DOI 10.1007/s00165-011-0204-5
The Author(s) © 2011. This article is published with open access at Springerlink.com
Formal Aspects of Computing (2013) 25: 659–681

Formal Aspects of Computing

Statechartable Petri nets

Rik Eshuis

Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands.
E-mail: h.eshuis@tue.nl

Abstract. Petri nets and statecharts can model concurrent systems in a succinct way. While translations from statecharts to Petri nets exist, a well-defined translation from Petri nets to statecharts is lacking. Such a translation should map an input net to a corresponding statechart, having a structure and behaviour similar to that of the input net. Since statecharts can only model a restricted form of concurrency, not every Petri net has a corresponding statechart. We identify a class of Petri nets, called statechartable nets, that can be translated to corresponding statecharts. Statechartable Petri nets are structurally defined using the novel notion of an area. We also define a structural translation that maps each statechartable Petri net to a corresponding statechart. The translation is proven sound and complete for statechartable Petri nets.

Keywords: Petri nets, Statecharts, Formal translation

1. Introduction

While finite state machines are a popular technique for modelling the control flow of simple systems, it has long been recognised that for complex concurrent systems more powerful techniques are needed. Petri nets and statecharts are two visual formalisms that extend finite state machines with constructs for modelling concurrency in a succinct way. Petri nets were introduced by Petri [Pet62], and have found their way in practical applications like manufacturing, workflow modelling and performance analysis [Mur89, RR98]. Statecharts were introduced by Harel [Har87] for use in the structured analysis method STATEMATE [HN96]. They have also been adopted in several object-oriented methods and the UML notation [UML03b]. In practice, both formalisms are used side by side. For instance, UML [UML03b] uses besides statecharts activity diagrams, which have been inspired by Petri nets.

Given this widespread usage of Petri nets and statecharts, it is useful to have well-defined translations between these two formalisms. While translations from statecharts to Petri nets exist, for instance [Ham05, HMP+02, SSH01], well-defined translations for the reverse direction are lacking. To fill this gap, this paper defines a *formal, structural translation from Petri nets to statecharts*.

To introduce the translation, Fig. 1 shows a Petri net (a) and its statechart translation (b). (The syntax of Petri nets and statecharts is explained in Sect. 2.) The containment relation between statechart nodes in (b) is visualised as an AND/OR tree in (c). The translation constructs a statechart whose structure resembles the input net. Corresponding syntactic constructs in both models carry the same label. Note that composite statechart nodes, like A1 and O1, do not correspond to any Petri net construct.

Correspondence and offprint requests to: R. Eshuis, E-mail: h.eshuis@tue.nl

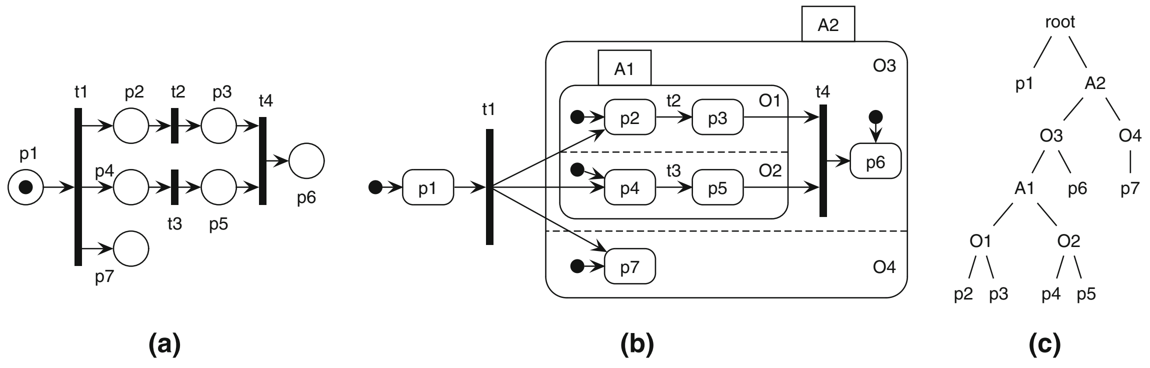


Fig. 1. Example Petri net (**a**), corresponding statechart with similar structure and behaviour (**b**) and its AND/OR tree (**c**)

The key difficulty of the translation is therefore defining the statechart composite nodes as well as the AND/OR tree, both of which have no counterpart in Petri net syntax. Moreover, composite node must be defined in such a way that the behaviour of the resulting statechart is similar to that of the input net. The translation defined in this paper constructs statecharts whose behaviour is similar to that of the input nets. Still, the translation is structural: it maps Petri net syntax to statechart syntax, without using any Petri net analysis technique like place invariants or reachability graphs.

Since statecharts can only model a restricted form of concurrency, not every Petri net can be translated to a corresponding statechart with similar structure and behaviour (see Sect. 3). We identify a class of Petri nets, called *statechartable nets*, that do have corresponding statechart translations. We structurally define statechartable nets using the novel Petri net notion of *area*. Statechartable nets are a new subclass of Petri nets. We show that statechartable nets are exactly the class of nets for which the translation returns statecharts with similar structure and behaviour.

Nevertheless, there are non-statechartable nets which do have a statechart translation with similar structure and behaviour. This implies that the class of statechartable nets is not complete. However, statecharts corresponding to non-statechartable nets are not likely to be drawn in practice, as we argue in Sect. 4.4, so this incompleteness does not seem to be a severe limitation in practice.

In this paper, we only consider safe nets, i.e., each place can contain at most one token. Unsafe nets cannot be translated to statecharts with similar structure and behaviour, as we explain in Sect. 3. Next, to simplify the exposition, we do not consider transition labels for statecharts and Petri nets in the definition of the translation. This implies we use a generic, abstract statechart semantics in which transitions are not triggered by events, but are taken when their input nodes are in the current state. The translation defined in this paper can provide the basis for more advanced translations that deal with events.

This paper is structured as follows. Section 2 provides background on Petri nets and statecharts. Section 3 relates Petri nets and statecharts, defining formally when a Petri net and a statechart correspond, i.e., when they are structurally and behaviourally similar. We also explain that not every Petri net has a corresponding statecharts. Section 4 structurally defines the class of statechartable nets, which are Petri nets that do have corresponding statechart. Section 5 structurally defines the class of statecharts that correspond to statechartable nets. Section 6 defines a declarative translation from statechartable nets to corresponding statecharts. We prove that the translation is sound and complete for statechartable nets: each statechartable net translates into a corresponding statechart with similar structure and behaviour, and the translation fails for non-statechartable nets. Section 7 presents related work. Section 8 winds up with conclusions and further work. The appendix contains formal definitions of Petri nets and statecharts and the proofs of the theorems.

2. Background

We informally present the basics of Petri nets and statecharts. Formal definitions can be found in Appendix A.

2.1. Petri nets

A Petri net (Place/Transition net) consists of places, represented by circles, transitions, represented by bars, and directed arcs connecting places to transitions and vice versa. Places and transitions are called elements. Let x, y be two elements. If there is an arc from x to y then x is input element of y and y is output element of y. For instance, in Fig. 1a place p1 is input place of transition t1 while t1 is output transition of p1. The set of all input elements of an element x is denoted $\bullet x$ and called the preset of x. The set of all output elements of an element x is denoted $x\bullet$ and called the postset of x. For example, in Fig. 1, for t1 we have $\bullet$t1 = {p1} and t1$\bullet$ = {p2, p4, p7}. We require that each transition has a non-empty preset and a non-empty postset.

Places belonging to the current state are marked with a token, visualised as a black dot. A state is also called a marking. A transition t is enabled in a state if all its input places have a token, so are in the state. In Fig. 1, transition t1 is enabled. Upon firing, from each input place a token is removed, and to each output place a token is added. This way, a new state (marking) is reached. In theory, a place can contain more than one token. However, as explained in the introduction, we consider safe nets: in each marking each place contains at most one token.

Formally, a (marked) Petri net is a tuple (P, T, F, M) where P is the set of places, T is the set of transitions such that $P \cap T = \emptyset$, $F \subseteq (P \times T) \cup (T \times P)$ is the set of arcs, and $M : P \rightarrow \mathbb{N}$ is the initial marking, which is a bag of places; each place $p \in P$ contains in the initial state $M(p)$ tokens. Standard definitions of Petri nets also use weights on arcs, but since weights are only useful for unsafe nets, we do not consider these.

2.2. Statecharts

Statecharts extend finite state machines with composite state nodes and event broadcasting [Har87]. As explained in the introduction, we do not focus on events, and therefore statechart transitions do not carry any label here.

Composite state nodes contain other state nodes. A composite node c directly contains another node n if all other nodes containing n contain c. For instance, in Fig. 1b composite node O4 directly contains p7 while A2 contains p7 but not directly.

There are two kinds of composite node: AND and OR. An AND node directly contains two or more orthogonal OR nodes, separated by dotted lines, which are active in parallel if the AND node is active. An OR node directly contains nodes that are exclusive: one of them is active if the OR node is active. For instance, if in Fig. 1b OR node O3 is active, either A1 or p6 is active. A node that is not composite is BASIC. All nodes in Fig. 1b that are prefixed with p are BASIC.

The (direct) containment relation can be visualised as a rooted AND/OR tree. If composite node x directly contains another node y, then x is parent of y in the tree. If composite node x indirectly contains another node y, then x is ancestor of y in the tree. Leaves of the tree are the innermost (BASIC) nodes of the statechart. Internal nodes of the tree are either AND nodes or OR nodes. For technical reasons, the root of the tree is always an OR node. The root is never shown in a statechart diagram.

Similar to Petri nets, nodes in a statechart can be connected by transitions, which we call hyperedges from now on to avoid confusion with transitions in a Petri net. Hyperedges can have input nodes (called source nodes) and output nodes (called target nodes). A hyperedge can have a non-BASIC node as source or target node. For hyperedge h, set $source(h)$ denotes the set of source nodes of h while $target(h)$ the set of target nodes. It is required that each pair of nodes in $source(h)$ and each pair of nodes in $target(h)$ are *orthogonal*, that is, given two different sources (targets), the smallest node containing both sources (targets) is an AND node. In Fig. 1b, p2 and p4 are orthogonal since the smallest node containing both is AND node A1. We adopt the UML notation [UML03b]: a hyperedge with a single source node and a single target node is visualised as a simple directed edge, while a hyperedge having more than one source or target node is visualised as a bar having incoming and outgoing edges.

A state C of a statechart, called a *configuration*, is a maximal set of nodes that the system can be in simultaneously. Configurations for the statechart in Fig. 1b are for example {p1, root} and {p3,p4,p7,O1,O2,O3, O4,A1,A2,root}. Each configuration C has to satisfy the following three constraints:

- if a non-root node is in C, its parent is in C too,
- if an AND node is in C, all its children are in C too,
- if an OR node is in C, one of its children is in C too.

A hyperedge is enabled in configuration C if all its source nodes are in C. However, the computation of the state reached after taking the hyperedge is more involved than for Petri nets, since the next state has to satisfy the configuration constraints. First, all nodes below the scope of h are left, so they are removed from the current configuration. The scope of h is the smallest OR node that contains all the input and output nodes of h, i.e., all other OR node that contain all the input and output nodes also contain the scope of h. The scope of hyperedge t2 in Fig. 1b is O1 while the scope of t1 is root.

Next, the targets of h are added to the state. If the resulting state is not a configuration, then $target(h)$ is not complete. For instance, suppose we add to Fig. 1b a hyperedge h with source node p1 and target node A2. The target set of h is incomplete, since {A2,root} is not a configuration, as it contains AND node A2, but not any children of A2.

By computing the default completion of a partial configuration, it can be extended to a configuration. The default completion of a partial configuration X, denoted $dcomp(X)$, is the configuration Y that contains X such that for each OR node $o \in Y$, if Y does not contain any child of o, then Y contains the default node of Y [PS91]. The default completion for X is unique. For instance, the default completion of {A2,root} is configuration {p6,p7,O3,O4,A2,root}.

Formally, a statechart is a tuple $(N, BN, AN, ON, H, source, target, child, default, r)$. Set N contains the nodes, set H the hyperedges, where $N \cap H = \emptyset$. Set N is partitioned into a set BN of BASIC nodes, AN of AND nodes, and ON of OR nodes. Functions $source, target : H \rightarrow \mathcal{P}(N)$ specify for each hyperedge the non-empty sets of input nodes and output nodes, respectively. Predicate $child \subseteq N \times N$ relates a node to its parent node, so $(x, y) \in child$ means x is child of y. The $child$ predicate arranges the nodes in a tree of which the root is OR node r and of which the leaves are BASIC nodes. Function $default$ specifies for each OR node which child node is entered by default when computing the default completion. The initial configuration is the default completion of set $\{r\}$.

The standard definition of statecharts in the literature [Esh09a, HPSS87, PS91] is slightly different: it uses a function that specifies for each node its type (instead of partitioning the set of nodes) and a function that specifies for each node its set of child nodes (instead of a *child* predicate). The formalisation we use is equivalent and considerably simplifies the definition of the translation in Sect. 6.

3. Correspondence between Petri nets and statecharts

When translating a Petri net to a statechart, the resulting statechart should be similar to the original Petri net. To define similarity, we consider both the structural and the behavioural aspect of Petri nets and statecharts. We use both kinds of similarity to define a correspondence relation between Petri nets and statecharts.

Structural similarity

A translation must map a Petri net to a structurally similar statechart, i.e., its syntactic structure must resemble the net structure. To define this precisely, we first have to relate Petri net and statechart syntax; see Table 1. A Petri net place corresponds to a statechart BASIC node while a Petri net transition corresponds to a statechart hyperedge. However, composite (AND/OR) statechart nodes have no counterpart in Petri net syntax.

Definition 3.1 A Petri net and a statechart are *structurally similar* if there is an isomorphism f that maps each place p to BASIC node $f(p)$ and each transition t to hyperedge $f(t)$ such that the flow relation F is preserved by the $source$ and $target$ functions, so $(p, t) \in F$ if and only if $f(p) \in source(f(t))$ and $(t, p) \in F$ if and only $f(p) \in target(f(t))$.

The definition does not refer to markings and default nodes; these are referred to in the next definition.

Table 1. Relating Petri net and statechart syntax

Petri net	Statechart
Place	BASIC node (BASIC state)
–	AND node
–	OR node
Transition	Hyperedge (compound transition)

Behavioural similarity

A translation from Petri nets to statecharts must preserve the behaviour of each input net [GK07], neither reducing nor adding behaviour. Otherwise, the behaviour specified in the original Petri net is changed in the statechart as a side effect of applying the translation, allowing errors to creep in.

The behaviour of both Petri nets and statecharts is defined in terms of transition systems. States in the transition systems are markings (for Petri nets) or configurations (for statecharts). A state change represents firing a transition (Petri nets) or taking a hyperedge (statecharts). Formal definitions are presented in the Appendix A.

Definition 3.2 A Petri net PN is *behaviourally similar* to a statechart SC if and only if the transition systems $TS(PN)$ and $TS(SC)$ are isomorphic, where $TS(PN)$ denotes the transition system of PN and $TS(SC)$ the transition system of SC.

Correspondence

We use both similarity notions to define a correspondence relation between Petri nets and statecharts.

Definition 3.3 A Petri net *corresponds* to a statechart if the Petri net is structurally and behaviourally similar to the statechart.

For example, the net in Fig. 1a corresponds to the statechart in Fig. 1b.

Not every Petri net has a corresponding statechart, for the following reason. Each configuration contains by definition each BASIC node at most once. By definition of structural similarity, one BASIC node relates to one place. Therefore, each configuration corresponds to a safe marking, in which each place is marked with at most one token. Therefore only safe nets can be translated to corresponding statecharts. However, even some safe nets do not have corresponding statecharts (for instance Fig. 2 in Sect. 4). The next section defines statechartable nets, a subclass of Petri nets that do have corresponding statecharts.

A Petri net can correspond to multiple statecharts. If a Petri net and statechart correspond, composite nodes can be inserted or removed in the statechart while preserving correspondence. For example, in the statechart in Fig. 1b an OR node can be inserted that is child of OR node O1 and parent of BASIC node p3. The resulting statechart still corresponds to the net in Fig. 1a. The translation defined in Sect. 6 constructs a minimal corresponding statechart for an input net.

Not every statechart has a corresponding Petri net, as we explain in Sect. 5. Next, a statechart can correspond to at most one Petri net. If two Petri nets correspond to the same statechart, they have isomorphic structures by Definition 3.1 and have identical initial markings by Definition 3.2 and are therefore equal.

4. Statechartable nets

We structurally define statechartable nets as a subclass of safe Petri nets. The translation defined in Sect. 6 maps each statechartable net to a corresponding statechart.

We first introduce the novel Petri net notion of an area. Next, we introduce constraints on areas. We use these constraints to define statechartable nets. Finally, we discuss the completeness and expressiveness of the class of statechartable nets.

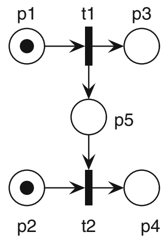


Fig. 2. Petri net whose covers are not nestable

4.1. Areas

An area resembles a thread of control, as we explain below after presenting the definition and a few examples.

Definition 4.1 (*Area*) Let (P, T, F, M) be a Petri net. An *area* is a non-empty set $X \subseteq P$ of places such that for every transition $t \in T$,

- if area X contains all input places of t, then X contains all output places of t, so $\bullet t \subseteq X \Rightarrow t\bullet \subseteq X$, and
- if area X contains all output places of t, then X contains all input places of t, so $t\bullet \subseteq X \Rightarrow \bullet t \subseteq X$.

Combining these two if-clauses, we have that X is an area if and only if for every $t \in T$, $\bullet t \subseteq X \Leftrightarrow t\bullet \subseteq X$.

For the net in Fig. 1a, set {p2,p3} is an area: it contains all input and output places of t2, but for every transition t other than t2 it does not contain all input and all output places of t. As a negative example, set {p3,p5,p6} is not an area because it violates the area constraints for t2 and t3: it contains their output places but not their input places p2 and p4. If these places are added, the set is an area.

An area resembles a thread of control that starts at a transition that has some output places in the area, but not all, and ends at a transition that has some input places in the area, but not all. For instance, the thread of control for the area {p2,p3} starts at t1 and ends at t3. Once started, an area does not need to synchronise with other areas to make progress by firing transitions. Areas correspond to OR nodes, as the translation defined in Sect. 6 will show.

In the sequel, we frequently use the minimal area that includes a given set of places. Let X be a non-empty set of places. The *minimal area* for X, denoted $minArea(X)$, is the smallest area that includes all places in X. The minimal area for X is unique. For instance, for $X = \{p2\}$ the minimal area is {p2,p3}, which is also a minimal area for $X = \{p2, p3\}$. The minimal area for $X = \{p2, p7\}$ is {p2,p3,p7}.

4.2. Constraints

Statechartable nets are defined in terms of three constraints on areas. Each constraint is illustrated with examples nets that violate it. The presented example nets have no corresponding statecharts.

Nestable covers

An area can contain other areas. For instance, for the net in Fig. 1a the area {p2,p3,p4,p5,p6} contains areas {p2,p3} and {p4,p5}. This resembles a subthread relation: the thread for {p2,p3,p4,p5,p6} spawns subthreads for {p2,p3} and {p4,p5}, but these subthreads run in parallel and are independent, i.e., they do not need to synchronise with each other to make progress by firing transitions. Two areas are in parallel if they are disjoint and contain places that are either input or output places of the same transition t, for instance {p2,p3} and {p4,p5} in Fig. 1 are started by t1.

However, the net in Fig. 2 has two parallel areas that do require cross-synchronisation: t2 can only fire after t1 has fired. The net in Fig. 2 has no corresponding statechart, since statecharts cannot express cross-synchronisation between parallel OR nodes. Therefore, we wish to define a constraint that rules out parallel areas that cross-synchronise.

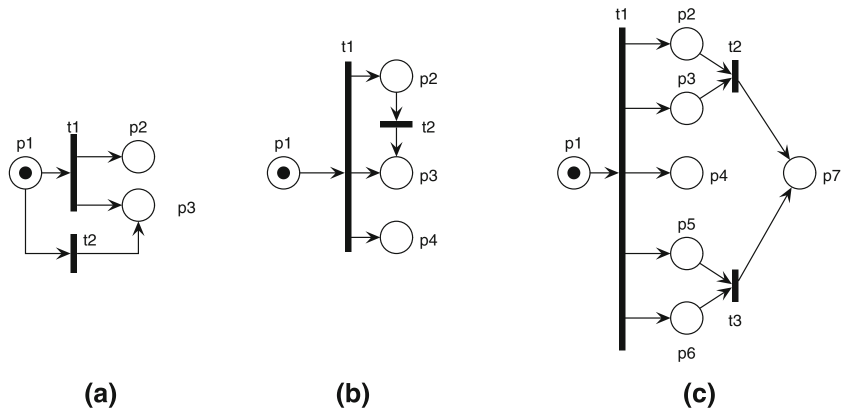


Fig. 3. Petri nets with inconsistent areas

To define such a constraint, we first need to identify the set of places included (visited) in a set of parallel areas (subthreads). Let $X \subseteq P$ be a non-empty set of places that is not singleton. The *cover* of X is the union of each minimal area of a strict, non-empty subsets of X:

$$cover(X) = \bigcup_{Y \mid \emptyset \subset Y \subset X} minArea(Y).$$

For instance, for the net in Fig. 1a $cover(\{\mathsf{p2}, \mathsf{p4}\}) = \{\mathsf{p2}, \mathsf{p3}, \mathsf{p4}, \mathsf{p5}\}$. Note that $minArea(\{\mathsf{p2}, \mathsf{p4}\}) = \{\mathsf{p2}, \mathsf{p3}, \mathsf{p4}, \mathsf{p5}, \mathsf{p6}\}$, so *cover* does not yield an area.

The following constraint rules out parallel areas that cross-synchronise.

Definition 4.2 A Petri net PN has *nestable covers* if and only if for every $X, Y \subseteq P$ such that X and Y are non-singleton and preset or postset of some transitions, $cover(X) \cap cover(Y) \neq \emptyset$ implies $cover(X) \subseteq cover(Y)$ or $cover(Y) \subseteq cover(X)$.

The net in Fig. 2 has no nestable covers: $cover(\mathsf{t1}\bullet) = \{\mathsf{p3}, \mathsf{p5}\}$ and $cover(\bullet\mathsf{t2}) = \{\mathsf{p2}, \mathsf{p5}\}$ are not nestable. However, the net in Fig. 1a does have nestable covers, since $cover(\bullet\mathsf{t4})$, which is $\{\mathsf{p2}, \mathsf{p3}, \mathsf{p4}, \mathsf{p5}, \mathsf{p6}\}$, is a subset of $cover(\mathsf{t1}\bullet)$, which is the set of all places.

Consistent areas

In some nets, areas overlap that should be in parallel. For instance, in the nets in Fig. 3a, b, transition t1 starts two parallel areas for p2 and p3, but due to t2 the area of p2 is included in (a) or equal to (b) the area of p3. The problem in Fig. 3a, b is caused by the fact that the minimal areas of places p2 and p3 overlap. Requiring that two distinct places in the same preset or postset should have disjoint minimal areas rules out the nets in Fig. 3a, b, but is not sufficient in general, as illustrated by the example in Fig. 3c. The minimal areas of among others p2 and p6 are disjoint. However, the minimal areas of {p2,p3} and {p5,p6} do overlap, since both are equal to the minimal area of p7. Due to t1, the minimal areas should be disjoint.

We define the following constraint that rules out nets such as the ones in Fig. 3.

Definition 4.3 A transition t has *consistent areas* if and only if for every set $X, Y \subseteq P$ such that $X \cup Y \subseteq \bullet t$ or $X \cup Y \subseteq t\bullet$, if $X \cap Y = \emptyset$ then $minArea(X) \cap minArea(Y) = \emptyset$. A Petri net PN has consistent areas if and only if each transition has consistent areas.

Configurable markings

Places marked in the initial marking M should be complete: their minimal area should include each place of the net, to prevent that some parts of the net are not reachable from M. For instance, the marking in Fig. 4a is not complete, since p3 and p4 are not in the minimal area induced by the initial marking.

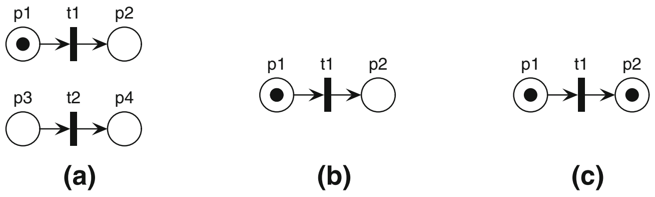


Fig. 4. Petri nets with non-configurable markings

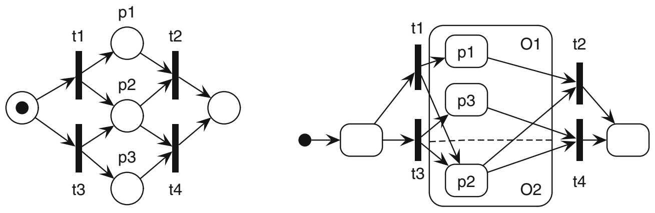


Fig. 5. Non-statechartable net and corresponding statechart

Definition 4.4 Let (P, T, F, M) be a Petri net and let $P_M = \{ p \in P \mid M(p) = 1 \}$. Marking M is *complete* if and only if $minArea(P_M) = P$.

Still, we need a stronger constraint, since M can be over-complete: reducing M by removing tokens from initial places then results in a complete marking. The Petri nets in Fig. 4b, c have over-complete markings. In both cases, if one token is removed, the resulting net still has a complete marking. In other words, a complete marking should be minimal.

Definition 4.5 Let $PN = (P, T, F, M)$ be a Petri net. Marking M is *configurable* (or PN has a configurable initial marking) if and only if M is complete and every submarking M', so $M' \sqsubset M$, is not complete.

The translation in Sect. 6 ensures that a net with a configurable initial marking translates into a statechart whose initial configuration corresponds to the initial marking.

4.3. Definition

We have defined three constraints on Petri nets: nestable covers, consistent areas and configurable markings. Each constraint is illustrated with nets that violate the constraint but satisfy the other two constraints. The constraints are therefore independent from each other, so each constraint is necessary.

Using these three constraints, we can now formally define statechartable nets.

Definition 4.6 (*Statechartable net*) A Petri net (P, T, F, M) is *statechartable* if and only if (P, T, F, M) has nestable covers, consistent areas and a configurable initial marking.

In Sect. 6 we define a sound and complete translation from statechartable nets to corresponding statecharts.

4.4. Incompleteness

We have defined the class of statechartable nets. We show in Sect. 6 that each statechartable net has a corresponding statechart. Naturally, there is also the class of Petri nets that have corresponding statecharts. This class by definition includes statechartable nets, but the question is whether statechartable nets coincide with this class. Phrased differently: if a Petri net has a corresponding statechart, is the net statechartable, so is the class of statechartable nets complete?

The answer is negative: there are Petri nets that are not statechartable, yet do have a corresponding statechart. Therefore, the class of statechartable nets is incomplete. For instance, the Petri net in Fig. 5 is not statechartable, since the covers of non-singleton presets and postsets {p1,p2} and {p2,p3} are not nestable. A corresponding statechart does exist, as shown in the same figure. Note that in the statechart the BASIC nodes p1 and p3 contained inside OR node O1 are only connected by hyperedges that leave or enter O1. To further illustrate this, Fig. 6 shows a statechartable net that is a slight extension of the net in Fig. 5: a transition has been added that connects places p1 and p3.

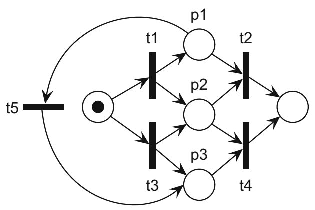


Fig. 6. Statechartable net that extends the net in Fig. 5

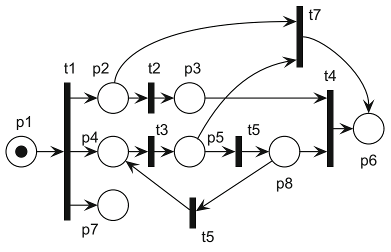


Fig. 7. Statechartable net with goto-like behaviour

This incompleteness does not imply that the class of statechartable nets is overly restrictive. For instance, each non-statechartable net shown in Figs. 2, 3 and 4 has no corresponding statechart. Moreover, this incompleteness does not seem to be a severe limitation in practice. A common though unwritten rule of thumb for designing statecharts is to group only related (connected) BASIC nodes in an OR node, as can be inferred from the many statechart examples in the literature, e.g. [Esh09a, Har87, HN96]. Therefore, statecharts corresponding to non-statechartable nets do not occur in practice. A much more obvious translation for the Petri net in Fig. 5b is to use statecharts with overlapping [HK92] and to construct a statechart with two overlapping AND nodes, one for cover {p1,p2} and one for cover {p2,p3}. Extending the translation defined in Sect. 6 to construct statecharts with overlapping is part of future work.

4.5. Expressiveness

This section already showed several examples of non-statechartable nets, while the main example of a statechartable net is presented in Fig. 1a. This example exhibits a high degree of (block-)structuredness, since it does not contain choices or loops. In the corresponding statechart in Fig. 1b, no goto-like constructs are used: for example OR nodes O1 and O2 each have a single entry and a single exit point. This may suggest that statechartable nets are close to block-structured nets, each block having a single entry and a single exit point.

To counter this suggestion, we present another statechartable net in Fig. 7, modified from Fig. 1a. Transition t7 leaves the loop headed by p4 in a goto-like way. The net is statechartable; the statechart constructed for this example is similar to the one in Fig. 1b, where p8 becomes child of O2. The example shows that also complex, unstructured nets that have a mixture of choices and loops can be statechartable.

Statechartable nets are a new class of nets that do not coincide with any of the existing classes. The most closely related class are state machine decomposable nets [BdC92], which are also called state machine coverable or S-coverable nets in the literature. A state machine decomposable net can be decomposed into state machines, which are sequential nets not having any parallelism, so each transition has one input place and one output place. If a state machine contains a place of the original net, it must contain all input and output transitions of that place in the net too. For instance, in Fig. 1a, if a state machine component includes p2, it must also contain t1 and t2. The net in Fig. 1a can be decomposed into three state machine components, one of which contains elements p1,t1,p7. Elsewhere [Esh05] we have proven that the net underlying a statechart is always state machine decomposable: for each maximal set of BASIC nodes in which each pair of nodes is non-orthogonal, a state machine can be generated from the corresponding places. For instance, in Fig. 2 set {p1,p7} is maximal non-orthogonal: each other BASIC node is not orthogonal to either p1 or p7. Each statechartable net is a state machine decomposable net. However, there are many state machine decomposable nets, for instance the ones in Figs. 2 and 5, that are not statechartable.

5. Statecharts corresponding to statechartable nets

In the previous section we defined statechartable nets. In this section we define the class of statecharts that correspond to statechartable nets.

Consider an arbitrary statechart that corresponds to a statechartable net. The statechart has the following two syntactic features. First, due to behavioural similarity (Definition 3.2) the transition systems of the net and the corresponding statechart are isomorphic. Take an arbitrary reachable marking M of the net and let C be the corresponding configuration of the corresponding statechart. Let h be a hyperedge that is enabled in C. Hyperedge h must only leave and enter BASIC nodes that are in its source and target set; otherwise, the configuration reached after taking h in C does not correspond to the marking reached by firing isomorphic transition $f^{-1}(h)$ in M. This implies that in the statechart no additional BASIC nodes are left or entered by taking h. This corresponds to the Petri net locality principle [DJ01] that states that each transition can only consume tokens from places in its preset and only produce tokens for places in its postset.

The following constraint characterises such a hyperedge. As explained in Sect. 2, a hyperedge h leaves and enters BASIC nodes contained inside OR node $scope(h)$. Hyperedge h is *complete* if and only if for each BASIC node $n \in BN$ that is contained inside $scope(h)$,

- if $n \notin source(h)$, there is a BASIC source node $s \in source(h)$ such that n and s are not orthogonal;
- if $n \notin target(h)$, there is a BASIC target node $t \in target(h)$ such that n and t are not orthogonal.

Recall from Sect. 2 that if two BASIC nodes n, n' are not orthogonal, then no configuration can contain both n and n'.

To illustrate this definition: in Fig. 1a all hyperedges are complete. If a hyperedge h would be inserted with $source(h) = \{\mathsf{p4}\}$ and $target(h) = \{\mathsf{p7}\}$, then h is not complete: if h is taken in a configuration that contains BASIC nodes p3, p4, and p7, then all nodes below $scope(h) = \mathsf{root}$, are left, including p3, and p6 (default of O3) is entered.

Second, BASIC nodes contained inside an OR node are weakly connected by hyperedges in the OR node: for every pair of BASIC nodes that is contained in the OR node, there is an undirected path of hyperedges inside the OR node that connect the BASIC nodes. So each hyperedge in the path does not leave or enter the OR node and has some source or target node in common with its predecessor and successor in the path. For instance, in Fig. 1b BASIC nodes p3 and p4 are connected by undirected path t4,t3. As explained in Sect. 4.4, this feature is specific to statecharts corresponding to statechartable nets but is also a design heuristic for statecharts that is implicitly used in practice. Formally, we call an OR node *connected* if and only if for every pair x, y of BASIC nodes contained inside o there is a path of hyperedges $h_1, h_2, .., h_n$ from x to y, so $x \in source(h_1)$ and $y \in target(h_n)$ and $target(h_i) \cap source(h_{i+1}) \neq \emptyset$ for $0 \leq i < n$, such that for each hyperedge in the path its scope is either contained in o or equal to o.

The next definition summarises both features.

Definition 5.1 (*Complete and OR-connected statechart*) Let $SC = (N, BN, AN, ON, H, source, target, child, default, r)$ be a statechart. If each hyperedge $h \in H$ is complete and each OR node $o \in ON$ is connected, then statechart SC is *complete and OR-connected*.

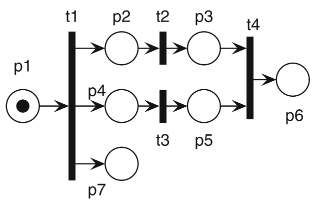
The next section presents a translation from statechartable nets to complete and OR-connected statecharts.

6. Translating statechartable nets to corresponding statecharts

The previous sections defined statechartable nets and their corresponding statecharts. This section defines a declarative translation $PNtoSC$ from statechartable nets to corresponding statecharts, which are complete and OR-connected. The translation is proven sound and complete for statechartable nets, i.e., $PNtoSC$ constructs for an input net a corresponding statechart if and only if the net is statechartable. Finally, we discuss a few alternative translations.

6.1. Definition

To relate a statechartable net to a corresponding statechart, we have to use an isomorphism f on the syntax of Petri net and statecharts according to Definition 3.1. To simplify the definition of the translation, we use the identify function $=$ for f, so $PNtoSC$ translates each Petri net (P, T, F, M) into a statechart $(N, BN, AN, ON, H, source, target, child, default, r)$ such that $BN = P$, $H = T$ and for each hyperedge $h \in H$, $source(h) = \bullet h$ and $target(h) = h\bullet$. Moreover, $N = BN \cup AN \cup ON$. The main difficulty, therefore, is the definition of AND/OR nodes, so sets AN and ON, relation $child$, function $default$, and the root node r.



minimal areas

$o_1 = \{\mathsf{p2}, \mathsf{p3}\}$
$o_2 = \{\mathsf{p4}, \mathsf{p5}\}$
$o_3 = \{\mathsf{p2}, \mathsf{p3}, \mathsf{p4}, \mathsf{p5}, \mathsf{p6}\}$
$o_4 = \{\mathsf{p7}\}$
$o_5 = P = \{\mathsf{p1}, \mathsf{p2}, \mathsf{p3}, \mathsf{p4}, \mathsf{p5}, \mathsf{p6}, \mathsf{p7}\}$

Fig. 8. Petri net of Fig. 1a (repeated) and its minimal areas for elements in $P \cup T$

We define these elements of the statechart tuple in terms of areas. In the sequel, we use the following shorthands: given a place $p \in P$ and transition $t \in T$, $minArea(p)$ abbreviates $minArea(\{p\})$ while $minArea(t)$ abbreviates $minArea(\bullet t \cup t\bullet)$.

OR nodes

Each OR node is defined by an area, where each place in the area corresponds to a BASIC node inside the OR node. However, not every area becomes an OR node: for example, for the net in Fig. 1a set {p2, p3, p7} is an area, but the set does not correspond to any OR node. The closest OR node is O4, but this corresponds to area {p2,p3,p4,p5,p6,p7}.

For each place $p \in P$, an OR node $minArea(p)$ is created, which acts as OR parent of BASIC node p in the statechart AND/OR tree. For example, for the net in Fig. 1a, the OR parent of p2 is {p2, p3}. Furthermore, for each transition $t \in T$ an OR node $minArea(t)$ is created. This OR node acts as scope of hyperedge t in the statechart translation. For example, the OR node for transition t2 in Fig. 1a becomes {p2, p3} while the OR node for t4 becomes {p2, p3, p4, p5, p6}. Note that for area {p2, p3, p7}, there is no element $e \in P \cup T$ such that $minArea(e) = \{\mathsf{p2}, \mathsf{p3}, \mathsf{p7}\}$.

Special area is the set P, which is the OR root node of the constructed AND/OR tree. If the Petri net is not connected, then there is no element $e \in P \cup T$ that has P as minimal area, so P needs to be included separately. For instance, the net in Fig. 4a has no element that generates minimal area P.

The set ON of OR nodes is therefore defined as the union of the set of the minimal areas created for each element $e \in P \cup T$ and set $\{P\}$:

$$ON = \{\ minArea(e)\ \mid\ e \in P \cup T\ \} \cup \{P\}.$$

Figure 8 shows the elements of set ON for the net in Fig. 1. Set o_1 corresponds to O1 in Fig. 1b, set o_2 to O2, etc. Note that different elements in $P \cup T$ can share the same minimal areas, for instance $minArea(\mathsf{p1}) = minArea(\mathsf{t1}) = o_5$.

AND nodes

Each AND node $a \in AN$ is defined as a set of minimal areas $\{x_1, x_2, .., x_n\}$. Each minimal area x_i in the set is an OR node that is child of a. For instance, for Fig. 8a we can infer from the statechart in Fig. 1b that we need to create an AND node $\{o_1, o_2\}$, which specifies that o_1 and o_2 are executed in parallel.

Of course, the question is how to infer AND nodes from the structure of the Petri net. First, observe that an AND node a needs to be constructed for each non-singleton set X that is preset or postset of a transition. We use the set $cover(X)$, defined in Sect. 4, to identify which areas have to become children of a. Set $cover(X)$ contains all BASIC nodes that have to be nested inside a. Each OR child o of a has to be a subset of $cover(X)$, to ensure that o does not contain a BASIC node that is not in $cover(X)$. Furthermore, o has to be a *strict* subset of $cover(X)$, since an OR child of an AND node never contains all BASIC nodes inside the AND node. Next, to allow AND nodes nested inside a, o must be *maximal*: there is no area o' such that $o \subset o' \subset cover(X)$. For the example in Fig. 8, the maximal areas that are subsets of $cover(\mathsf{t1}\bullet)$ are o_3 and o_4. Thus, the constructed AND node $a = \{o_3, o_4\}$. Note that for example area o_1 is also a strict subset of $cover(\mathsf{t1}\bullet)$, but is not maximal, since $o_1 \subset o_3$. Area o_1 is child of AND node $\{o_1, o_2\}$, which is nested in a.

Next, we define a function $andNode$ that takes a set Y of BASIC nodes and a set of OR nodes ON and returns an AND node that contains exactly the BASIC nodes in Y:

$$andNode(Y, ON) = \{\ o \in ON\ \mid\ o \subset Y \wedge \text{there does not exist } o' \in ON : o \subset o' \subset Y\ \}.$$

If X is a non-singleton preset or postset, then $andNode(cover(X), ON)$ is the desired AND node for X.

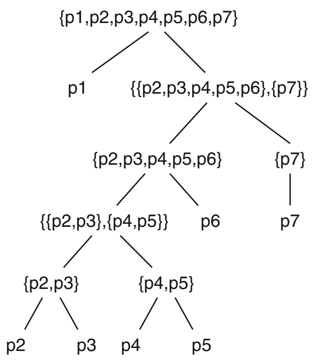


Fig. 9. AND/OR tree constructed by $PNtoSC$ for Fig. 1

If a Petri net is not connected, then P is not the minimal area of any place or transition. In that case, the unconnected components should be grouped in an AND node that is child of root P. For instance, consider the net in Fig. 4a with an additional token in p3: an AND node needs to be created that puts OR nodes {p1,p2} and {p3,p4} in parallel. In that case, there is no non-singleton preset or postset that can be used to construct the AND node. Instead, we invoke function $andNode$ with as first parameter P. This defines an AND node of which the children are the top-level (maximal) OR nodes of the unconnected components. For instance for the net in Fig. 4a function $andNode(P, ON)$ returns AND node {{p1,p2},{p3,p4}}.

To summarise, the set AN of AND nodes is defined as:

$$AN = \{\ andNode(Y, ON)\ \mid\ \text{there exists a non-singleton preset or postset} X \text{ such that } Y = cover(X) \\ \vee\ \ Y = P \text{ and there is no element } e \in P \cup T \text{ such that } minArea(e) = P\ \}.$$

For the examPLE net in Fig. 8, set AN contains AND nodes $\{o_1, o_2\}$, which corresponds to A1 in Fig. 1b, and $\{o_3, o_4\}$, which corresponds to A2.

The child relation

The *child* relation follows in a straightforward way from the definition of the nodes. A BASIC node $n \in BN$ is child of area $o \in ON$ if and only if $o = minArea(p)$. An OR node $o \in ON$ is child of an AND node $a \in AN$ if and only if $o \in a$. Defining the parent of an AND node a is more involved. Set $\bigcup_{x \in a} x$ contains all the BASIC nodes that are contained in a. The minimal area of this set, so $minArea(\bigcup_{x \in a} x)$, is the OR parent of a. For instance, the parent of AND node $\{o_1, o_2\}$ is o_3 since $minArea(o_1 \cup o_2) = o_3$.

Relation *child* is defined as follows:

$$\begin{aligned} child\ =\ & \{\ (p, o) \in P \times ON\ \mid\ minArea(p) = o\ \} \\ \cup\ & \{\ (o, a) \in ON \times AN\ \mid\ o \in a\ \} \\ \cup\ & \left\{\ (a, o) \in AN \times ON\ \mid\ minArea\left(\bigcup_{x \in a} x\right) = o\ \right\}. \end{aligned}$$

The AND/OR tree shown in Fig. 9 visualises the *child* relation constructed by the translation for the net in Fig. 1.

The default function

First, we observe that since the translation has to return a statechart corresponding to the input net, we have to ensure that each hyperedge is complete (cf. Sect. 5). If h is complete, then default nodes are irrelevant when h is taken. That is, if h enters OR node o, so $scope(h)$ contains o and o contains a target node of h, then the next configuration never contains the default node of o when h is taken. For instance, in Fig. 1b hyperedge t1 enters OR node O3. The default node of O3, BASIC node p6, is irrelevant: if t1 is taken then the next configuration does not include p6.

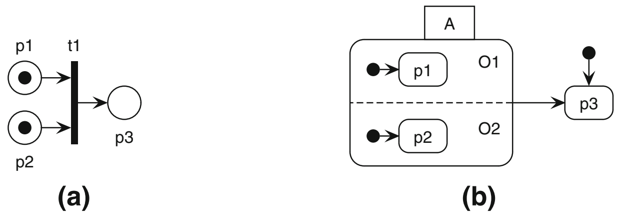


Fig. 10. Petri net and statechart that are structurally similar but have different initial states

Since default nodes are to be irrelevant, we can define an arbitrary, total function $auxdefault : ON \to BN \cup AN$, which assigns to each OR node one arbitrary child (by construction either a BASIC node or AND node) as default node:

$$auxdefault = \{n \mapsto c_1 \mid n \in ON \wedge c_1 \in BN \cup AN \wedge children(n) = \{c_1, c_2, .., c_k\}\}$$

However, due behavioural similarity (Definition 3.2) the initial marking and the initial configuration of the statechart should correspond: each place in the initial marking should relate to a BASIC node in the configuration and vice versa. The initial configuration is defined as the default completion of root node r, so $dcomp(\{r\})$, which is a set that contains default nodes. For each place p contained in the initial marking, the corresponding BASIC node must act as default node of its OR parent in the statechart, to ensure that BASIC node p is included in $dcomp(\{r\})$. Moreover, if BASIC node p is nested inside an AND node y and y is child of an OR node x, then y has to be the default node of x. To see why: the statechart in Fig. 10 has an initial configuration that contains BASIC node p3, since the default node of root r is p3. But the initial marking of the net contains places p1 and p2. In the statechart, the default node of r should be A to ensure that the default completion contains p1 and p2.

To define default nodes according to the initial marking, we use partial function $initdefault : ON \to BN \cup AN$, which assigns to each OR node x that contains one of the places p in the initial marking, as default node the unique child y of x that either equals p or contains p. By definition of *child*, each OR node x has only BASIC and AND children.

$$initdefault = \{x \mapsto y \mid x \in ON \wedge y \in AN \cup BN \wedge (y, x) \in child \wedge \exists p \in P : M(p) = 1 \wedge (p, y) \in child^*\}$$

Relation $child^*$ denotes the reflexive-transitive closure of *child*: if $(x, y) \in child^*$ then y contains x or $x = y$. Note that therefore it is possible that $y = p$ in the definition above. Since $(y, x) \in child$ and $(p, y) \in child^*$ implies $(p, x) \in child^*$, we do not need to test for $(p, x) \in child^*$. For the net in Fig. 1a, $initdefault = \{(r, \mathsf{p1})\}$ while for the net in Fig. 10, $initdefault = \{(r, \mathsf{A}), (\mathsf{O1}, \mathsf{p1}), (\mathsf{O2}, \mathsf{p2})\}$.

The actual default function is total function *auxdefault* overridden with partial function *initdefault*:

$$default = auxdefault \oplus initdefault.$$

Thus, for OR nodes in the domain of *initdefault* the default nodes contain or equal BASIC nodes that correspond to the places in the initial marking, while for the other OR nodes the default nodes are arbitrarily defined.

Root

The root is defined to be set P, which is an area. For the example in Fig. 8, set P equals area o_5, which corresponds to root in Fig. 1b.

6.2. Correctness

We show the correctness of the translation *PNtoSC* by proving two theorems. The proofs can be found in Appendix B.

In Sect. 4, we defined the class of statechartable nets. The first theorem shows that statechartable nets are precisely the class of nets for which *PNtoSC* returns a corresponding statechart.

Theorem 6.1 Petri net (P, T, F, M) is statechartable if and only if tuple $PNtoSC((P, T, F, M))$ is a statechart that corresponds to (P, T, F, M).

According to the first theorem, translation $PNtoSC$ fails if (P, T, F, M) is not a statechartable net: it then either returns a tuple that is not a statechart (nets in Figs. 2 and 3) or returns a statechart that does not correspond to (P, T, F, M) (nets in Fig. 4). Therefore, $PNtoSC$ is sound and complete for statechartable nets.

The second theorem shows that statecharts constructed by $PNtoSC$ are complete and OR-connected.

Theorem 6.2 If Petri net (P, T, F, M) is statechartable then $PNtoSC((P, T, F, M))$ is a complete and OR-connected statechart.

The reverse implication is not true. If (P, T, F, M) is not statechartable, then for nets with non-configurable markings such as in Fig. 4 $PNtoSC$ can still construct a complete and OR-connected statechart. But in that case the statechart does not correspond to the input net.

6.3. Alternative translations

The translation constructs for an input net a minimal statechart, in which each constructed OR node does not have another OR node as child, but only BASIC nodes and/or AND nodes. As mentioned in Sect. 3, each statechart constructed by the translation can be extended to other statecharts that also correspond to the input net.

The translation has been defined declaratively. The translation can equivalently be defined in an operational way using reduction rules on Petri nets. Such an operational translation is more efficient to compute and easier to implement than the declarative translation, but more difficult to link to the definition of statechartable nets than the declarative translation. In previous work [Esh09b], we defined and implemented an operational translation for a restricted form of statechartable nets. That operational translation has been implemented using model transformation technology [VE10].

It is straightforward to define a reverse translation from complete and OR-connected statecharts to corresponding statechartable nets. Places and transitions of the net are the BASIC nodes and hyperedges of the input statechart, respectively. Using Definition 3.1 the flow relation can be easily derived. The initial marking contains the places whose BASIC nodes are contained in the initial configuration of the statechart. Such a translation would map the statechart in Fig. 1b to the Petri net in Fig. 1a. The translation $PNtoSC$ is much more intricate since it needs to define AND and OR nodes, which have no counterpart in Petri net syntax.

7. Related work

Only a few papers consider translations from Petri nets to statecharts [RK97, SNK99]. The only published work with a considerable amount of detail is a paper by Schnabel et al. [SNK99], who informally describe an interactive method to translate a safe Petri net into a StateFlow statechart [Mat]. Since the paper is written in German, we will describe their method elaborately.

The method consists of two main phases. In the first phase, the net is reduced by performing the following two steps. First, each linear sequence of places is aggregated into a singe place. No formalisation is presented, but from the text and the presented examples it becomes clear that each place in the sequence must have a single input and a single output transition in the net. Next, in the resulting net, sets of places with the same input and output transitions are aggregated into a single place, which represents an AND node. These steps are repeated until the net can no longer be reduced.

In the second phase, the reduced net is mapped to a statechart by mapping each place invariant of the net to a parallel OR node of the statechart. A place invariant is a set of places for which the sum of tokens contained in these places remains constant during execution. Each place in an invariant maps to a BASIC node in the corresponding parallel node. Since the same place can occur in several place invariants, it can translate into several BASIC nodes. Schnabel et al. outline some ways to prevent such duplications, but sometimes duplications cannot be avoided. Finally, for each BASIC node representing an aggregate place constructed in the first phase, the aggregated Petri net structure is inserted, which results in a statechart with additional nested states. To illustrate this, Fig. 11b shows the statechart translation according to Schnabel et al. for the net in Fig. 11a.

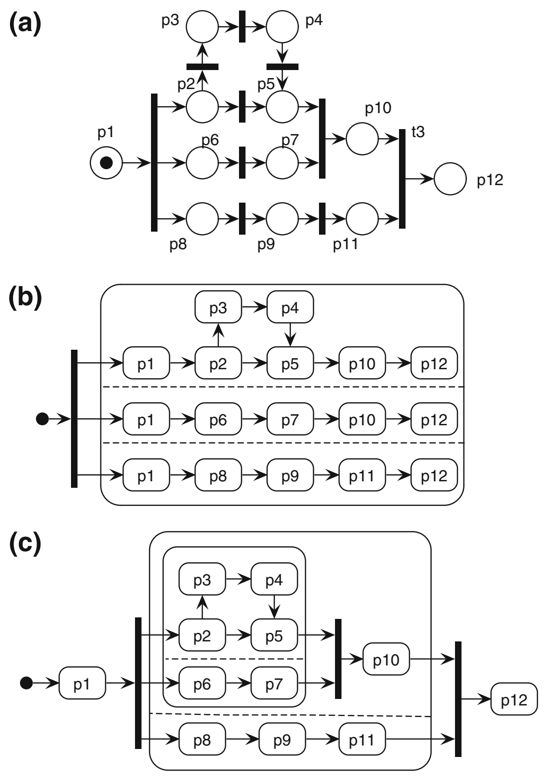


Fig. 11. Safe Petri net (**a**), skeleton of statechart translation according to Schnabel et al. [SNK99] (**b**), and statechart translation according to *PNtoSC* (**c**)

Duplicated BASIC nodes in a statechart correspond to a single place in the Petri net. To ensure that the behaviour of the input net resembles the behaviour of the statechart, duplicated nodes must be entered and left simultaneously. To achieve this, Schnabel et al. make use of event synchronisation and auxiliary variables (not shown in Fig. 11b). However, their solution is specific to StateFlow.

The translation of Schnabel et al. [SNK99] resembles ours to some extent but there are important differences. We have formally defined our translation, proven its correctness, and characterised the class of Petri nets for which the translation yields statecharts with equivalent behaviour, while Schnabel et al. [SNK99] only informally present their approach, do not give a formal correctness proof, and do not explicitly characterise the class of nets handled by their translation. Moreover, their translation is interactive, while ours is fully automatic. Finally, our translation does not duplicate any nodes.

Next, there is some other related work which has a different scope than our paper. For UML 1.x activity diagrams [UML03a], whose syntax resembles Petri net syntax, a balancedness constraint was defined to give them a semantics in terms of UML statecharts. However, we are unaware of any translation from Petri nets (or activity diagrams) to statecharts in which this constraint and the corresponding translation is formalised. A Petri net has balanced forks and joins if each fork is eventually followed by a join, and fork-join pairs are properly nested. (A fork is a transition with more than one output place, a join is a transition with more than one input place.) For such nets, each place translates to a BASIC node, each Petri net transition to statechart transition, and finally each fork-join pair to an AND node with corresponding OR children (see Fig. 12). Our translation does not require input nets to be balanced (cf. Fig. 7), so it is more general.

Other related work has considered the relation between statecharts and Petri nets. Kishinevsky et al. [KCK$^+$97] define a Petri net variant that incorporates some statechart features. The variant, called place chart net, uses hierarchy on places and preemptive transitions: a transition does not only empty its input places but also all descendant places of the input places. However, the relation between place chart nets and Petri nets is not formally analysed.

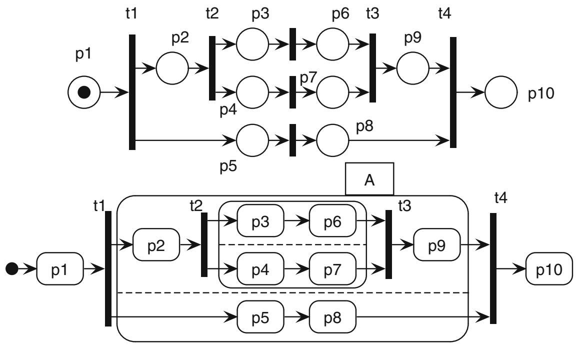


Fig. 12. Balanced Petri net and corresponding statechart

Drusinsky and Harel [DH94] show that a class of concurrency models that includes both statecharts and Petri nets is more succinct than finite state machines. However, they do not explicitly make a distinction between statecharts and Petri nets: these fall in the same class.

8. Conclusion

We have introduced statechartable nets, a subclass of safe Petri nets that can be translated to corresponding (i.e. similar) statecharts. Statechartable nets have been structurally defined using the novel notion of an area. We have shown that the class of statechartable nets is not complete, since some non-statechartable nets do have corresponding statecharts. However, such statecharts are not likely to be drawn in practice, so this incompleteness does not appear to be severe limitation in practice.

Next, we have defined a declarative, structural translation from statechartable nets to corresponding statecharts. The translation uses areas to infer statechart AND and OR nodes, which have no counterpart in Petri net syntax. The translation has been proven sound and complete for statechartable nets: it fails for non-statechartable nets. Elsewhere [Esh09a] we have presented an operational translation that has been implemented using model transformations [VE10].

We envision two specific applications of the translation. First, it can support the communication of a Petri net design to designer and end-users only familiar with statecharts. In particular, it can facilitate the automated exchange of models [Gra97, GK07, RK97] between different Petri net and statechart tools, thus enabling designers to use both Petri net and statechart tools for their designs.

Second, the translation can be used to synthesise statecharts. For instance, several mappings from message sequence charts to Petri nets exist [AB08, Klu03]. Combining such a translation with the translation defined in this paper, a statechart can be synthesised from a scenario-based specification. Approaches for synthesising a statechart from a scenario-based specification [HK02, WS00] produce typically statecharts that are basically sets of communicating sequential finite state machines, i.e., the only concurrency is at the top level of the statechart. The translation defined in this paper can construct statecharts in which concurrency occurs at arbitrary levels.

There are several directions for further work. An interesting question is to precisely characterise the class of Petri nets that correspond to statecharts. Another extension is to consider Petri nets with events or data, for instance coloured Petri nets [Jen92], and statecharts with local variables and action statements as in STATEMATE [HN96] and UML [UML03b]. Defining such a translation can be complicated, since coloured Petri nets use a functional programming language to express action statements, whereas statecharts use an imperative programming language.

Another direction is to consider statecharts with overlapping [HK92], which generalise statecharts by allowing the nodes to be arranged as a directed acyclic graph rather than a tree. In particular, it will be interesting to analyse how the conditions in the definition of statechartable nets can be relaxed to characterise the class of nets corresponding to statecharts with overlapping.

Acknowledgements

The comments of one of the anonymous referees helped to significantly improve the presentation.

A. Transition systems, Petri nets and statecharts

We recall definitions of the formalisms used in the paper: transition systems, Petri nets and statecharts. Readers familiar with Petri nets and statecharts can skip this section. Formal definitions of Petri nets can be found in [Mur89, Rei85] and of statecharts in [Esh09a, HPSS87, PS91], among others.

A.1. Transition systems

The execution semantics of both Petri nets and statecharts map into transition systems. A *transition system* is a tuple $(S, \rightarrow, init)$ where S is a set of states, $\rightarrow\ \subseteq S \times S$ the transition relation, and $init \in S$ is the initial state.

Let $(S_1, \rightarrow_1, init_1)$ and $(S_2, \rightarrow_2, init_2)$ be two transition systems. An *isomorphism* is a bijective function $h : S_1 \rightarrow S_2$ such that $h(init_1) = init_2$ and $(x, y) \in\ \rightarrow_1$ if and only if $(h(x), h(y)) \in\ \rightarrow_2$. Two transition systems $TS_1 = (S_1, \rightarrow_1, init_1)$ and $TS_2 = (S_2, \rightarrow_2, init_2)$ are *isomorphic* if and only if they are related by an isomorphism.

A.2. Petri nets

Syntax. A *Petri net* is directed, bipartite graph that consists of two types of nodes, places and transition. Places are represented by circles, transitions by bars. Formally, a Petri net (place/transition net) is a tuple (P, T, F), where

- P is a finite set of places,
- T is a finite set of transitions, $P \cap T \neq \emptyset$, and
- $F \subseteq (P \times T) \cup (T \times P)$ is a finite set of arcs, the flow relation.

Standard definitions of Petri nets also use weights on arcs, but since statecharts lack weights, we do not consider these here.

Given an element $e \in P \cup T$, its preset $\bullet e = \{\, x \mid (x, e) \in F \,\}$ is the set of input places and transitions of e, whereas its postset $e\bullet = \{\, x \mid (e, x) \in F \,\}$ is defined as the set of output places and transitions of e. We require that each transition has a non-empty preset and a non-empty postset. If a place is in the preset(postset) of t, then it is input(output) to t. For each transition $t \in T$, we require that both $\bullet t$ and $t\bullet$ are nonempty.

Semantics. The global state of a Petri net, called the *marking*, is a function $M : P \rightarrow \mathbb{N}$ that assigns to each place the number of times it is active. Each single activation of a place is visualised by a black dot in the place, called a *token* in Petri net terminology.

From a marking M another marking M' can be reached by firing transitions. A transition t can fire in a marking M if and only if M enables t. Marking M enables transition t if and only if all of t's input places are active, so for all $p \in \bullet t : M(p) \geq 1$. If t fires in M, marking M' is reached, written $M[t\rangle M'$, where for every $p \in P$:

$$M'(p) = \begin{cases} M(p) - 1, & \text{if } p \in \bullet t \setminus t\bullet \\ M(p) + 1, & \text{if } p \in t\bullet \setminus \bullet t \\ M(p), & \text{otherwise.} \end{cases}$$

A marking M' is reachable from M if and only if there is a sequence of transitions $t_1, t_2, .., t_n$ such that $M_1[t_1\rangle M_2[t_2\rangle M_3 .. M_n[t_n\rangle M_{n+1}$ where $M_1 = M$ and $M_{n+1} = M'$.

A *marked Petri net* is a tuple (P, T, F, M), where (P, T, F) is a Petri net and M the (initial) marking of (P, T, F). Given a marked net (P, T, F, M), marking M' is *reachable* if and only if M' is reachable from M. A marked net $PN = (P, T, F, M)$ maps into a transition system $TS(PN) = (S, \rightarrow, init)$ of which the states are markings, the transitions represent firing of Petri net transitions, and the initial state is the initial marking:

$$\begin{aligned} S &= \{M' : P \rightarrow \mathbb{N} \mid M' \text{ is reachable }\} \\ \rightarrow &= \{(M, M') \in S \times S \mid \exists\, t \in T : M[t\rangle M'\} \\ init &= M. \end{aligned}$$

This transition system is usually called reachability graph or marking graph in Petri net terminology.

In a marked net (P, T, F, M), the bound of a place $p \in P$ is the maximum number of tokens assigned to p by any marking reachable from M. A marked Petri net is *safe* or 1-bounded if and only if every place has bound of 1, i.e., no reachable marking puts more than one token in some place. As explained in the introduction, we restrict ourselves to safe marked nets in this paper.

In the main text, we only consider and show marked Petri nets. When we refer to Petri nets in the main text, we actually mean marked Petri nets.

A.3. Statecharts

Syntax. A statechart is hierarchical hypergraph [Har88], consisting of nodes arranged in a tree and directed hyperedges. There are three types of nodes: BASIC, AND, and OR. BASIC nodes are leaves of the tree while AND and OR node are internal. In the literature [Esh09a, HPSS87, PS91], often a function is used that specifies for each node its type. Here we use a different but equivalent formalisation: nodes of the same type are grouped in subsets. This different formalisation simplifies the definition of the declarative translation in Sect. 6. Furthermore, in the literature often a function is used that specifies for each node its set of child nodes in the tree. But we use a binary relation on nodes that specifies the child-parent relation. Again, the formalisation is equivalent and used to simplify the definition of the declarative translation.

Formally, a statechart is a tuple $(N, BN, AN, ON, H, source, target, child, default, r)$, where

- N is a set of nodes, which is partitioned into sets BN, AN, and ON,
- BN is a finite set of BASIC nodes, which are not decomposed into other nodes,
- AN is a finite set of AND nodes, which specify parallel decomposition,
- ON is a finite set of (X)OR nodes, which specify exclusive-or decomposition,
- H is a finite set of hyperedges, $N \cap H = \emptyset$,
- $source : H \rightarrow \mathcal{P}(N)$ is a function defining the non-empty set of source nodes for each hyperedge,
- $target : H \rightarrow \mathcal{P}(N)$ is a function defining the non-empty set of target nodes for each hyperedge,
- $child \subseteq N \times N$ is a predicate that relates a child node to its parent node, so $(n, n') \in child$ means n is child of n'. We require that $child$ arranges the nodes in N in a rooted tree, so every node in N, except the root, has one parent node, and every node is indirectly child of the root. We require $x \in BN$ if and only if $\{y \mid (y, x) \in child\} = \emptyset$, so only non-BASIC nodes have no children.
- $default : ON \rightarrow N$ is a function that identifies for each OR node n one of its children as the default node: $default(n) \in children(n)$. As defined below in the semantics, if a hyperedge h enters n but does not explicitly enter any of its children, then h enters $default(n)$.
- $r \in N$ is the root of the tree induced by $child$. For technical reasons, r is required to be an OR node.

Next, we introduce some auxiliary definitions for the tree induced by $child$. For any node $n \in N$, where $n \neq r$, we denote by $parent(n)$ the unique parent node of n in the tree, so $(n, parent(n)) \in child$. We denote by $children(n)$ the set of children of n, so $\{n' \in N \mid (n', n) \in child\}$. Next, $children^*$ denotes the reflexive-transitive closure of $children$, so $children^*(n) = \bigcup_{i \geq 0} children_i(n)$, where $children_0(n) = \{n\}$ and $children_{i+1}(n) = \bigcup_{n' \in children(n)} children_i(n')$. If $n' \in children^*(n)$, we say that n is *ancestor* of n' and n' is *descendant* of n. Note that each node is ancestor and descendant of itself. Two nodes n, n' are ancestrally related if either n is an ancestor of n' or n' an ancestor of n.

Semantics. Every global state of a statechart, called a configuration, must satisfy several constraints, defined below. First, we introduce some auxiliary definitions. The *lowest common ancestor* of a set $X \subseteq N$ of nodes, written $lca(X)$, is the most nested node $n \in N$ that is an ancestor of every node in X:

$$X \subseteq children^*(n)$$
$$\forall n' \in N : X \subseteq children^*(n') \Rightarrow n \in children^*(n')$$

Given a set X of nodes, $lca^+(X)$ is the most nested OR node that is ancestor of every node in X.

Two nodes $n, n' \in N$ are *orthogonal* if and only if they are not ancestrally related and their lca is an AND node. A set X of nodes is *consistent*, written $consistent(X)$, if and only if for every pair $x, y \in X$, either x and y are ancestrally related or x and y are orthogonal. For each hyperedge h, its source set and target set are required to be consistent, so $consistent(source(h))$ and $consistent(target(h))$.

A *configuration* is a maximal consistent set of nodes: adding a node to a configuration would make it inconsistent. A configuration C satisfies the following properties, for every $x \in C$:

- $x \in ON \Rightarrow |children(x) \cap C| = 1$
- $x \in AN \Rightarrow children(x) \subset C$
- $x \neq r \Rightarrow parent(x) \in C$.

A consistent set X can be turned into a configuration by computing the default completion $D \subseteq N$ of set X. For each non-root node, its parent is included in D. For each AND node, all children are added to D, while for an OR node its default node is added to D, but only if none of its children is in X. Given a consistent set X of nodes, the default completion $dcomp(X)$ is the smallest set D such that:

- $X \subseteq D$
- if $n \in D$ and $n \in AN$ then $children(n) \subseteq D$
- if $n \in D$ and $n \in ON$ and $children(n) \cap X = \emptyset$ then $default(n) \in D$
- if $n \in D$ and $n \neq r$ then $parent(n) \in D$.

From the definition follows immediately that if X is consistent, then $dcomp(X)$ is a configuration.

The initial configuration is defined to be the default completion of root r.

A hyperedge h is *enabled* in configuration C if all its source nodes are in C, so $source(h) \subseteq C$. To define the effects of taking an enabled hyperedge, we need some additional definitions. The *scope* of a hyperedge h is the most nested OR node containing the sources and targets of h:

$$scope(h) = lca^+(source(h) \cup target(h)).$$

Upon taking h, all strict descendants of $scope(h)$ will be left and the target nodes of h are all entered. However, the resulting set $X = C \setminus children^+(scope(h)) \cup target(h)$ of nodes might not be a configuration, for instance if $target(n)$ contains a node n but not its parent below $scope(h)$, or if $target(h)$ contains composite node n but none of n's children. To turn set X into a configuration, the default completion of X is computed. Since $target(h)$ is consistent, the default completion of X is a configuration.

Thus, upon taking hyperedge h, the configuration C changes into C', written $C[h\rangle C'$, where

$$C' = dcomp((C \setminus children^+(scope(h))) \cup target(h)).$$

Given a statechart SC, configuration C is reachable if and only if there is a sequence of hyperedges leading from the initial configuration $dcomp(\{r\})$ to C.

A statechart SC maps into a transition system $TS(SC) = (S, \rightarrow, init)$, where

$$S = \{\ C \subseteq N \ \mid\ C \text{ is a configuration and reachable }\}$$
$$\rightarrow\ = \{(C, C') \in S \times S \ \mid\ \exists\, h \in H : C[h\rangle C'\}$$
$$init = dcomp(\{r\}).$$

B. Proofs

This section contains the proofs of Theorems 6.1 and 6.2.

Proof of Theorem 6.1. Let $SC = (N, BN, AN, ON, H, \mathit{source}, \mathit{target}, \mathit{child}, \mathit{default}, r)$ be the tuple returned by $PNtoSC((P, T, F, M))$. We do not prove that PN and SC are structurally similar, since this follows immediately from the definition of $PNtoSC$, where the identity function $=$ plays the rôle of f.

$\Rightarrow$: We prove that SC is a complete and OR-connected statechart and that PN and SC are behaviourally similar.

SC is a statechart:

We prove that SC is a statechart by showing that the *child* predicate induces a tree, each hyperedge has consistent source and target sets, and that the *default* relation is a function.

We show that the *child* predicate induces a tree, by proving (i) child^+ is acyclic, and (ii) each node not equal to root P has a single parent. Since there is only one root node, this implies that the *child* relation induces a tree on the set of nodes, of which the root is P.

(i) Suppose the *child* relation induces a cycle. From the definition of $PNtoSC$, it follows that if $(n, n') \in \mathit{child}$ then every place $p \in P$ that is descendant of n, so $(p, n) \in \mathit{child}^+$, is also descendant of n', so $(p, n') \in \mathit{child}^+$. Thus, if child^+ contains a cycle, then there must be an AND node $a \in AN$ and OR node $o \in ON$ such that $(o, a) \in \mathit{child}^+$ and $(a, o) \in \mathit{child}^+$. Observe that $(a, o) \in \mathit{child}^+$ implies $\mathit{flatten}(a) \subseteq o$, and $(o, a) \in \mathit{child}^+$ implies that $o \subseteq \mathit{flatten}(a)$. Thus, $\mathit{flatten}(a) = o$, so $a = \{o\}$. By definition of $PNtoSC$, there is a transition with a preset or postset $X \subseteq P$ such that $|X| > 1$ and $a = \mathit{andNode}(X, ON)$. Let $Y = X \setminus (o \cap X)$. Since $Y \subseteq X$, by definition of *cover*, $\mathit{minArea}(Y) \subseteq \mathit{cover}(X)$. Since $a = \{o\}$, $\mathit{minArea}(Y) \not\subset \mathit{cover}(X)$. Therefore, $\mathit{minArea}(Y) = \mathit{cover}(X)$. Since $\mathit{minArea}(X \setminus Y) \subset \mathit{cover}(X)$, we have $\mathit{minArea}(X \setminus Y) \subset \mathit{minArea}(Y)$. But $X \cap (X \setminus Y) = \emptyset$. Thus, the transition of which X is preset or postset does not have consistent areas, which contradicts the assumption. Therefore, child^+ is acyclic.

(ii) Given a node $n \in N$. There are three cases:

- $n \in BN$. By definition of ON, set $\mathit{minArea}(n) \in ON$. Furthermore, *minArea* identifies a unique set, so n has exactly one OR parent.
- $n \in AN$. Since *flatten* is a function and *minArea* identifies a unique set, there is exactly one set $Y = \mathit{minArea}(\mathit{flatten}(n))$. Next, we have to show that $Y \in ON$. Let $t \in T$ be the transition with preset or postset X such that $n = \mathit{andNode}(X, ON)$. By definition of area, $\mathit{minArea}(X) = \mathit{minArea}(t)$. We show that $\mathit{minArea}(X) = \mathit{minArea}(\mathit{flatten}(a))$, which proves the claim.
  $\subseteq$: Take $x \in X$. By definition of *cover*, $\mathit{minArea}(x) \subseteq \mathit{cover}(X)$. Since t has consistent areas, $\mathit{minArea}(x) \subset \mathit{cover}(X)$. Therefore, $x \in \mathit{flatten}(X)$. So $X \subseteq \mathit{flatten}(X)$. Consequently, $\mathit{minArea}(X) \subseteq \mathit{minArea}(\mathit{flatten}(X))$.
  $\supseteq$: Take $x \in \mathit{flatten}(X)$. Then $x \in \mathit{cover}(X)$. By definition of *cover*, there exists $Y \subseteq X$ such that $x \in \mathit{minArea}(Y)$. Since $\mathit{minArea}(Y) \subseteq \mathit{minArea}(X)$, $x \in \mathit{minArea}(X)$. Therefore, $\mathit{flatten}(X) \subseteq \mathit{minArea}(X)$. Next, by definition of *minArea*, $\mathit{minArea}(\mathit{flatten}(X)) \subseteq \mathit{minArea}(X)$.
- $n \in ON$. We show that if $n \neq P$ (so then n is not the root node), n has one parent.
  - n has at least one parent: Since $n \neq P$, there is are two possibilities. (i) There is a transition t such that $n \subset \mathit{minArea}(t)$ and there is no other transition t' such that $n \subset \mathit{minArea}(t') \subset \mathit{minArea}(t)$. So $\mathit{minArea}(t)$ is the minimal area that strictly contains n. (There could also be a place p such that $\mathit{minArea}(p)$ is the minimal area that strictly contains n, but in that case p is the single input or single output place of transition t.) Then t has a preset or postset X such that $X \cap n \neq \emptyset$. By definition of *andNode*, $n \in \mathit{andNode}(X, ON)$, so $\mathit{andNode}(X, ON)$ is the AND parent of n. (ii) There is no element $e \in P \cup T$ such that $n \subset \mathit{minArea}(e)$. Then by definition of AN, an AND node is the parent of n.
  - n has at most one parent: Suppose that n has two AND parents $a_1, a_2 \in AN$ that are different, so $a_1 \neq a_2$. By definition of AN, then there are two sets X_1, X_2 such that $a_1 = \mathit{andNode}(X_1, ON)$ and $a_2 = \mathit{andNode}(X_2, ON)$. Since a_1 and a_2 are different, $\mathit{cover}(X_1) \neq \mathit{cover}(X_2)$ by definition of *andNode*. For a contradiction, suppose $\mathit{cover}(X_1) \subset \mathit{cover}(X_2)$. However, then during construction of a_2 the unique OR parent of a_1 (see previous item) would be a child of a_2. But then, since $n \in a_1 \cap a_2$, node n would have only a_1 as parent, not a_2. So $\mathit{cover}(X_1) \not\subset \mathit{cover}(X_2)$. By similar reasoning, $\mathit{cover}(X_2) \not\subset \mathit{cover}(X_1)$. Due to n, $\mathit{cover}(X_1) \cap \mathit{cover}(X_2) \neq \emptyset$.
    Therefore, $\mathit{cover}(X_1)$ and $\mathit{cover}(X_2)$ are not nestable, which contradicts the assumption. Therefore, n has at most one parent.

Next, we have to show that each hyperedge has consistent source and target sets. Take an arbitrary source or target set X of some hyperedge $h \in H$ and suppose X is not consistent. Then there are two BASIC nodes $x_1, x_2 \in X$ such that $lca(\{x_1, x_2\}) \in ON$. Let $l = lca(\{x_1, x_2\})$. By definition of $PNtoSC$ there is an element $e \in P \cup T$ such that $minArea(e) = l$. Then there are two disjoint directed paths from x_1 to e and from x_2 to e. Since $x_1, x_2 \in minArea(e) \subset minArea(X)$, there are disjoint sets X_1 and X_2 such that $x_1 \in X_1$ and $x_2 \in X_2$ and $X_1 \cup X_2 \subseteq X$ and $minArea(X_1) = minArea(e) = minArea(X_2)$. However, then transition t does not have consistent areas, which contradicts the assumption. So set X is consistent.

Finally, we have to show that the *default* relation is a function. From the definition of *auxdefault* follows immediately that *auxdefault* is a function. We therefore only show that *initdefault* is a function. For a contradiction, suppose that there is an OR node o such that it has two default nodes according to *initdefault*, say x and y. Then by definition of *initdefault*, there is a place p_x that is descendant of x and $M(p_x) = 1$. By similar reasoning, there is a place p_y that is descendant of y and $M(p_y) = 1$. Moreover, $o = minArea(\{p_x, p_y\})$. If $x = p_x$ and $y = p_y$ then $minArea(p_x) = minArea(p_y) = o$, so M is not configurable since p_y can be removed. Otherwise, by definition of $PNtoSC$, for both x and y, there exist non-singleton preset or postset Z_x and Z_y such that $x = andNode(Z_x, ON)$ and $y = andNode(Z_y, ON)$. Let $P_M = \{p \in P \mid M(p) = 1\}$. Since M is configurable, $minArea(Z_x \cap P_M) = o$. But then p_y can be removed from P_M, so P_M is not minimal and M is not configurable. Therefore, *initdefault* and thus *default* is a function.

Behavioural similarity:

We have to prove that $TS(SC)$ and $TS(PN)$ are isomorphic. We will show that bijective function g, as defined below is an isomorphism. Function g relates configurations to markings and is defined as

$$g(C_i) = \{n \mapsto 1 \mid n \in C_i \cap BN\} \cup \{n \mapsto 0 \mid n \in BN \setminus C_i\}.$$

for each configuration C_i.

First, we need to prove that $g(dcomp(\{r\})) = M$. Let $P_M = \{p \in P \mid M(p) = 1\}$. We will show $p \in dcomp(\{r\}) \cap BN \Leftrightarrow p \in P_M$. From the definition of *initdefault* follows immediately that for each $p \in P_M$, $p \in dcomp(\{r\})$. The reverse implication we prove by contradiction. Suppose there is a $p \in dcomp(\{r\}) \cap BN$ such that $p \notin P_M$. Let o be the most nested OR node containing p but not any of the BASIC nodes in P_M. Since $p \in dcomp(\{r\})$ and $P_M \subset dcomp(\{r\})$, the parent of o is an AND node and none of the BASIC descendants of o are in P_M. But then by definition of *minArea*, $minArea(P_M)$ does not contain any of the BASIC nodes contained in o. So $minArea(P_M) \subset P$ and thus M is not configurable, which contradicts the assumption. So $p \in dcomp(\{r\}) \cap BN \Leftrightarrow p \in P_M$.

Next, we need to prove that $(C, C') \in \rightarrow_{SC}$ if and only if $(g(C), g(C') \in \rightarrow_{PN}$, where $\rightarrow_{SC}$ and $\rightarrow_{PN}$ are the transition relations of the transition systems of SC and PN, respectively. This can be proven using $C[h\rangle C' \Leftrightarrow g(C)[h\rangle g(C')$, which follows immediately from the fact that each hyperedge h is complete, so $C[h\rangle C'$ implies $(C \setminus C') \cap BN = source(h)$ and $(C' \setminus C) \cap BN = target(h)$.

$\Leftarrow$: We prove that (P, T, F, M) is statechartable.

Nestable covers:

For a contradiction, suppose two covers are not nestable, so there are non-singleton sets $X, Y \subseteq P$ such that X and Y are preset or postset of some transitions in T, $cover(X) \cap cover(Y) \neq \emptyset$ and $cover(X) \not\subseteq cover(Y)$ and $cover(Y) \not\subseteq cover(X)$. Since $cover(X)$ and $cover(Y)$ overlap, there is a place $p \in cover(X) \cap cover(Y)$. By construction, the AND nodes created for $cover(X)$ and $cover(Y)$ are different, but both are ancestor of p. Since $cover(X) \not\subseteq cover(Y)$ and $cover(Y) \not\subseteq cover(X)$, the AND nodes created for $cover(X)$ and $cover(Y)$ are not ancestrally related. So BASIC node p has two ancestors which are not ancestrally related themselves. This contradicts the tree property. So PN has nestable covers.

Consistent areas:

Take a set $S \subseteq BN$ such that there is a hyperedge h with $source(h) = S$ or $target(h) = S$. Let a be the least common ancestor of S. Since S is consistent, a is an AND node.

For any subset $S' \subset S$, define $ON_{S'}$ as the set of OR nodes that contain any node in S', are descendant of a, and are maximal, so if $o_1, o_2 \in ON_{S'}$ then $o_1 \not\subset o_2$ and $o_2 \not\subset o_1$. Since *child* induces a tree, this means that for any pair of nodes $o_1, o_2 \in ON_{S'}$, $o_1 \cap o_2 = \emptyset$. By definition of $PNtoSC$ and $ON_{S'}$, for every $o \in ON_{S'}$, $minArea(o \cap S') = o$. Since all OR nodes in $ON_{S'}$ are pairwise disjoint, $minArea(S') = \bigcup_{o \in ON_{S'}} minArea(o \cap S') = flatten(ON_{S'})$, where auxiliary function *flatten* : $\mathcal{P}(N) \rightarrow N$ takes as input a set of subsets of nodes and returns the union of the subsets:

$$flatten(Xs) = \bigcup_{X \in Xs} X.$$

Now take two disjoint subsets $X, Y \subseteq S$. Since X and Y are disjoint, also ON_X and ON_Y are disjoint. Consequently, $\mathit{flatten}(ON_X) = \mathit{minArea}(X)$ and $\mathit{flatten}(ON_Y) = \mathit{minArea}(Y)$ are disjoint too, for any transition h with preset or postset S. Thus (P, T, F, M) has consistent areas.

Configurable initial marking:

Let $P_M = \{p \in P \mid M(p) = 1\}$. Since $TS(SC)$ and $TS(PN)$ are isomorphic and $PNtoSC$ is structure-preserving, $P_M = dcomp(\{root\}) \cap BN$. Since $dcomp(\{r\})$ is a configuration, set P_M of BASIC nodes is maximal consistent: $consistent(P_M)$ and for each BASIC node $n \in BN \setminus P_M$, $\neg\, consistent(P_M \cup \{n\})$.

We first prove $minArea(P_M) = P$. For a contradiction, suppose $minArea(P_M) \subset P$. Let $o \in ON$ be the smallest area such that $minArea(P_M) \subset o \subseteq P$. Since $minArea(P_M) \subset o$, either (i) there is a transition t such that $\bullet t \cup t\bullet \subseteq o$ yet $\bullet t \cup t\bullet \not\subseteq minArea(P_M)$ or (ii) $o = P$ and there is no transition t' such that $o \subset minArea(t')$. For (i), then there is a place $p \in \bullet t \cup t\bullet$ such that $p \notin minArea(P_M)$. However, by definition of $PNtoSC$ BASIC node p is consistent with all BASIC nodes in P_M. For (ii), then there is a place p that is not connected to any of the places in P_M. The statechart then contains a top-level AND node of which one OR child contains BASIC node p and none of the nodes in P_M. Therefore, in the statechart p is consistent with all BASIC node in P_M. From both (i) and (ii) follows that for SC, P_M is not maximal consistent for BASIC nodes (since p can be added), which is a contradiction. Therefore, $minArea(P_M) = P$.

Next, we show that P_M is minimal. For a contradiction, take a subset $X \subset P_M$ and suppose $minArea(X) = P$. Take a place $p \in P_M \setminus X$. Since P_M is (maximal) consistent, BASIC node p is consistent with each BASIC node in P_M. Let a be the most nested AND ancestor of BASIC node p, so each AND node that contains p also contains a. Each BASIC node $n \in X$ is either contained in a or orthogonal to a. If each BASIC node $n \in X$ is orthogonal to a, then all BASIC nodes contained in a are not in $minArea(X)$ by definition of $minArea$, which contradicts that $minArea(X) = P$. Therefore, there is a BASIC node $n \in X$ that is contained in a and n is consistent with p. Since $minArea(X) = P$ and $p \notin X$, by definition of $minArea$ the OR child o of a that contains p has to contain at least one other place y of X. Since a is the most nested AND node containing p, the least common ancestor of p and y is o. But then P_M is not consistent (since it contains both p and y) and there is a contradiction. So P_M is minimal.

Since $minArea(P_M) = P$ and P_M is minimal, initial marking M is configurable. □

Proof of Theorem 6.2. Let (P, T, F, M) be a statechartable net. From Theorem 6.1 follows that $SC = PNtoSC((P, T, F, M))$ is a statechart.

To show that SC is a complete and OR-connected statechart, it suffices to show that (i) each hyperedge is complete and (ii) each OR node is connected. (i) follows directly from the definition of set AN by $PNtoSC$. (ii) follows from the fact that statechartable nets have nestable covers: if an OR node is unconnected, then the unconnected parts are contained in unnestable covers. □

References

[AB08] Ameedeen MA, Bordbar B (2008) A model driven approach to represent sequence diagrams as free choice Petri nets. In: Proc. EDOC 2008. IEEE Computer Society, pp 213–221

[BdC92] Bernardinello L, de Cindio F (1992) A survey of basic net models and modular net classes. In: Rozenberg G (ed) Advances in Petri nets 1992. Lecture notes in computer science, vol 609. Springer, Berlin, pp 304–351

[DH94] Drusinsky D, Harel D (1994) On the power of bounded concurrency I: finite automata. J ACM 41(3):517–539

[DJ01] Desel J, Juhás G (2001) What is a Petri net? Informal answers for the informed reader. In: Ehrig H, Juhás G, Padberg J, and Rozenberg G (eds) Unifying Petri nets. Lecture notes in computer science, vol 2128. Springer, Berlin, pp 1–27

[Esh05] Eshuis R (2005) On nets with structured concurrency. Beta Working Paper Series, WP 155, Eindhoven University of Technology

[Esh09a] Eshuis R (2009) Reconciling statechart semantics. Sci Comput Program 74(3):65–99

[Esh09b] Eshuis R (2009) Translating safe Petri nets to statecharts in a structure-preserving way. In: Cavalcanti A, Dams D (eds) FM 2009. Lecture notes in computer science, vol 5850. Springer, Berlin, pp 239–255

[GK07] Grumberg O, Katz S (2007) Veritech: a framework for translating among model description notations. STTT 9(2):119–132

[Gra97] Grahlmann B (1997) The PEP tool. In Grumberg O (ed) Proc. CAV '97. Lecture notes in computer science, vol 1254. Springer, Berlin, pp 440–443

[Ham05] Hammal Y (2005) A formal semantics of UML statecharts by means of timed Petri nets. In: Wang F (ed) Proc. FORTE 2005. Lecture notes in computer science, vol 3731. Springer, Berlin, pp 38–52

[Har87] Harel D (1987) Statecharts: a visual formalism for complex systems. Sci Comput Program 8(3):231–274

[Har88] Harel D (1988) On visual formalisms. Commun ACM 31(5):514–530

[HK92] Harel D, Kahana C-A (1992) On statecharts with overlapping. ACM Trans Softw Eng Methodol 1(4):399–421

[HK02] Harel D, Kugler H (2002) Synthesizing state-based object systems from LSC specifications. Int J Found Comput Sci 13(1):5–51

[HMP$^+$02] Huszerl G, Majzik I, Pataricza A, Kosmidis K, Dal Cin M (2002) Quantitative analysis of UML statechart models of dependable systems. Comput J 45(3):260–277

[HN96] Harel D, Naamad A (1996) The STATEMATE semantics of statecharts. ACM Trans Softw Eng Methodol 5(4):293–333

[HPSS87] Harel D, Pnueli A, Schmidt JP, Sherman S (1987) On the formal semantics of statecharts. In: Proceedings of the second IEEE symposium on logic in computation. IEEE, pp 54–64

[Jen92] Jensen K (1992) Coloured Petri nets. Basic concepts, analysis methods and practical use. In: EATCS monographs on theoretical computer Science. Springer, Berlin

[KCK+97] Kishinevsky M, Cortadella J, Kondratyev A, Lavagno L, Taubin A, Yakovlev A (1997) Coupling asynchrony and interrupts: place chart nets. In: Azéma P, Balbo G (eds) Proc ICATPN 1997. Lecture notes in computer science, vol 1248. Springer, Berlin, pp 328–347

[Klu03] Kluge O (2003) Modelling a railway crossing with message sequence charts and Petri nets. In: Ehrig H, Reisig W, Rozenberg G, Weber H (eds) Petri Net technology for communication-based systems. Lecture notes in computer science, vol 2472. Springer, Berlin, pp 197–218

[Mat] The Mathworks. Stateflow user's guide. http://www.mathworks.com

[Mur89] Murata T (1989) Petri nets: properties, analysis, and applications. In: Proc IEEE 77(4):541–580

[Pet62] Petri CA (1962) Kommunikation mit Automaten. PhD thesis, Institut für instrumentelle Mathematik, Bonn

[PS91] Pnueli A, Shalev M (1991) What is in a step: on the semantics of statecharts. In: Ito T, Meyer AR (eds) Theoretical aspects of computer software. Lecture notes in computer science, vol 526. Springer, Berlin, pp 244–265

[Rei85] Reisig W (1985) Petri Nets: an introduction. In: EATCS monographs on theoretical computer science, vol 4. Springer, Berlin

[RK97] Rausch M, Krogh B (1997) Transformations between different model forms in discrete event systems. In: Proc IEEE SMC 1997, vol 3, pp 2841–2846

[RR98] Reisig W, Rozenberg G (eds) (1998) Lectures on Petri nets I: advances in Petri nets. In: Lecture notes in computer science, vol 1492. Springer, Berlin

[SNK99] Schnabel M, Nenninger G, Krebs V (1999) Konvertierung sicherer Petri-netze in statecharts (in German). Automatisierungstechnik 47(12):571–580

[SSH01] Saldhana JA, Shatz SM, Hu Z (2001) Formalization of object behavior and interactions from UML models. Int J Softw Eng Knowl Eng 11(6):643–673

[UML03a] UML Revision Taskforce. OMG UML specification v. 1.5. Object Management Group, 2003. OMG Document Number formal/2003-03-01

[UML03b] UML Revision Taskforce. UML 2.0 superstructure specification. Object Management Group, 2003. OMG Document Number ptc/03-07-06

[VE10] Van Gorp P, Eshuis R (2010) Transforming process models: executable rewrite rules versus a formalized java program. In: Petriu DC, Rouquette N, Haugen Ø (eds) Proc MoDELS 2010. Lecture notes in computer science, vol 6395. Springer, Berlin, pp 258–272

[WS00] Whittle J, Schumann J (2000) Generating statechart designs from scenarios. In: Proc ICSE, pp 314–323

Received 2 February 2010
Revised 24 June 2011
Accepted 7 August 2011 by Jim Woodcock
Published online 24 September 2011