete algorithm. Let as restate the main theorem of [15]:

Theorem 2. *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 $\mathcal{S}$.*

This result combined with the claim of theorem 1 indicates the main difference between signal-based approach and flattening not using any message-like communication.

5 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. Note that it exactly agrees with formula 3 presented above. Thus their algorithm can be used as another argument explaining observation 1. The question of establishing strict bounds for arbitrary models in the size of the model still remains open.

Drusinsky and Harel[7] discuss the succinctness introduced by cooperative concurrency, however they

do not consider the influence of hierarchy on succinctness. The present result explores a different dimension of their succinctness space for statecharts.

Alur et al.[1] thoroughly discuss the impact of hierarchy on model checking problems and the size of models. However they omit the relation between concurrent hierarchical models and flat models in our sense, thus their results cannot be directly used to state the hardness of flattening. More over Alur et al. exploit the sharing of subhierarchies in their semantics, which is not commonly used in engineering modeling languages (see UML statecharts).

As it has been mentioned before Wąsowski [15] presents a polynomial flattening algorithm taking an advantage of message passing, at the same time showing that signal communication is a nontrivial extension to the language. An implementation of this algorithm is presently used in SCOPE code generator [11].

6 Conclusion & Future Work

Flattening has been formally defined as a semantics preserving translation of hierarchical statechart to a set of synchronized Mealy machines in absence of signal communication. I have presented a subexponential, but superpolynomial, lower bound for this problem, and studied the applicability of the result for various dialects, including the standardized UML statecharts. It has been formally shown that hierarchy and concurrency interplay in increasing the hardness of the problem. In addition our proof contained a generic technique of translating results for statecharts over arbitrary alphabets to statecharts over binary alphabets.

A number of actual flattening algorithms have been indicated, which face the size explosion issue, which has now been shown to be inherent for the problem (under the conditions they have to meet), not only for the algorithms themselves.

Our lower bound result presents an argument against code generation techniques for statecharts, which are based on flattening in absence of message passing, or any other concept enforcing the order of execution internally in the model. Such techniques would be tempting otherwise, since lack of signal communication significantly lowers the usage of writable memory, which is a crucial requirement in many engineering applications, especially in the embedded systems domain.

We conjecture that an algorithm similar to signal-based flattening of [15] cannot be used for original Harel's statecharts, which use sets for storing pending signals instead of queues (as UML does). Thus the result of theorem 1 is likely to be strengthened, by allowing set-based signal communication in the target language. This remains the main open question in the future work.

7 Acknowledgements

Author would like to thank Peter Sestoft and Kim G. Larsen for reading and commenting on an earlier version of this text.

References

[1] Rajeev Alur, Sampath Kannan, and Mihalis Yannakakis. Communicating hierarchical state machines. In Jirí Wiedermann, Peter van Emde Boas, and Mogens Nielsen, editors, *26th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 1644 of *Lecture Notes in Computer Science*, pages 169–178, Prague, Czech Republic, July 1999. Springer-Verlag.

[2] G. Behrmann, K. G. Larsen, H. R. Andersen, H. Hulgaard, and J. Lind Nielsen. Verification of hierarchical state/event systems using reusability and compositionality. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 1579 of *Lecture Notes in Computer Science*, pages 163–177, Amsterdam, The Netherlands, March 1999. Springer-Verlag.

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

[4] Kirill Bogdanov and Mike Holcombe. Properties of concurrently taken transitions of Harel statecharts. In *Workshop on Semantic Foundations of Engineering Design Languages (SFEDL)*, Grenoble, France, April 2002.

[5] Gregory W. Bond, Franjo Ivancic, Nils Klarlund, and Richard Trefler. Eclipse feature logic analysis. In *2nd IP-Telephony Workshop*, pages 100–107, New York City, USA, April 2001.

[6] Alexandre David, M. Oliver Möller, and Wang Yi. Formal verification of UML statecharts with real-time extensions. In Ralf-Detlef Kutsche and Herbert Weber, editors, *Fundamental Approaches to Software Engineering (FASE)*, volume 2306 of *Lecture Notes in Computer Science*, pages 218–232, Grenoble, France, April 2002. Springer-Verlag.

[7] Doron Drusinsky and David Harel. On the power of bounded concurrency I: Finite automata. *Journal of ACM*, 41(3):517–539, May 1994.

[8] David Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8:231–274, 1987.

[9] David Harel and Amnon Naamad. The STATEMATE semantics of statecharts. *ACM Transactions on Software Engineering and Methodology*, 5(4):293–333, 1996.

[10] Ella E. Roubtsova, Jan van Katwijk, Ruud C. M. de Rooij, and Hans Toetenel. Transformation of UML specification to XTG. In Dines Bjørner, Manfred Broy, and Alexandre V. Zamulin, editors, *Perspectives of System Informatics (PSI), 4th International Andrei Ershov Memorial Conference, Revised Papers*, volume 2244 of *Lecture Notes in Computer Science*, pages 249–256, Akademgorodok, Novosibirsk, July 2001. Springer-Verlag.

[11] SCOPE: A statechart compiler, 2003. `http://www.mini.pw.edu.pl/~wasowski/scope`.

[12] Anthony J. H. Simons. The compositional properties of UML statechart diagrams. In C. J. van Rijsbergen, editor, *Third Electronic Workshop on Rigorous Object-Oriented Methods*. British Computer Society, 2000.

[13] Jørgen Staunstrup, Henrik Reif Andersen, Henrik Hulgaard, Jørn Lind-Nielsen, Kim Guldstrand Larsen, Gerd Behrmann, Kaare J. Kristoffersen, Arne Skou, Henrik Leerberg, and Niels Bo Theilgaard. Practical verification of embedded software. *IEEE Computer*, 5(33):68–75, 2000.

[14] Michael von der Beeck. A comparison of statecharts variants. In *ProCoS: Third International Symposium on Formal Techiniques in Real Time and Fault-Tolerant Systems.*, volume 863 of *Lecture Notes in Computer Science*, pages 128–148. Springer-Verlag, 1994.

[15] Andrzej Wąsowski. Flattening Statecharts without Explosions. Submitted, 2004.