A Probabilistic Extension of UML Statecharts
Specification and Verification

David N. Jansen[1], Holger Hermanns[2], and Joost-Pieter Katoen[2]

[1] Universiteit Twente, Information Systems Group.
[2] Universiteit Twente, Formal Methods and Tools Group.
Postbus 217, 7500 AE Enschede, The Netherlands.
{dnjansen,hermanns,katoen}@cs.utwente.nl

Abstract. This paper is the extended technical report that corresponds to a published paper [14].
This paper introduces means to specify system randomness within UML statecharts, and to verify probabilistic temporal properties over such enhanced statecharts which we call probabilistic UML statecharts. To achieve this, we develop a general recipe to extend a statechart semantics with discrete probability distributions, resulting in Markov decision processes as semantic models. We apply this recipe to the requirements-level UML semantics of [8]. Properties of interest for probabilistic statecharts are expressed in PCTL, a probabilistic variant of CTL for processes that exhibit both non-determinism and probabilities. Verification is performed using the model checker Prism. A model checking example shows the feasibility of the suggested approach.

Keywords: Markov decision processes, model checking, probabilities, semantics, UML statecharts.

1 Introduction

The Unified Modelling Language (UML) is more and more pervading system design and engineering. Accordingly, it is not far fetched to predict that the coming years shall see substantial efforts to extend the UML and the accompanying design methodology towards soft real-time, fault-tolerance, quality of service and the like. First work in this direction has been undertaken, e. g., in [5, 9, 15, 17].

One of the principal modelling paradigms needed to express such aspects is the concept of *probability,* allowing one to quantitatively describe the randomness the system is exposed to, the randomness the system itself exhibits, or both. For instance, probability is a useful means to describe varying workload, to quantify uncertainty in the system timing, as well as to properly model randomised distributed algorithms. Furthermore, probability is also an abstraction means, e. g., it allows one to hide data dependencies by just representing the likelihood of particular branches to be taken.

There are two facets of the probability concept that are worth to be distinguished. On the one hand, each reactive system is exposed to external stimuli that exhibit some kind of randomness. We call this *environmental randomness.*

On the other hand, the system behaviour itself may ask for a probabilistic description, either because the system embodies a randomised algorithm, or because probability is used for abstraction. We call this *system randomness.*

This paper addresses system randomness. It introduces probabilistic UML-statecharts as a means to support the design of probabilistic systems. More concretely, the paper extends statecharts by probabilistic elements: a transition is allowed to lead to one of several states depending on a probability distribution; each probability distribution is guarded by a trigger, inspired by [21]. The interference of probabilities, priorities and nondeterminism raises some subtle semantic issues. We attack these issues in a way that allows one still to employ an arbitrary priority scheme to resolve or reduce nondeterminism. The semantics is formally defined as a mapping on (strictly) alternating probabilistic transition systems [10], a subset of Markov decision processes (MDP) [20]. To allow verification of probabilistic temporal properties over probabilistic UML-statecharts, properties are expressed in the probabilistic branching-time temporal logic PCTL [1], the prime logic for property specification and verification of models that exhibit both probabilities and nondeterminism. These properties can be checked using the model checker Prism [16].

Care is taken to achieve a conservative extension of standard UML statecharts. Among the various published semantics for statecharts, we take as a representative the requirements-level semantics of Eshuis and Wieringa [8], which is based on the semantics by Damm et al. [6] which in turn formalises the Statemate semantics of statecharts [13]. A requirements-level model focuses on the design; an implementation-level model describes the implementation of a design. Requirements-level semantics mostly use the perfect technology assumption, which abstracts from limitations (in speed and memory) imposed by an implementation [19]. The chosen semantics combines the Statemate semantics with communication and classification. We have chosen it because it is simple and close to the most used semantics for UML. A detailed justification of this semantics and comparisons to Statemate semantics as well as implementation-level semantics can be found in [8]. The setup of our probabilistic extension, however, is independent of the UML basis we take. This means that other formal statechart semantics can equally well be enhanced with a similar probabilistic extension, as long as certain principal conditions are respected. We give an account of these conditions in Sect. 7.

We omit some minor features of the semantics, that could be added easily but would clutter up the exposition. These features include actions on initial transitions and entry and exit actions. We also leave out real-time aspects. To facilitate model-checking of the underlying model, we confine ourselves to bounded integer variables and employ a closed world assumption.

In summary, this paper makes the following contributions. (i) It introduces a generic recipe to conservatively extend a statechart dialect with probabilistic features. (ii) It details this recipe for the requirement-level semantics of [8], and identifies subtle interferences between the imposed priority scheme, and the order of resolving nondeterminism and probabilistic choice. (iii) It proposes to use the

probabilistic logic PCTL to specify properties over statecharts and shows how model checking of probabilistic statecharts can be performed effectively.

Organisation of the paper. Sect. 2 and 3 introduce syntax and semantics of probabilistic statecharts. In Sect. 4, we show that this semantics conservatively extends the non-probabilistic statechart semantics. Sect. 5 presents a logic for P-statecharts and Sect. 6 demonstrates model checking with a larger example. Sect. 7 discusses our approach in the broader context of statechart semantics and of probabilistic models.

2 Probabilistic UML Statecharts

This section introduces probabilistic UML-statecharts (P-statecharts, for short), together with some drawing conventions. We first fix some notations. We assume familiarity with basic measure and probability theory [22].

Notation and terminology. The powerset of a set A is denoted by $\mathbb{P}(A)$.

Given a set Ω, a system $\mathcal{A}$ of subsets of Ω is called a *σ-algebra* if: (i) $\Omega \in \mathcal{A}$, (ii) if $A_n \in \mathcal{A}$ (for $n \in \mathbb{N}$), then $\bigcup_{n=1}^{\infty} A_n \in \mathcal{A}$, and (iii) $A \in \mathcal{A}$ implies $\Omega \setminus A \in \mathcal{A}$.

A *probability space* is a triple $(\Omega, \mathcal{A}, P)$ where Ω is the set of possible outcomes of a probabilistic experiment, $\mathcal{A} \subseteq \mathbb{P}(\Omega)$ is a σ-algebra of measurable sets and $P : \mathcal{A} \to [0,1]$ is a probability measure. A set $A \in \mathcal{A}$ has probability $P(A)$. For discrete probability spaces, we sometimes write (Ω, P) instead of $(\Omega, \mathbb{P}(\Omega), P)$.

A *probability weight* $\mu : \Omega \to [0,1]$ is a function that can serve to define a discrete probability space: if $\sum_{\omega \in \Omega} \mu(\omega) = 1$, then (Ω, P) with $P(A) = \sum_{\omega \in A} \mu(\omega)$ is a discrete probability space.

Collection of statecharts. A system consists of a finite collection of communicating statecharts. In the following, we assume a given finite collection of P-statecharts, denoted by $\{PSC_1, \ldots, PSC_n\}$.

Syntax. A single P-statechart PSC_i consists of the following elements:

- A finite set $Nodes_i$ of nodes with a tree structure, described by a function $children_i : Nodes_i \to \mathbb{P}(Nodes_i)$. (If $x' \in children_i(x)$, then x is the parent of x'. Of course, $children_i$ has to fulfil several constraints to make it describe a tree structure. For simplicity, these are omitted here.) Descendants are children or children of children, etc. The root is denoted $root_i$ (so, $Nodes_i \setminus children_i(Nodes_i) = \{root_i\}$ holds).
  The function $type_i : Nodes_i \to \{\text{BASIC}, \text{AND}, \text{OR}\}$ assigns to every node its type. Nodes that are leaves of the tree have type BASIC; children of AND nodes have type OR; $type_i(root_i) = \text{OR}$; other nodes have type OR or AND.
  The partial function $default_i : Nodes_i \multimap Nodes_i$ identifies for each OR node one of its children as the default (or initial) node.

- A finite set *Events* of events. (Note that the set of events is identical for all P-statecharts in the collection.) We will use the symbol $\perp$ to denote "no event required"; $\perp \notin \mathit{Events}$.
- A finite set Vars_i of variables together with an initial valuation $V_{0,i} : \mathit{Vars}_i \to \mathbb{Z}$, which assigns initial values to the variables. (We only allow bounded integer variables.)
- A set Guards_i of guard expressions. Guard expressions are boolean combinations of the atoms $j.\mathit{isin}(x)$, for $j \in \{1, \ldots, n\}$ and $x \in \mathit{Nodes}_j$ (with the intuitive meaning "the P-statechart PSC_j is in node x"), and comparisons like $\mathit{expr} \leqslant \mathit{expr}$ and $\mathit{expr} > \mathit{expr}$, for arithmetic expressions made up from the variables and integer constants.
- A set $\mathit{Actions}_i$ of actions. Actions are $\mathsf{v} := \mathit{expr}$, which denotes an assignment to $\mathsf{v} \in \mathit{Vars}_i$, and **send** $j.\mathsf{e}$ (for $j \in \{1, \ldots, n\}$ and $\mathsf{e} \in \mathit{Events}$) with the intuitive meaning "send event e to the P-statechart PSC_j".[3]
- A finite set PEdges_i of P-edges. A P-edge is a tuple (X, e, g, P) where $X \subseteq \mathit{Nodes}_i$ is a non-empty set of source state nodes, $e \in \mathit{Events} \cup \{\perp\}$, $g \in \mathit{Guards}_i$ is a guard, and P is a probability measure in the discrete probability space $(\mathbb{P}(\mathit{Actions}_i) \times (\mathbb{P}(\mathit{Nodes}_i) \setminus \{\varnothing\}), P)$. We assume that there is a bijective index function $\iota : \{1, \ldots, |\mathit{PEdges}_i|\} \to \mathit{PEdges}_i$ to simplify the identification of P-edges.

Note that Guards_i and $\mathit{Actions}_i$ are defined in terms of the other components. We will therefore denote a P-statechart simply by $\mathit{PSC}_i = (\mathit{Nodes}_i, \mathit{Events}, \mathit{Vars}_i, \mathit{PEdges}_i)$.

The above definition differs from the usual statecharts syntaxes (e. g., [8]) in the way edges are refined into P-edges. A P-edge (X, e, g, P) can be considered as a hyperedge with source node set X, and possibly multiple targets (A, Y), each target having a certain probability $P(A, Y)$. Note that a target is an action set A together with a non-empty set Y of successor nodes. Once the P-edge is triggered by event e and guard g holds in node X, a target (A, Y) is selected with probability $P(A, Y)$.

Drawing a P-Statechart. We adopt the following drawing conventions. The root node is not drawn. Nodes that are not children of an AND-node are drawn as rectangles with rounded corners. A parent node encloses its children. Children of an AND-node partition the node by dashed lines. Each OR-node encloses a black dot and indicates its default node by an arrow directed from the dot to the default node. A trivial P-edge (where probability 1 is assigned to a unique action set/node set) with event e, guard g and action set A is denoted as an arrow $\xrightarrow{\mathsf{e}[\mathsf{g}]/A}$. A P-edge possessing a non-trivial probability space consists of two parts: first an arrow with event and guard $\xrightarrow{\mathsf{e}[\mathsf{g}]}$ that points to a symbol Ⓟ (a so-called P-pseudonode), then several arrows emanating from the P-pseudonode, each with a probability and an action set $\xrightarrow{p/A}$. This notation is inspired by C-pseudonodes Ⓒ, used for case selection purposes e. g., in [12]. If the event on a

[3] We will abbreviate **send** $j.\mathsf{e}$ sometimes by $j.\mathsf{e}$.

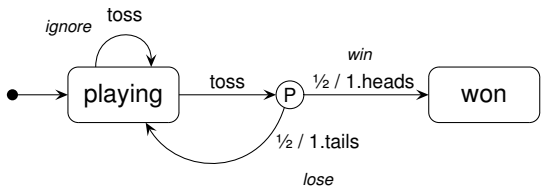

Fig. 1. Example P-statechart. The labels printed in italics (*ignore* etc.) are not part of the diagram, but are included to facilitate referencing the edges near the labels.

P-edge is $\bot$, we may omit it: $\xrightarrow{[g]/A}$. Further, if the guard is *true*, we may omit it: $\xrightarrow{e/A}$. Similarly an empty set of actions may be omitted: $\xrightarrow{e[g]}$.

Example 1. Figure 1 depicts a P-statechart which shows the behaviour when playing with an *unreliable, but fair coin:* the event "toss" may or may not be ignored. If the system reacts, it generates "heads" or "tails", each with 50 % chance. If the output is heads, the system stops playing. It is unspecified how (un)reliable the system is.

3 P-Statechart Semantics

This section discusses the semantics of P-Statechart, which is an adaptation of the nonprobabilistic semantics in [8]. The semantics is defined in two phases. First, it is defined what will be a step. This encompasses the resolution of nondeterminism, probabilistic choice and priorities within a single P-statechart. Subsequently, these steps are used as the buiding blocks in a mapping of a collection of P-statecharts onto a Markov decision process.

Closed-world assumption. Opposed to [8] we assume a closed system model in the sense that we do *not* consider the system to be subject to inputs from the environment. This "closed-world assumption" simplifies model checking. The "open-world" semantics of [8] provides for input from the environment of the statecharts, and therefore, their state transitions consist of several phases, cluttering up the semantics. However, if one wants to consider specific environmental inputs, one can easily add another component that generates the desired events. See Example 3 for such a component.

Intuitive semantics for a single P-statechart. The intuitive behaviour of a P-statechart can be described as follows. The statechart is always in some state (which consists of one or several nodes). A P-edge is taken if the P-statechart is in the source node(s), the event of the edge happens and its guard holds. Then, the system chooses one of the possible results (probabilistically and nondeterministically); it leaves the source nodes, executes the chosen action and enters the chosen target nodes of the P-edge. More than one edge may be taken simultaneously, even within a single statechart.

3.1 Step Construction

This section describes how a step is constructed for a single P-statechart PSC_i. A step is, basically, the set of edges taken simultaneously in one transition.

Configurations and states. A *configuration* C_i of P-statechart PSC_i is a set of nodes that fulfils the conditions:

- $root_i \in C_i$.
- If an OR-node is in C_i, then exactly one of its children is in C_i.
- If an AND-node is in C_i, then all its children are in C_i.

The set of all configurations of PSC_i is denoted $Conf_i$. A *state* of PSC_i is a triple (C_i, I_i, V_i) where C_i is a configuration, $I_i \subseteq Events$ is a set of events (to which the statechart still has to react), and $V_i : Vars_i \to \mathbb{Z}$ is a valuation of the variables. The set of all valuations of PSC_i is denoted Val_i. The validity of guard g in a state depends on the configurations $C_1, \ldots, C_n$ and the valuations $V_1, \ldots, V_n$. We write $(C_{1\ldots n}, V_{1\ldots n}) \vDash g$ iff g holds in the state of the collection of P-statecharts.

Edges. An edge is a triple (j, A, Y), where j identifies a P-edge, $A \subseteq Actions_i$ is a set of actions and $Y \subseteq Nodes_i$ is a set of target nodes. The set $Edges_i$ is defined as: $\{(j, A, Y) \mid \exists X, e, g, P : \iota(j) = (X, e, g, P) \in PEdges_i \wedge P(\{(A, Y)\}) > 0\}$.

Scope. The *scope* of an edge (j, A, Y) is the smallest (in the parent–child-hierarchy) OR-node that contains both the source nodes $\iota(j).X$ and the target nodes Y. Since this node only depends on source and target nodes, we refer to it as $scope(\iota(j).X, Y)$. (The scope is the smallest node that is not affected when the edge is executed.)

Steps. A step is a set of edges that are taken together as a reaction to events. The edges in a step for P-statechart PSC_i depend on its current state (C_i, I_i, V_i) (and, for the guards, on the configurations and valuations of the other statecharts in the collection). A step has to obey several constraints, which are in close correspondance to [8]:

Enabledness. All edges in the step must be enabled. A P-edge (X, e, g, P) is enabled if the current configuration C_i contains its source state nodes X, the event e is in the current input set I_i and the guard g holds: $(C_{1\ldots n}, V_{1\ldots n}) \vDash g$. An edge is enabled if its corresponding P-edge is enabled.[4]

Consistency. All edges in the step must be pairwise consistent. This means that they are either identical or that their scopes are different children of some AND-node or their descendants (in the latter case, the scopes are called orthogonal in [8]). If two edges are not consistent, they are called inconsistent.

[4] It may happen that no P-edge, and consequently no edge, is enabled. Because of the closed world assumption, this is a deadlock.

Priority. We assume a given priority scheme (a partial order on the edges) that resolves some of the inconsistencies: If an enabled edge e is not in the step, then there must be an edge in the step that is inconsistent with e and does not have lower priority than e.

Maximality. A step must be maximal. This means that adding any edge leads to a violation of the above conditions.[5]

We now give an algorithm to construct a step of a single statechart which – by construction – satisfies the conditions above. The algorithm employs a specific order with respect to the resolution of nondeterminism and probabilities. Assume that the current state is (C_i, I_i, V_i).

Algorithm 1. Step Construction Algorithm.

1. Calculate the set of enabled P-edges: for $j \in \{1, \ldots, |PEdges_i|\}$,

$$j \in EnP(C_i, I_i, V_i) \quad \text{iff} \quad \iota(j).X \subseteq C_i \wedge (\iota(j).e \in I_i \cup \{\bot\}) \wedge (C_{1\ldots n}, V_{1\ldots n}) \vDash \iota(j).g$$

2. Draw samples from the probability spaces of the enabled P-edges, reducing the set $EnP(C_i, I_i, V_i)$ to a set $En(C_i, I_i, V_i)$ of enabled edges.
3. Calculate $Steps(En(C_i, I_i, V_i))$, where $Steps(E)$ (for $E \subseteq Edges_i$) contains all maximal, prioritized, consistent sets of edges $\subseteq E$.
4. Choose nondeterministically an element of $Steps(En(C_i, I_i, V_i))$.

Task 2 can be formalised as follows: For every state (C_i, I_i, V_i), we define a discrete probability space $\mathcal{PR}_{(C_i, I_i, V_i)}$ over $\mathbb{P}(Edges_i)$. Its probability measure is the lift of the following probability weight μ to sets of sets: for any selection of A_j and Y_j (for $j \in EnP(C_i, I_i, V_i)$),

$$\mu(\{(j, A_j, Y_j) \mid j \in EnP(C_i, I_i, V_i)\}) = \prod_{j \in EnP(C_i, I_i, V_i)} (\iota(j).P)(\{(A_j, Y_j)\})$$

and $\mu(E) = 0$ otherwise. Note that if $EnP(C_i, I_i, V_i) = \varnothing$, then μ is the trivial probability weight such that $\mu(\varnothing) = 1$.

Tasks 3 and 4 can be achieved by applying the original algorithm for *nextstep* (see [8]) to the calculated set $En(C_i, I_i, V_i)$. It is a nondeterministic algorithm that calculates a maximal, prioritized, consistent step, given a set of enabled edges. As a consequence, the above algorithm (consisting of Tasks 1–4) leads to a step that is enabled, consistent, prioritized and maximal.

3.2 Order of Tasks in Algorithm 1

It is worth to highlight that (after calculating the enabled possibilities in Task 1), we first choose probabilistically (in Task 2) according to the probabilities given

[5] If there is no unique maximum, the system is nondeterministic.

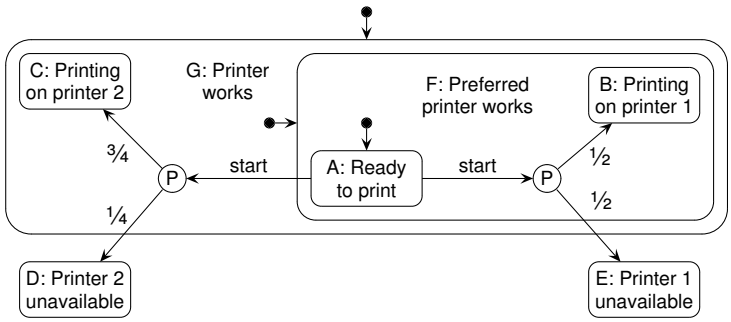

Fig. 2. Example of priority depending on target state

by the P-edges. Only after that, in Tasks 3 and 4, we resolve the nondeterminism between the remaining possibilities. This order – first resolve probabilism, then nondeterminism – is essential, as shown by the following two examples, and can only be reversed by sacrificing the expressivity of P-statecharts, e. g., by disallowing arrows to cross node boundaries from a P-pseudonode to a target node.

Priority depends on probabilistic choices. In the mostly used priority schemes, priority depends on the scope. For example, in the UML priority scheme, smaller (in the parent–child-hierarchy) scopes have higher priority, according to [12]. The P-statechart in Fig. 2 describes a fragment of a system with two printers, of which the preferred printer (printer 1) is available only with probability $\frac{1}{2}$, and the other (printer 2) is available in $\frac{3}{4}$ of the cases. The probabilities are independent. If a print request is started, the system directs it to the best available printer. The edge leading from A: Ready to print to C: Printing on printer 2 (denoted $\mathsf{A} \to \mathsf{C}$) with scope G: Printer works has higher priority than $\mathsf{A} \to \mathsf{D}$ and $\mathsf{A} \to \mathsf{E}$ (with scopes *root*), but $\mathsf{A} \to \mathsf{C}$ has lower priority than the edge $\mathsf{A} \to \mathsf{B}$ (with scope F: Preferred printer works), because F: Preferred printer works is a descendant of G: Printer works. So, if in configuration $\{\mathsf{A}, \mathsf{F}, \mathsf{G}, \mathit{root}\}$ event start happens, the step produced by the above algorithm is:

- $\{\mathsf{A} \to \mathsf{B}\}$ with probability $\mu(\{\mathsf{A} \to \mathsf{B}, \mathsf{A} \to \mathsf{C}\}) + \mu(\{\mathsf{A} \to \mathsf{B}, \mathsf{A} \to \mathsf{D}\}) = \frac{1}{2} \cdot \frac{3}{4} + \frac{1}{2} \cdot \frac{1}{4} = \frac{1}{2}$, as edge $\mathsf{A} \to \mathsf{B}$ has priority over all other edges.
- $\{\mathsf{A} \to \mathsf{C}\}$ with probability $\mu(\{\mathsf{A} \to \mathsf{E}, \mathsf{A} \to \mathsf{C}\}) = \frac{1}{2} \cdot \frac{3}{4} = \frac{3}{8}$, as $\mathsf{A} \to \mathsf{C}$ has priority over $\mathsf{A} \to \mathsf{E}$.
- Either $\{\mathsf{A} \to \mathsf{D}\}$ or $\{\mathsf{A} \to \mathsf{E}\}$ with probability $\mu(\{\mathsf{A} \to \mathsf{E}, \mathsf{A} \to \mathsf{D}\}) = \frac{1}{2} \cdot \frac{1}{4} = \frac{1}{8}$. The choice between these two steps is purely nondeterministic.

As the edges that belong to one P-edge have different scopes, it is impossible to resolve the priorities prior to resolving the probabilistic choice. Although this is exemplified using the priority scheme of [12], a similar P-statechart can be drawn for the Statemate priority scheme (which also depends on the scope).

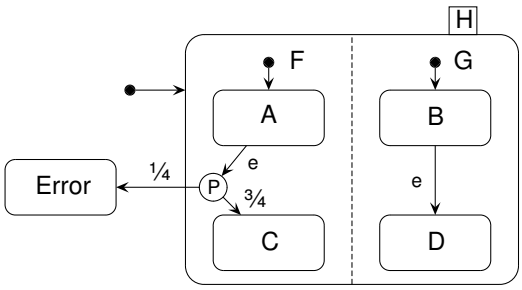

Fig. 3. Example of consistency depending on target state

The priority scheme of [23] does not depend on the scope, but only on the source nodes. For such a priority scheme, the above phenomenon is not a problem; but the next example is independent from priorities.

Consistency depends on probabilistic choices. The P-statechart in Fig. 3 shows a system which reacts to an event e in two independent components, of which one causes an error with probability $\frac{1}{4}$. The edge $\mathsf{A} \to \mathsf{Error}$ is inconsistent with $\mathsf{B} \to \mathsf{D}$, as the scopes *root* and G are not (descendants of) different children of an AND-node (orthogonal). So, if in configuration $\{\mathsf{A}, \mathsf{B}, \mathsf{F}, \mathsf{G}, \mathsf{H}, \mathit{root}\}$ event e happens, the step produced by the above algorithm is:

- $\{\mathsf{A} \to \mathsf{C}, \mathsf{B} \to \mathsf{D}\}$ with probability $\mu(\{\mathsf{A} \to \mathsf{C}\}) = \frac{3}{4}$.
- Either $\{\mathsf{B} \to \mathsf{D}\}$ or $\{\mathsf{A} \to \mathsf{Error}\}$ with probability $\mu(\{\mathsf{A} \to \mathsf{Error}\}) = \frac{1}{4}$; most priority schemes only allow one of the two cases.

Thus, the probability of taking edge $\mathsf{B} \to \mathsf{D}$ as a reaction to event e depends on the resolution of the probabilistic choice in the parallel node F. It is impossible to resolve the nondeterminism first, as there may or may not be inconsistent edges.

In summary, the influence of the target state in the construction of a step, as present in both the consistency definition and the priority scheme, forces us to resolve probabilism *prior* to establishing consistency and priority.

Changing the order anyway. If we restrict P-statecharts in two points, we may change the order of resolution:

1. Arrows from P-pseudonodes to states are not allowed to cross node boundaries (i. e., for every P-edge, there is a node x such that all target nodes entered with positive probability are children of x).
2. The priority of an edge depends on the source nodes or on the scope of an edge.

The scopes of two edges (j, A_1, Y_1) and (j, A_2, Y_2) of one P-edge whose target nodes have the same parent (i. e., $\exists x \in \mathit{Nodes}_i : Y_1 \subseteq \mathit{children}_i(x) \wedge Y_2 \subseteq \mathit{children}_i(x)$) are equal. Consistency and most priority schemes only depend on

the scope. Therefore, with these restrictions, we could change Algorithm 1 such that it first resolves nondeterminism and then probabilism.

3.3 Step Execution

After having settled how steps are selected within a single P-statechart, we now consider their joint execution in the collection $\{PSC_1, \ldots, PSC_n\}$. The execution of a step is the same as in [8], as probabilistic aspects are not involved anymore. On the level of a single statechart, executing a step consists of two parts: updating the variables and events occurring in the actions and determining the new state. As the actions of one P-statechart may influence the sets of events of other P-statecharts, we describe the step execution of the complete collection of P-statecharts.

Default completion. The *default completion* C' of some set of nodes C is the smallest superset of C such that C' is a configuration. If C' contains an OR-node x but C contains none of its descendants, C' contains its default node $default_i(x)$.

Executing a step. Given configurations $(C_1, \ldots, C_n)$, steps $(T_1, \ldots, T_n)$, and valuations $(V_1, \ldots, V_n)$, we define for P-statechart PSC_i the new state (C_i', I_i', V_i') by:

- C_i' is the default completion of the union of $\bigcup_{(j,A,Y)\in T_i} Y$ (all target nodes entered) and

$$\{x \in C_i \mid \forall (j, A, Y) \in T_i : x \text{ is not a descendant of } scope(\iota(j).X, Y)\}.$$

- $I_i' = \bigcup_{k=1}^{n} \{\mathsf{e} \mid \exists (j, A, Y) \in T_k : \textbf{send } i.\mathsf{e} \in A\}$
- $V_i' = V_i[\{\mathsf{v} := expr \mid \mathsf{v} := expr \in A, (j, A, Y) \in T_i\}]$, the same valuation as before except for the assignments in any action of the step. If these assignments are inconsistent, pick (nondeterministically) any of them.[6]

We denote this as: $Execute(C_{1\ldots n}, T_{1\ldots n}, V_{1\ldots n}) = ((C_1', I_1', V_1'), \ldots, (C_n', I_n', V_n'))$.

3.4 BPTS Semantics

Recall that in the step construction algorithm we resolve probabilistic choices prior to resolving non-determinism. A semantic model – that contains both nondeterminism and discrete probabilities – preferably obeys the same order. Bundle probabilistic transition systems (BPTS) [7] is one of the rare models for which this is indeed the case.

[6] This is according to both Eshuis' and Damm's semantics. In the article, I first wrote that V_i' be undefined.

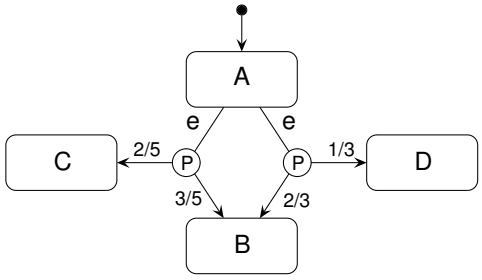

Fig. 4. Example of non-disjoint transitions

Bundle probabilistic transition systems. A *BTPS* is a quadruple (Σ, T, L, σ_0) where:

- Σ is a finite, non-empty set of states.
- The transition relation T assigns to each state a probability space over $\mathbb{P}(\Sigma)$.
- $L : \Sigma \to \mathbb{P}(AP)$ assigns to each state a set of atomic propositions.
- $\sigma_0 \in \Sigma$ is the initial state.

Our definition differs from the original one as follows: [7] allow infinite Σ; they include indices to distinguish individual transitions and a set of actions; and they omit the state-labelling L and the initial state σ_0. In addition, they require that the transitions are disjoint. We do not constrain the transitions because we see no reason to forbid P-statecharts like the one in Fig. 4.

BPTS semantics of a collection of P-statecharts. Let the set of atomic propositions $AP = \bigcup_{i=1}^{n} \{i.isin(x) \mid x \in Nodes_i\}$. For a finite collection of P-statecharts (indexed from 1 to n), the BPTS (Σ, T, L, σ_0) is defined by:

- $\Sigma = \bigtimes_{i=1}^{n} (Conf_i \times \mathbb{P}(Events) \times Val_i)$.
- Given a state $\sigma = ((C_1, I_1, V_1), \ldots, (C_n, I_n, V_n)$, let $T(\sigma) = (\mathbb{P}(\Sigma), P)$ where P is the lift of the following probability weight μ to sets of sets: For $E_i \subseteq Edges_i$, let $\mu(\{s \mid \exists T_i \in Steps(E_i) : s = Execute(C_{1\ldots n}, T_{1\ldots n}, V_{1\ldots n})\}) = \prod_{i=1}^{n} P_i(\{E_i\})$, where P_i is the probability measure of $\mathcal{PR}_{(C_i,I_i,V_i)} = (\mathbb{P}(Edges_i), P_i)$ mentioned in the explanation of Algorithm 1. For other sets $S \subseteq \Sigma$, let $\mu(S) = 0$.
- $L((s_1, \ldots, s_n)) = \bigcup_{i=1}^{n} \{i.isin(x) \mid x \in C_i\}$.
- $\sigma_0 = ((C'_{0,1}, \varnothing, V_{0,1}), \ldots, (C'_{0,n}, \varnothing, V_{0,n}))$, where $C'_{0,i}$ is the default completion of $\{root_i\}$ and $V_{0,i}$ is the initial valuation in the ith P-statechart.

3.5 Markov Decision Process Semantics

Although a BPTS model would be appropriate as semantical model, we also give a semantics in terms of the (more standard) model of Markov decision processes

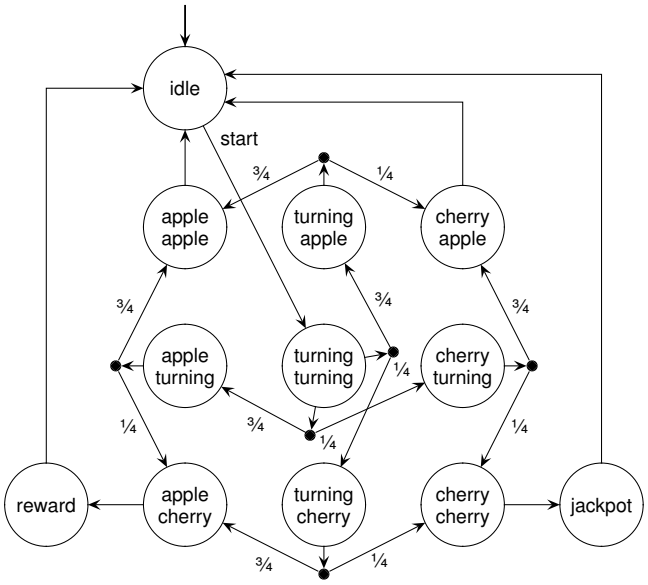

Fig. 5. Model of a simple gambling machine

(MDP) [20]. This slightly complicates the semantics, but has the nice property that it facilitates model checking of probabilistic properties.

We now embed the step semantics described above in a global semantics, mapping a collection of P-statecharts onto a finite Markov decision process. To allow the interpretation of temporal logic formulas later on, we equip an MDP with a state-labelling that assigns a set of atomic propositions to states. We assume a given fixed set AP of atomic propositions.

Markov decision processes. An *MDP* is a quadruple $(S, \mathit{Distr}, L, s_0)$ where:

- S is a finite, non-empty set of states.
- Distr assigns to each state a finite, non-empty set of distributions[7] on S.
- $L : S \to \mathbb{P}(AP)$ assigns to each state a set of atomic propositions.
- $s_0 \in S$ is the initial state.

In state s, the atomic propositions in $L(s)$ hold. Informally speaking, an MDP exhibits the following behaviour. Whenever the system is in state s, a probability distribution $\mu \in \mathit{Distr}(s)$ is chosen nondeterministically. Then, the system chooses probabilistically the next state according to the selected distribution μ.

Example 2. Figure 5 describes a simple gambling machine with two reels. In most states, the figure shows the state of the first reel atop the state of the second

[7] Probability textbooks [22] call this a probability weight, but "distribution" is found in the MDP literature.

reel. The reels, once stopped, show apples (with probability $\frac{3}{4}$) or cherries (with probability $\frac{1}{4}$). They do not stop in some specific order. When both reels show cherries, the jackpot state is reached; in the combination apple / cherry, there is some reward.[8]

Paths in an MDP. A *path* is an infinite sequence of states $(s_0, s_1, \ldots)$ such that s_0 is the initial state and the probability that s_{i+1} is reached from s_i is > 0, for each i. A path represents a possible behaviour of an MDP.

MDP semantics of a collection of P-statecharts. In an MDP, first a non-deterministic choice is made (among the available distributions) after which a next state is selected probabilistically. This order is reversed for the construction of a step of a P-statechart. To overcome this difference, we add auxiliary states to the MDP. Recall that (original) states consist of, per P-statechart, a configuration, a set of events, and a valuation, written (C, I, V). Auxiliary states will correspond to the outcome of Task 2 of the step construction algorithm and consist of, per P-statechart, a configuration, a set of enabled edges and a valuation, written (C, E, V). Each auxiliary state will be labelled with the distinguished atomic proposition $\triangle$. It offers a non-deterministic choice of trivial distributions (assigning probability 1 to a single state) only, such that each successor state is an original state (not labelled $\triangle$). Original states, in turn, do only possess singleton sets of probability distributions (hence there is no non-determinism), and all states with positive probability will be states labelled $\triangle$. This type of MDPs is also known as (strictly) alternating probabilistic transition systems [10].

Let the set $AP = \{\triangle\} \cup \bigcup_{i=1}^{n} \{i.isin(x) \mid x \in Nodes_i\}$. For a finite collection of P-statecharts (indexed from 1 to n), the MDP $(S, Distr, L, s_0)$ is defined by:

- $S = O \cup A$ where

  $$O = \bigtimes_{i=1}^{n} (Conf_i \times \mathbb{P}(Events) \times Val_i) \text{ and } A = \bigtimes_{i=1}^{n} (Conf_i \times \mathbb{P}(Edges_i) \times Val_i).$$

- $Distr((s_1, \ldots, s_n)) =$
  - $\{(s'_1, \ldots, s'_n) \mapsto \prod_{i=1}^{n} \mu_i(s'_i)\}$, if $(s_1, \ldots, s_n) \in O$, where μ_i is the probability weight corresponding to $\mathcal{PR}_{s_i} = (\mathbb{P}(Edges_i), P)$, defined by $\mu_i(C_i, E', V_i) = P(\{E'\})$ for $E' \subseteq Edges_i$ and $\mu_i(s) = 0$ for other states s.[9]
  - $\{\mu_s^1 \mid \exists T_i \in Steps(E_i) : s = Execute(C_{1\ldots n}, T_{1\ldots n}, V_{1\ldots n})\}$ otherwise. μ_s^1 denotes the trivial distribution that assigns probability 1 to state s.
- $L((s_1, \ldots, s_n)) = \begin{cases} \bigcup_{i=1}^{n} \{i.isin(x) \mid x \in C_i\} & \text{if } (s_1, \ldots, s_n) \in O \\ \{\triangle\} & \text{otherwise} \end{cases}$

[8] Because a gambling machine contains multiple similar reels that exhibit parallel, highly independent behaviour, it is a good example of an application of P-statecharts! However, an explicit modelling style, as shown in Fig. 5, makes it easier to assign rewards to specific combinations of outcomes.

[9] If no edges are enabled in P-statechart i, this will lead to $\mu_i(C_i, \varnothing, V_i) = 1$.

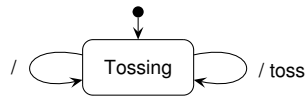

Fig. 6. P-statechart that generates "toss" events at random

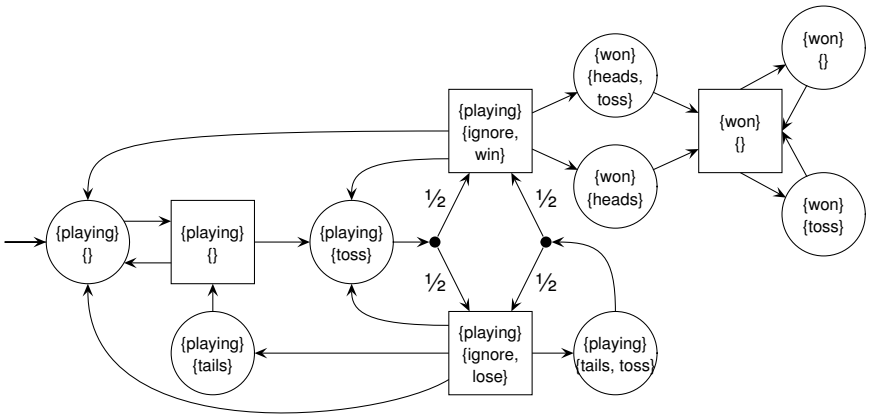

Fig. 7. MDP semantics of the P-statechart of Fig. 1

– $s_0 = ((C'_{0,1}, \varnothing, V_{0,1}), \ldots, (C'_{0,n}, \varnothing, V_{0,n})) \in O$, where $C'_{0,i}$ is the default completion of $\{root_i\}$ and $V_{0,i}$ is the initial valuation in the ith P-statechart.

where $s_i = (C_i, I_i, V_i)$ if $(s_1, \ldots, s_n) \in O$ and $s_i = (C_i, E_i, V_i)$ otherwise.

Example 3. To illustrate how P-statecharts are mapped onto MDPs, we consider the "unreliable, but fair coin" P-statechart from Fig. 1. We compose this P-statechart with an event generator, which generates "toss" events at random; see Fig. 6. (If we don't add a component which generates "toss" events, the only reachable state of the system would be the initial state, due to the closed world assumption.) The MDP semantics of this collection of two P-statecharts is illustrated in Fig. 7. Here, original states of the form $(C, I, \varnothing)$ (where I is a set of events) are shown by circles with the sets C and I inscribed. Auxiliary states of the form $(C, E, \varnothing)$ (where E is a set of edges) are shown by boxes with the sets C and E inscribed. The names used for edges are shown in *italics* in Fig. 1; the node and edges of the event generator are omitted.

4 Comparison to Non-Probabilistic Semantics

In this section, we compare our P-statechart semantics to the semantics of the corresponding non-probabilistic statechart. For the sake of simplicity, we adapt

the semantics of [8] to our notation and abstract from some minor aspects of their semantics (the very same aspects mentioned in Sect. 1).

A (traditional) statechart is a tuple $(\mathit{Nodes}, \mathit{Events}, \mathit{Vars}, \mathit{Edges})$ where $\mathit{Edges} \subseteq \mathbb{P}(\mathit{Nodes}) \times (\mathit{Events} \cup \{\bot\}) \times \mathit{Guards} \times \mathbb{P}(\mathit{Actions}) \times \mathbb{P}(\mathit{Nodes})$ is a set of edges, and the other components are as for a P-statechart.[10] A step is – like before – an enabled, consistent, prioritized and maximal set of edges. The execution of steps in a finite collection of statecharts leads to a new state of the statecharts similar to the procedure described in Sect. 3.3. The semantics of a collection of statecharts is a Kripke structure (instead of an MDP). A Kripke structure KS is a quadruple (S, T, L, s_0), where S, L and s_0 are defined as for MDPs and $T \subseteq S \times S$ is the (non-probabilistic) transition relation.

Projections. To facilitate the comparison, we define three projections: one that abstracts from the probabilistic choices in a P-statechart (called α_1), and one that abstracts from the probabilistic choices in an MDP (called α_2). The projections replace probabilities by nondeterminism. Let $\alpha_1(\mathit{PSC}_i) = \mathit{SC}_i$ where SC_i is obtained from PSC_i by replacing the set of P-edges PEdges_i by the set of edges Edges'_i and by adding some variables to handle the interplay between probabilities and priorities correctly. Later on, we will use a third projection π to remove the additional variables and *Init* again. Further, $\alpha_1(\{\mathit{PSC}_1, \ldots, \mathit{PSC}_n\}) = \{\alpha_1(\mathit{PSC}_1), \ldots, \alpha_1(\mathit{PSC}_n)\}$.

Definition of α_1. A naïve definition of α_1 would just be $\alpha_1(\mathit{PSC}_i) = (\mathit{Nodes}_i, \mathit{Events}, \mathit{Vars}_i, \mathit{Edges}_i)$. However, in the case that some edge has a higher priority than another edge which belongs to the same P-edge, the latter is never taken. For example, the P-statechart of Fig. 2 contains edges of this kind.

To solve this problem, we refine the translation as follows: The main idea is to add some extra variables to *Vars* and guards to the edges. As a consequence, at most one of the edges corresponding to a P-edge is enabled.

1. For each P-edge $\iota(d) = (X, e, g, P) \in \mathit{PEdges}_i$, define $T_d := \{(d, A, Y) \mid P(A, Y) > 0\} \subseteq \mathit{Edges}_i$. This set is finite.
2. If T_d contains two edges that have different priority, choose a bijective index function $\iota_d : \{1, \ldots, |T_d|\} \to T_d$.
   Without loss of generality, we assume that the P-edges $\iota(1), \ldots, \iota(d_0)$ are the ones with edges of different priority (for a suitable $d_0 \in \{0, \ldots, |\mathit{PEdges}_i|\}$).
3. Now, translate the P-statechart PSC as follows to a statechart $\alpha_1(\mathit{PSC}_i) = (\mathit{Nodes}'_i, \mathit{Events}_i, \mathit{Vars}'_i, \mathit{Edges}'_i)$:
   - $\mathit{Nodes}'_i = \mathit{Nodes}_i \cup \{\mathit{Init}\}$. A new initial node *Init* is added. *Init* is a child of root_i; it becomes its default child; it is a BASIC node. For the rest, the set of nodes, its tree structure and its types are not changed.
   - The set *Events* is unchanged.
   - $\mathit{Vars}'_i = \mathit{Vars}_i \cup \{\mathsf{v}_1, \mathsf{v}_2, \ldots, \mathsf{v}_{d_0}\}$, where the v_i are new variables, i. e., $\mathsf{v}_i \notin \mathit{Vars}_i$.

[10] In Sect. 3.1, an edge is defined slightly different, by referring to P-edges. We have to adapt the definition here to the fact that a statechart has no P-edges.

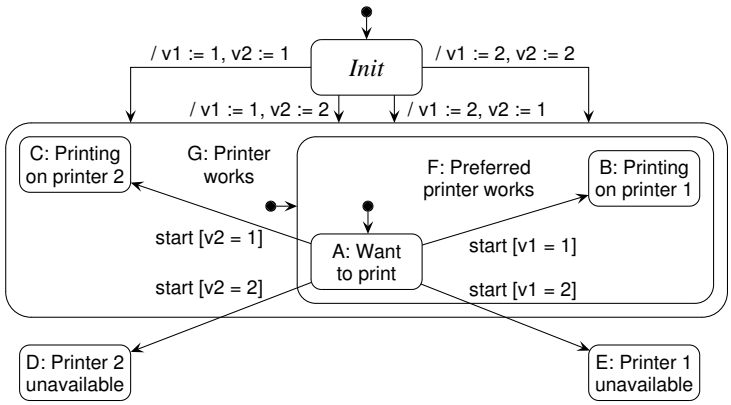

Fig. 8. A translation of Fig. 2 to a statechart

- The sets $Guards'_i$ and $Actions'_i$ of the translated statecharts are defined as normal and therefore include assignments to the new variables.
- Given a P-edge $\iota(d) = (X, e, g, P)$ of PSC_i with $d > d_0$ and a possible target $(d, A, Y) \in T_d$, the statechart contains an edge (X, e, g, A', Y) if the set of actions A' is $A \cup \{\mathsf{v}_j := k_j\}$, for some choice of k_j, for $j \in \{1, \ldots, d_0\}$.
- Given a P-edge $\iota(d) = (X, e, g, P)$ of PSC_i with $d \leqslant d_0$ and a possible target $\iota_d(k) = (d, A, Y) \in T_d$, the statechart contains an edge (X, e, g', A', Y) if the guard g' is $g \wedge \mathsf{v}_d = k$ and A' is defined as above.
- There are edges from *Init* to the original default child of $root_i$: $(\{Init\}, \bot, true, A', \{default_i(root_i)\})$, where $A' = \{\mathsf{v}_j := k_j\}$, for some choice of k_j, for $j \in \{1, \ldots, d_0\}$.

Example 4. Figure 8 illustrates α_1: it contains a possible translation of Fig. 2 to a statechart. As both P-edges in the P-statechart have edges of different priority, α_1 adds two variables.

Definition of π. Given a finite collection of P-statecharts $(PSC_1, \ldots, PSC_n)$, we define the projection π from $KS'_2 = sem_2(\alpha_1(PSC_1, \ldots, PSC_n)) = (S, T, L, s_0)$ to the Kripke structure $KS_2 = (S', T', L', s'_0)$:

- Given a state $s = ((C_1, I_1, V_1), \ldots, (C_n, I_n, V_n)) \in S$, S' contains the state $\pi(s) = ((C_1, I_1, V'_1), \ldots, (C_n, I_n, V'_n))$, where $V'_i = V_i \restriction_{Vars_i \setminus \{\mathsf{v}_1, \ldots, \mathsf{v}_{d_0}\}}$. ($\restriction$ denotes the restriction of a function to some subset of its original domain.)
- Given a transition $(s_1, s_2) \in T$ where $s_1 \neq s_0$, T' contains the transition $(\pi(s_1), \pi(s_2))$.
- $L'(s') = L(s)$, for any s such that $\pi(s) = s'$. Note that L' is well-defined because the truth values of atomic propositions do not depend on the variables.

– $s'_0 = \pi(s)$, for any s such that $(s_0, s) \in T$. Note that s'_0 is well-defined because s_0 is (the image of) the pre-initial state *Init* added by α_1; all transitions from s_0 lead to states that only differ in the values of $\mathsf{v}_1, \ldots, \mathsf{v}_{d_0}$.

Note that although π identifies some states, it does not add any new behaviour; for any path in $\pi(sem_2(\alpha_1(S)))$, there is a corresponding path in $sem_2(\alpha_1(S))$.

Definition of α_2. The projection α_2 is defined by: $\alpha_2(S, Distr, L, s_0) = (S', T, L\restriction_{S'}, s_0)$, where:

– $S' = \{((C_1, I_1, V_1), \ldots, (C_n, I_n, V_n)) \in S \mid I_i \subseteq Events\}$
– $T = \{(s, s') \mid \exists \mu \in Distr(s) : (\exists \tilde{s} \in S : \mu(\tilde{s}) > 0 \wedge \exists \mu' \in Distr(\tilde{s}) : \mu'(s') > 0)\}$

State set S' contains all non-auxiliary states, and $(s, s') \in T$ whenever s can move to s' via some auxiliary state $\tilde{s}$ in the original MDP with positive probability.

Theorem 1. *The following diagram commutes:*

$$\begin{array}{ccc} \{PSC_1, \ldots, PSC_n\} & \xrightarrow{\alpha_1} & \{SC_1, \ldots, SC_n\} \\ {\scriptstyle sem_1}\downarrow & & \downarrow{\scriptstyle \pi \circ sem_2} \\ MDP & \xrightarrow{\alpha_2} & KS \end{array}$$

where sem_1 denotes our MDP-semantics and sem_2 denotes the KS-semantics of [8].

Proof. Assume given a finite collection of P-statecharts $\mathcal{C} = \{PSC_1, \ldots, PSC_n\}$ with $PSC_i = \{(Nodes_i, Events, Vars_i, PEdges_i$. We denote $KS_1 = (S_1, Tr_1, L_1, s_{0,1}) = \alpha_2(sem_1(\mathcal{C}))$ and $KS_2 = (S_2, Tr_2, L_2, s_{0,2}) = \pi(sem_2(\alpha_1(\mathcal{C})))$. We have to prove: $KS_1 = KS_2$.

– The sets of states are the same. The set of MDP states according to $sem_1(\mathcal{C})$ is $\bigtimes_{i=1}^{n} Conf_i \times Events \times Val_i \cup \bigtimes \cdots$. Consequently, S_1 is the first part of this union: $S_1 = \bigtimes_{i=1}^{n} Conf_i \times Events \times Val_i$.
  S_2 contains all non-intermediary states of the statecharts $\alpha_1(\mathcal{C})$ (without the extra variables). These are exactly the states in S_1.
– In each state, the transitions are the same. The central point of this proof is: The sets of steps in both semantics correspond to each other.
  Assume given a state $s_i = (C_i, I_i, V_i)$ of one component of $\mathcal{C}$ (according to sem_1) and a step T_i with positive probability. We have to prove that there is a corresponding step T'_i in a corresponding state $s'_i = (C_i, I_i, V'_i)$ of $sem_2(\alpha_1(\mathcal{C}))$.
  V'_i is defined by: $V'_i(\mathsf{v}_d) = k$ if $\iota_d(k) \in T_i$; $V'_i(\mathsf{v}) = V_i(\mathsf{v})$ if $\mathsf{v} \in Vars_i$; choose any value for $V'_i(\mathsf{v}_d)$ otherwise.
  We have some freedom of choice in the target of T'_i: the step T_i does not prescribe any assignments to the new variables $\mathsf{v}_1, \ldots, \mathsf{v}_{d_0}$. If one considers

a path, one would choose the target that enables the next step in the path. For this proof, we assume given an assignment $a : \mathsf{v}_j \mapsto a(\mathsf{v}_j)$ that describes the desired new values.

- Given $(d, A, Y) \in T_i$, where $d > d_0$, T_i' contains the edge $(\iota(d).X, \iota(d).e, \iota(d).g, A', Y)$, where $A' = A \cup \{\mathsf{v}_j := a(\mathsf{v}_j)\}$.
- Given $(d, A, Y) \in T_i$, where $d \leqslant d_0$, T_i' contains the edge $(\iota(d).X, \iota(d).e, g', A', Y)$, where $g' = \iota(d).g \wedge \mathsf{v}_d = \iota_d^{-1}(d, A, Y)$ and A' is defined as above.

This set is a step: it contains only edges enabled in s_i' and it fulfils the other step properties because T_i is a step.

Conversely, when given a state $s_i' = (C_i, I_i, V_i')$ of one component of S (according to $sem_2(\alpha_1(S))$) and a step T_i', we have to prove that there is a corresponding step T_i in a corresponding state $s_i = (C_i, I_i, V_i)$ of $sem_1(S)$. This is exactly the inverse operation of the above: restrict V_i' to V_i and define T_i by deleting references to the new variables v_j from T_i'.

The complete proof that the transitions are the same can easily be derived from the above.

– In each state, the same propositions hold in both translations. Trivial.
– The two translations have the same initial state. Easy. □

Corollary 1. *The P-statechart semantics is a conservative extension of the statechart semantics of [8].*

5 Property Specification for P-Statecharts

As a property specification language for P-statecharts we propose to use the probabilistic branching time logic PCTL, which extends CTL with probabilistic features. PCTL was originally interpreted over fully probabilistic systems, i. e., systems that do not exhibit any non-determinism [11]. We use the interpretation of PCTL over MDPs defined by Baier and Kwiatkowska [1, 16],[11] similar to pCTL and pCTL* [3]. PCTL allows one to express properties such as (Ψ) "the probability that a system crashes within 13 steps without ever visiting certain states is at most 10^{-5}". In order to decide these properties, the non-determinism is resolved by means of schedulers (also known as adversaries or policies). Temporal formulas are then interpreted with respect to all schedulers or some schedulers. Here, we restrict ourselves to the fragment of PCTL for which actual model-checking tool-support is available; i. e., we only consider path properties interpreted for all fair schedulers. Formulas of the form "There is a fair scheduler such that ..." can be checked via duality. The model-checking algorithm thus returns "true" for property (Ψ), iff (Ψ) holds for all fair schedulers that resolve the non-determinism. For simplicity, we do not consider next-formulas.[12]

[11] Baier and Kwiatkowska sometimes call the logic PBTL.

[12] This is done because in our MDP semantics, a single step is translated to two consecutive steps.

Syntax and informal semantics. The syntax of PCTL is given by the following grammar, where a denotes an atomic proposition, $p \in [0,1]$ denotes a probability and $\sqsupseteq$ is a placeholder for a comparison operator $<, \leqslant, =, \geqslant, >$:

$$\varphi, \psi ::= \mathbf{true} \mid \mathbf{false} \mid a \mid \mathsf{v} \leqslant k \mid \mathsf{v} \geqslant k \mid \varphi \wedge \psi \mid \neg\varphi \mid \mathcal{P}_{\sqsupseteq p}[\varphi\, \mathcal{U}^{\leqslant k}\, \psi] \mid \mathcal{P}_{\sqsupseteq p}[\varphi\, \mathcal{U}\, \psi]$$

The meaning of **true**, comparisons, conjunction and negation is standard. Recall from Sect. 3.5 that atomic propositions are $\triangle$ and $i.isin(x)$, which holds in states where P-statechart i is in node x. Formula $\mathcal{P}_{\sqsupseteq p}[\varphi\, \mathcal{U}^{\leqslant k}\, \psi]$ holds in a state if the probability of the set of paths that reach a ψ-state in at most k steps while passing only through φ-states is $\sqsupseteq p$. Property Ψ, e. g., is expressed as $\mathcal{P}_{\leqslant 10^{-5}}[\neg\varphi\ \mathcal{U}^{\leqslant 13}\ crash]$ where φ describes the states that should be avoided. $\mathcal{P}_{\sqsupseteq p}[\varphi\ \mathcal{U}\ \psi]$ has the same meaning, but does not put a bound on the number of steps needed to reach the ψ-state. The temporal operator $\Diamond$ can be defined e. g., as $\mathcal{P}_{\sqsupseteq p}[\Diamond^{\leqslant k}\varphi] = \mathcal{P}_{\sqsupseteq p}[\mathbf{true}\, \mathcal{U}^{\leqslant k}\, \varphi]$. (A formal interpretation on MDPs is omitted here, and can be found in [1]).

Schedulers and fair schedulers. The above explanation is ambiguous if non-determinism is present, because the probability will (in general) depend on the resolution of non-determinism. Non-determinism is resolved by schedulers. A *scheduler* selects, for each initial fragment of a path through the MDP, one of the possible (non-deterministic) continuations. It does not resolve probabilistic choices. Several types of schedulers do exist, see [1]. Here, we consider *fair* schedulers. A fair scheduler only selects fair paths. A path π is *fair* if, for each state s that appears infinitely often in π, each of the possible non-deterministic continuations in s also appears infinitely often. Thus, for instance, $\mathcal{P}_{\leqslant 10^{-5}}[\neg\varphi\, \mathcal{U}^{\leqslant 13}\ crash]$ is valid if for all fair schedulers the probability to reach a crash-state within 13 steps (without visiting a φ-state) is at most 10^{-5}. From now on, we assume all PCTL-formulas to be interpreted over all fair schedulers. (Properties that should be interpreted over some fair scheduler can be stated via duality, e. g.: $\mathcal{P}_{\leqslant p}(red\ \exists\mathcal{U}\ blue) \equiv \neg\mathcal{P}_{>p}(red\, \mathcal{U}\ blue)$.)

Example 5. For the P-statechart in Fig. 1, we express the following properties:

- "The probability that eventually the game will be over is 1":

$$\mathcal{P}_{=1}[\Diamond\ \neg 1.isin(playing)]$$

- "In less than 50 % of the cases, the game will be won within at most 20 steps":

$$\mathcal{P}_{<0.5}[\Diamond^{\leqslant 20}\ 1.isin(won)]$$

PCTL interpreted over P-statecharts. In our setting, a formula is interpreted over a finite collection of P-statecharts $\{PSC_1, \ldots, PSC_n\}$ and one of its states $(s_1, \ldots, s_n)$ where $s_i = (C_i, I_i, V_i)$. Formally, the semantics is defined via the MDP semantics, i. e., $\{PSC_1, \ldots, PSC_n\} \vDash \varphi$ iff the corresponding MDP satisfies $\overline{\varphi}$. Here $\overline{\ }$ denotes a syntactic translation needed to "hop along" the auxiliary

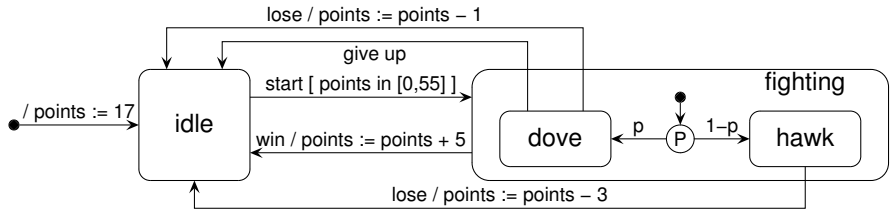

Fig. 9. Statechart of a contestant in the hawk–dove-game.

(△-labelled) MDP states. It is defined by induction over the structure of formulas. For elementary PCTL-formulas, such as atomic propositions and variable constraints, this translation is simply the identity, e. g., $\overline{\mathbf{true}} = \mathbf{true}$. For the remaining operators we have:

$$\begin{aligned}
\overline{\varphi \wedge \psi} &= \overline{\varphi} \wedge \overline{\psi} \\
\overline{\neg\varphi} &= \neg\overline{\varphi} \\
\overline{\mathcal{P}_{\sqsupseteq p}[\varphi\, \mathcal{U}^{\leqslant k}\, \psi]} &= \mathcal{P}_{\sqsupseteq p}[(\overline{\varphi} \vee \triangle)\, \mathcal{U}^{\leqslant 2k}\, \overline{\psi}] \\
\overline{\mathcal{P}_{\sqsupseteq p}[\varphi\, \mathcal{U}\, \psi]} &= \mathcal{P}_{\sqsupseteq p}[(\overline{\varphi} \vee \triangle)\, \mathcal{U}\, \overline{\psi}]
\end{aligned}$$

6 Example: Hawks and Doves

This section applies P-statecharts and PCTL to the specification and verification of the behaviour of a small example taken from theoretical biology. Conflicts between animals are often analysed using simulation techniques. We consider the following variant of the hawk–dove-game [4, 18]. In a population of animals, individuals combat for some advantage (such as food, dominance, or mates), their success being measured in points. Individuals may fight using several strategies. In particular, we consider

Hawk strategy: Hawk-like individuals will fight with great effort, until they win the contest (+5 points) or are severely injured (−3 points).

Dove strategy: Dove-like individuals will fight with limited effort, until they win the contest (+5 points) or give up after some fight (−1 point). When facing a hawk, they immediately give up (±0 points).

We consider a small scenario with three individuals and an arbiter. In every round, the arbiter chooses nondeterministically a pair of individuals; they will be opponents in the next contest. The two individuals select probabilistically the hawk or dove strategy. The arbiter decides who wins. Figures 9 and 10 show the P-statechart for one individual and the arbiter, respectively. The players all start off with 17 points and the individual scores may float in the interval $[0, 55]$ (otherwise they stop). Applying the MDP semantics of Sect. 3 together with some further optimisations (leaving out trivial intermediary states, encoding the configuration efficiently) leads to a system of 3,147,947 reachable states. The size

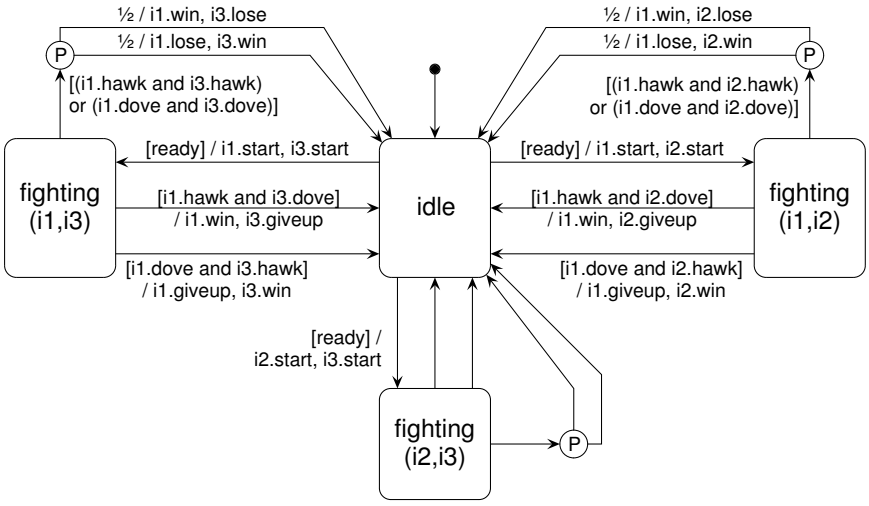

Fig. 10. Statechart of the arbiter in the hawk–dove-game. We have omitted some P-edge labels from and to node fighting(i2,i3), which are analogous the other fighting nodes.

of the state space is mainly dominated by the integer variables storing the scores. Different scenarios were checked with the model checker PRISM [16] where each scenario consisted of different types of animals. These types were generated by taking different values for p, the probability to behave like a dove. Formulas are checked for the initial state. The three considered scenarios are the following.

One daring and two careful players. This is a scenario with two individuals (c_1 and c_2) for which $p = 0.75$ and one individual (d) with $p = 0.5$. The probability that any individual dies (its points drop below zero: $dead(i) := (points_i < 0)$) turns out to be very small, with the daring individual running a higher risk of being killed, since

$$\mathcal{P}_{\leqslant 10^{-7}}[(\neg dead(c_1) \wedge \neg dead(d))\, \mathcal{U}\ dead(c_2)] \quad \text{holds}$$

(and likewise with c_1 and c_2 reversed), but

$$\mathcal{P}_{\leqslant 10^{-7}}[(\neg dead(c_1) \wedge \neg dead(c_2))\, \mathcal{U}\ dead(d)] \quad \text{is refuted.}$$

The actual probability of d dying first is at most (depending on the scheduler) $7.206 \cdot 10^{-7}$, while the probability of the careful one dying first is at most $5.923 \cdot 10^{-8}$ (each). On the other hand, the daring individual is likely to outperform the others on accumulating a certain number of points, say 37. This follows from verifying:

$$\mathcal{P}_{<0.5}[(points_{c_1} < 37 \wedge points_d < 37)\, \mathcal{U}\ points_{c_2} \geqslant 37] \quad \text{which is valid, and}$$

$$\mathcal{P}_{\leqslant 0.75}[(points_{c_1} < 37 \wedge points_{c_2} < 37)\, \mathcal{U}\ points_d \geqslant 37] \quad \text{which is refuted.}$$

Three aggressive players. In this scenario each animal (d_1, d_2, d_3) plays hawk with probability 0.9 (i. e., $p = 0.1$). The probability that some of the individuals dies is relatively high, e. g.,

$$\mathcal{P}_{\leqslant 0.01}[(\neg \mathit{dead}(d_1) \wedge \neg \mathit{dead}(d_2))\, \mathcal{U}\ \mathit{dead}(d_3)] \quad \text{is refuted}$$

(and likewise for the permutations of the d_i). So, there are schedulers which will lead to d_3 dying first with more than 1 % chance. The probability that one of the individuals gets more than 37 points within 100 steps is always less than 0.75, as

$$\mathcal{P}_{<0.75}[\Diamond^{\leqslant 100}\ (\mathit{points}_{d_1} \geqslant 37)] \quad \text{holds.}$$

Three careful players. In the opposite situation (the three individuals play dove with probability 0.9), the individuals (c_1, c_2, c_3) are less likely to die and more likely to get a reward fast. The probability that any of the individuals dies is rather low as, e. g.,

$$\mathcal{P}_{\leqslant 10^{-10}}[(\neg \mathit{dead}(c_1) \wedge \neg \mathit{dead}(c_2))\, \mathcal{U}\ \mathit{dead}(c_3)] \quad \text{holds.}$$

So, for any scheduler, the probability of c_3 dying first never exceeds 10^{-10}. The probability that one of the individuals gets more than 37 points within 100 steps turns out to be greater than 0.8, since

$$\mathcal{P}_{\leqslant 0.8}[\Diamond^{\leqslant 100}\ (\mathit{points}_{c_1} \geqslant 37)] \quad \text{is refuted.}$$

Conclusion. As a general conclusion of the experiments we may state that it is good for a population as a whole if the animals are careful; but an individual may be at an advantage if it is more daring than the others.

7 Discussion and Conclusion

This section discusses our approach in the broader context of statechart semantics and of probabilistic models.

Contribution. This paper has developed a recipe to conservatively extend a statechart dialect with probabilistic features. We have applied this recipe to the requirement-level UML semantics of [8], mapping the probabilistic extension onto Markov decision processes as semantic models. Further, we have shown how to use the probabilistic logic PCTL to specify properties over probabilistic statecharts and how model checking of probabilistic statecharts can be performed effectively.

Adaptation to other statechart semantics. Various semantics have been published for statecharts, [2] lays out the spectrum of the many possibilities in defining a semantics. The extension to P-statecharts described in this paper can be applied to a wide range of other semantic definitions. The main idea of our extension is:

1. Syntactically, probabilities are trigger-guarded, i. e., reactions to triggers may depend on the result of a probabilistic experiment, whereas the triggers themselves are not subjected to probabilities. This restricts our approach to describing system randomness, opposed to environmental randomness.
2. Semantically, we reduce the P-statechart probabilistically to a (traditional) statechart, and this is done just before a step. The step is constructed and executed in the traditional statechart setting, and the step's result is interpreted in the P-statechart again. Such a reduction is possible as long as the effects of probabilistic experiments are encapsulated in the steps.

In principle it is possible to define a (traditional) statechart semantics which – if interpreted in the probabilistic extension – would break the encapsulation of probabilities within a step. For instance, one could imagine a semantics where a state variable depends on the enabledness of specific outgoing edges (which could only be decided after resolving the probabilism). However, such a feature appears to be rarely used, the overview given in [2] does not mention any feature like this.

Possible simplifications. Sect. 3.5 has mentioned BPTS as the most natural model for a P-statechart semantics. Thus, we could have simplified the semantics if there were a BPTS model checker available.

On the other hand, observe that the examples in Sect. 3.1 depend on the fact that some P-edge allows a probabilistic choice between target nodes with different parents. For most priority schemes, and for consistency, not the actually entered nodes are relevant, but their parents. If we disallow probabilistic choices between target nodes with different parents, we could resolve nondeterminism and probabilism in a different order and simplify several points: Theorem 1 can be formulated and proved simpler. It becomes feasible to give a direct semantics in terms of MDPs (i. e., without intermediary states), which is closer to the intuition behind P-statecharts. However, it is no more possible to express behaviours like the examples in Sect. 3.1.

Lessons learnt. It is easy to formulate an intuitive extension of statecharts with probabilities. However, when we started formalising and detailing it, a delicate balance had to be found with the other features of statecharts. Our first version of Theorem 1, for example, didn't work properly in the case that some edge has a higher priority than another edge which belongs to the same P-edge.

We have tried to formulate the extension as powerful as possible. This also revealed the problems of extending statecharts more clearly. For some applications, a simpler extension, or a simpler variant of basic statecharts is enough. In that case, one should ask whether the result is worth the effort.

Future work. Apart from exploring the specification and verification approach to system randomness on larger case studies, we are intending to investigate the very same approach in the context of environmental randomness. This asks for modelling and verification support for probabilistic and timing aspects of external stimuli a reactive system is exposed to.

Acknowledgements. The authors thank Rik Eshuis for pointing out a flaw in an earlier version of Theorem 1. Roel Wieringa is thanked for his comments on a draft version of the paper.

References

1. Christel Baier and Marta Kwiatkowska. Model checking for a probabilistic branching time logic with fairness. *Distributed Computing*, 11(3):125–155, 1998.
2. Michael von der Beeck. A comparison of statecharts variants. In H. Langmaack, W.-P. de Roever, and J. Vytopil, editors, *Formal Techniques in Real-Time and Fault-Tolerant Systems : ... proceedings*, pages 128–148, Berlin, 1994. Springer.
3. Andrea Bianco and Luca de Alfaro. Model checking of probabilistic and nondeterministic systems. In P. S. Thiagarajan, editor, *Foundations of Software Technology and Theoretical Computer Science : ... proceedings*, volume 1026 of *LNCS*, pages 499–513, Berlin, 1995. Springer.
4. Philip H. Crowley. Hawks, doves, and mixed-symmetry games. *Journal of Theoretical Biology*, 204(4):543–563, June 2000.
5. M. Dal Cin, G. Huszerl, and K. Kosmidis. Quantitative evaluation of dependability critical systems based on guarded statechart models. In *Proceedings, 4th IEEE International Symposium on High-Assurance Systems Engineering : ... HASE*, pages 37–45, Los Alamitos, 1999. IEEE.
6. Werner Damm, Bernhard Josko, Hardi Hungar, and Amir Pnueli. A compositional real-time semantics of statemate designs. In Willem-Paul de Roever, Hans Langmaack, and Amir Pnueli, editors, *Compositionality : the significant difference. COMPOS '97*, volume 1536 of *LNCS*, pages 186–238, Berlin, 1998. Springer.
7. Pedro R. D'Argenio, Holger Hermanns, and Joost-Pieter Katoen. On generative parallel composition. In Christel Baier, Michael Huth, Marta Kwiatkowska, and Mark Ryan, editors, *PROBMIV'98 First International Workshop on Probabilistic Methods in Verification : Indianapolis, Ind. ... 1998*, volume 22 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 2000. URL: `http://www.elsevier.nl/locate/entcs/volume22.html`.
8. Rik Eshuis and Roel Wieringa. Requirements-level semantics for UML statecharts. In Scott F. Smith and Carolyn L. Talcott, editors, *Formal Methods for Open Object-Based Distributed Systems IV : ... FMOODS*, pages 121–140, Boston, 2000. Kluwer Academic Publishers.
9. Stefania Gnesi, Diego Latella, and Mieke Massink. A stochastic extension of a behavioural subset of UML statechart diagrams. In L. Palagi and R. Bilof, editors, *Fifth International Symposium on High-Assurance Systems Engineering (HASE)*, pages 55–64. IEEE Computer Society Press, 2000.
10. H. A. Hansson. *Time and Probability in Formal Design of Distributed Systems*. PhD thesis, University of Uppsala, 1991.
11. Hans Hansson and Bengt Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6:512–535, 1994.
12. David Harel and Eran Gery. Executable object modeling with statecharts. *Computer*, 30(7):31–42, July 1997. IEEE.
13. David Harel and Amnon Naamad. The STATEMATE semantics of statecharts. *ACM Transactions on Software Engineering and Methodology*, 5(4):293–333, 1996.
14. David N. Jansen, Holger Hermanns, and Joost-Pieter Katoen. A probabilistic extension of UML statecharts : specification and verification. In Werner Damm

and Ernst-Rüdiger Olderog, editors, *Formal Techniques in Real-Time and Fault-Tolerant Systems : 7th intl. symposium ... proceedings*, volume 2469 of *LNCS*, Berlin, 2002. Springer.

15. Peter King and Rob Pooley. Derivation of Petri net performance models from UML specifications of communications software. In Boudewijn R. Haverkort, Henrik C. Bohnenkamp, and Connie U. Smith, editors, *Computer Performance Evaluation : Modelling Techniques and Tools ; ... TOOLS 2000*, volume 1786 of *LNCS*, pages 262–276, Berlin, 2000. Springer.
16. Marta Kwiatkowska, Gethin Norman, and David Parker. Probabilistic symbolic model checking with prism: A hybrid approach. In Joost-Pieter Katoen and Perdita Stevens, editors, *Tools and Algorithms for the Construction and Analysis of Algorithms : ... TACAS*, volume 2280 of *LNCS*, pages 52–66, Berlin, 2002. Springer.
17. C. Lindemann, A. Thümmler, A. Klemm, M. Lohmann, and O. P. Waldhorst. Quantitative system evaluation with DSPNexpress 2000. In *Workshop on Software and Performance (WOSP)*, pages 12–17. ACM, 2000.
18. J. Maynard Smith and G. R. Price. The logic of animal conflict. *Nature*, 246, November 1973.
19. S. McMenamin and J. Palmer. *Essential Systems Analysis*. Yourdon Press, New York, 1984.
20. Martin L. Puterman. *Markov Decision Processes : Discrete Stochastic Dynamic Programming*. Wiley, New York, 1994.
21. R. Segala and N. Lynch. Probabilistic simulations for probabilistic processes. *Nordic Journal of Computing*, 2:250–273, 1995.
22. A. N. Shiryaev. *Probability*, volume 95 of *Graduate texts in mathematics*. Springer, New York, 1996.
23. UML Revision Task Force. *OMG UML Specification 1.3*. Object Management Group, 1999. `http://www.omg.org/cgi-bin/doc?ad/99-06-08`.