Probabilistic Behavioural State Machines

Peter Novák

Department of Informatics
Clausthal University of Technology
Julius-Albert-Str. 4, D-38678 Clausthal-Zellerfeld, Germany
peter.novak@tu-clausthal.de

Abstract Development of embodied cognitive agents in agent oriented programming languages naturally leads to writing underspecified programs. The semantics of BDI inspired rule based agent programming languages leaves room for various alternatives as to how to implement the action selection mechanism of an agent (paraphrased from [5]).
To facilitate encoding of heuristics for the non-deterministic action selection mechanism, I introduce a probabilistic extension of the framework of *Behavioural State Machines* and its associated programming language interpreter *Jazzyk*. The language rules coupling a triggering condition and an applicable behaviour are extended with labels, thus allowing finer grained control of the behaviour selection mechanism of the underlying interpreter. In consequence, the agent program not only prescribes a set of mental state transitions enabled in a given context, but also specifies a probability distribution over them.

1 Introduction

Situated cognitive agents, such as mobile service robots, operate in rich, unstructured, dynamically changing and not completely observable environments. Since various phenomena of real world environments are not completely specifiable, as well as because of limited, noisy, or even malfunctioning sensors and actuators, such agents must operate with *incomplete information*.

On the other hand, similarly to mainstream software engineering, *robustness* and *elaboration tolerance* are some of the desired properties for cognitive agent programs. Embodied agent is supposed to operate reasonably well also in conditions previously unforeseen by the designer and it should degrade gracefully in the face of partial failures and unexpected circumstances (robustness). At the same time the program should be concise, easily maintainable and extensible (elaboration tolerance).

Agent programs in the reactive planning paradigm [15] are specifications of partial plans for the agent about how to deal with various situations and events occurring in the environment. The inherent incomplete information on one side, stemming from a level of knowledge representation granularity chosen at the agent's design phase, and striving for robust and easily maintainable programs on the other yield a trade-off of *intentional underspecification* of resulting agent programs.

Most BDI inspired agent oriented programming languages on both sides of the spectrum between theoretically founded (such as *AgentSpeak(L)/Jason* [2], *3APL* [3] or *GOAL* [4]) to pragmatic ones (e.g., *JACK* [16] or *Jadex* [14]) facilitate encoding of underspecified, non-deterministic programs. Any given situation, or an event can at the same time trigger multiple behaviours, which themselves can be non-deterministic, i.e, can include alternative branches.

A precise and exclusive qualitative specification of behaviour triggering conditions is often impossible due to the, at the design time chosen and fixed, level of knowledge representation granularity. This renders the qualitative condition description a rather coarse grained means for steering agent's life-cycle. In such contexts, a quantitative heuristics steering the language interpreter's choices becomes a powerful tool for encoding developer's informal knowledge, or intuitions about agent's run-time evolutions. For example, it might be appropriate to execute some applicable behaviours more often than others, or some of them might intuitively perform better than other behaviours in the same context, and therefore should be preferably selected.

In this paper I propose a *probabilistic extension* of a rule-based agent programming language. The core idea is straightforward: language rules coupling a triggering condition with an applicable behaviour are extended with labels denoting a probability with which the interpreter's selection mechanism should choose the particular behaviour in a given context. The idea is directly applicable also to other agent programming languages, however here I focus on extension of the theoretical framework of *Behavioural State Machines* [10] and its associated programming language instance *Jazzyk*, which I use in my long-term research. One of elegant implications of the extension of the *BSM* framework is that subprograms with labelled rules can be seen as specifications of probability distributions over actions applicable in a given context. This allows steering agent's focus of deliberation on a certain sub-behaviour with only minor changes to the original agent program. I call this technique *adjustable deliberation*.

After a brief overview of the framework of *Behavioural State Machines* (*BSM*) with the associated programming language *Jazzyk* in Section 2, sections 3 and 4 introduce *P-BSM* and *Jazzyk(P)*, their respective probabilistic extensions. Section 5 discusses practical use of the *P-BSM* framework together with a brief overview of related work. Finally, a summary with final remarks concludes the paper in Section 6.

2 Behavioural State Machines

In [10] I introduced the framework of *Behavioural State Machines*. *BSM* framework draws a clear distinction between the *knowledge representation* and *behavioural* layers within an agent. It thus provides a programming framework that clearly separates the programming concerns of *how to represent an agent's knowledge* about, for example, its environment and *how to encode its behaviours* for acting in it. This section briefly introduces the *BSM* framework, for simplic-

ity without treatment of variables. For the complete formal description of the *BSM* framework, see [10].

2.1 Syntax

BSM agents are collections of one or more so-called *knowledge representation modules* (KR modules), typically denoted by $\mathcal{M}$, each representing a part of the agent's knowledge base. KR modules may be used to represent and maintain various mental attitudes of an agent, such as knowledge about its environment, or its goals, intentions, obligations, etc. Transitions between states of a *BSM* result from applying so-called *mental state transformers* (*mst*), typically denoted by τ. Various types of mst's determine the behaviour that an agent can generate. A *BSM agent* consists of a set of KR modules $\mathcal{M}_1, \ldots, \mathcal{M}_n$ and a mental state transformer $\mathcal{P}$, i.e. $\mathcal{A} = (\mathcal{M}_1, \ldots, \mathcal{M}_n, \mathcal{P})$; the mst $\mathcal{P}$ is also called an *agent program*.

The notion of a KR module is an abstraction of a partial knowledge base of an agent. In turn, its states are to be treated as theories (i.e., sets of sentences) expressed in the KR language of the module. Formally, a KR module $\mathcal{M}_i = (\mathcal{S}_i, \mathcal{L}_i, \mathcal{Q}_i, \mathcal{U}_i)$ is characterised by a knowledge representation language $\mathcal{L}_i$, a set of states $\mathcal{S}_i \subseteq 2^{\mathcal{L}_i}$, a set of query operators $\mathcal{Q}_i$ and a set of update operators $\mathcal{U}_i$. A query operator $\models \in \mathcal{Q}_i$ is a mapping $\models : \mathcal{S}_i \times \mathcal{L}_i \to \{\top, \bot\}$. Similarly an update operator $\oplus \in \mathcal{U}_i$ is a mapping $\oplus : \mathcal{S}_i \times \mathcal{L}_i \to \mathcal{S}_i$.

Queries, typically denoted by φ, can be seen as operators of type $\models : \mathcal{S}_i \to \{\top, \bot\}$. A primitive query $\varphi = (\models\phi)$ consists of a query operator $\models \in \mathcal{Q}_i$ and a formula $\phi \in \mathcal{L}_i$ of the same KR module $\mathcal{M}_i$. Complex queries can be composed by means of conjunction $\wedge$, disjunction $\vee$ and negation $\neg$.

Mental state transformers enable transitions from one state to another. A primitive mst $\oslash\psi$, typically denoted by ρ and constructed from an update operator $\oslash \in \mathcal{U}_i$ and a formula $\psi \in \mathcal{L}_i$, refers to an update on the state of the corresponding KR module. Conditional mst's are of the form $\varphi \longrightarrow \tau$, where φ is a query and τ is a mst. Such a conditional mst makes the application of τ depend on the evaluation of φ. Syntactic constructs for combining mst's are: non-deterministic choice | and sequence $\circ$.

Definition 1 (mental state transformer). *Let $\mathcal{M}_1, \ldots, \mathcal{M}_n$ be KR modules of the form $\mathcal{M}_i = (\mathcal{S}_i, \mathcal{L}_i, \mathcal{Q}_i, \mathcal{U}_i)$. The set of* mental state transformers *is defined as below:*

- **skip** *is a* primitive *mst,*
- *if $\oslash \in \mathcal{U}_i$ and $\psi \in \mathcal{L}_i$, then $\oslash\psi$ is a* primitive *mst,*
- *if φ is a query, and τ is a mst, then $\varphi \longrightarrow \tau$ is a* conditional *mst,*
- *if τ and τ' are mst's, then $\tau|\tau'$ and $\tau \circ \tau'$ are mst's (*choice*, and* sequence *respectively).*

2.2 Semantics

The *yields* calculus, summarised below after [10], specifies an update associated with executing a mental state transformer in a single step of the language inter-

preter. It formally defines the meaning of the state transformation induced by executing an mst in a state, i.e., a mental state transition.

Formally, a *mental state* σ of a *BSM* $\mathcal{A} = (\mathcal{M}_1, \ldots, \mathcal{M}_n, \tau)$ is a tuple $\sigma = \langle \sigma_1, \ldots, \sigma_n \rangle$ of KR module states $\sigma_1 \in \mathcal{S}_1, \ldots, \sigma_n \in \mathcal{S}_n$, corresponding to $\mathcal{M}_1, \ldots, \mathcal{M}_n$ respectively. $\mathcal{S} = \mathcal{S}_1 \times \cdots \times \mathcal{S}_n$ denotes the space of all mental states over $\mathcal{A}$. A mental state can be modified by applying primitive mst's on it and query formulae can be evaluated against it. The semantic notion of truth of a query is defined through the satisfaction relation $\models$. A primitive query $\vDash\phi$ holds in a mental state $\sigma = \langle \sigma_1, \ldots, \sigma_n \rangle$ (written $\sigma \models (\vDash\phi)$) iff $\vDash(\phi, \sigma_i)$, otherwise we have $\sigma \not\models (\vDash\phi)$. Given the usual meaning of Boolean operators, it is straightforward to extend the query evaluation to compound query formulae. Note that evaluation of a query does not change the mental state σ.

For an mst $\oslash\psi$, we use $(\oslash, \psi)$ to denote its semantic counterpart, i.e., the corresponding *update* (state transformation). Sequential application of updates is denoted by $\bullet$, i.e. $\rho_1 \bullet \rho_2$ is an update resulting from applying ρ_1 first and then applying ρ_2. The application of an update to a mental state is defined formally below.

Definition 2 (applying an update). *The result of applying an update* $\rho = (\oslash, \psi)$ *to a state* $\sigma = \langle \sigma_1, \ldots, \sigma_n \rangle$ *of a* BSM $\mathcal{A} = (\mathcal{M}_1, \ldots, \mathcal{M}_n, \mathcal{P})$*, denoted by* $s \bigoplus \rho$*, is a new state* $\sigma' = \langle \sigma_1, \ldots, \sigma'_i, \ldots, \sigma_n \rangle$*, where* $\sigma'_i = \sigma_i \oslash \psi$ *and* σ_i*,* $\oslash$*, and* ψ *correspond to one and the same* $\mathcal{M}_i$ *of* $\mathcal{A}$*. Applying the empty update* **skip** *on the state* σ *does not change the state, i.e.* $\sigma \bigoplus \mathbf{skip} = \sigma$*.*

Inductively, the result of applying a sequence of updates $\rho_1 \bullet \rho_2$ *is a new state* $\sigma'' = \sigma' \bigoplus \rho_2$*, where* $\sigma' = \sigma \bigoplus \rho_1$*.* $\sigma \overset{\rho_1 \bullet \rho_2}{\rightarrow} \sigma'' = \sigma \overset{\rho_1}{\rightarrow} \sigma' \overset{\rho_2}{\rightarrow} \sigma''$ *denotes the corresponding compound transition.*

The meaning of a mental state transformer in state σ, formally defined by the *yields* predicate below, is the update set it yields in that mental state.

Definition 3 (yields calculus). *A mental state transformer* τ *yields an* update ρ *in a state* σ*, iff* $yields(\tau, \sigma, \rho)$ *is derivable in the following calculus:*

$$\frac{\top}{yields(\mathbf{skip}, \sigma, \mathbf{skip})} \quad \frac{\top}{yields(\oslash\psi, \sigma, (\oslash, \psi))} \qquad \textit{(primitive)}$$

$$\frac{yields(\tau, \sigma, \rho), \sigma \models \phi}{yields(\phi \longrightarrow \tau, \sigma, \rho)} \quad \frac{yields(\tau, \sigma, \rho), \sigma \not\models \phi}{yields(\phi \longrightarrow \tau, \sigma, \mathbf{skip})} \qquad \textit{(conditional)}$$

$$\frac{yields(\tau_1, \sigma, \rho_1),\ yields(\tau_2, \sigma, \rho_2)}{yields(\tau_1 | \tau_2, \sigma, \rho_1),\ yields(\tau_1 | \tau_2, \sigma, \rho_2)} \qquad \textit{(choice)}$$

$$\frac{yields(\tau_1, \sigma, \rho_1),\ yields(\tau_2, \sigma \bigoplus \rho_1, \rho_2)}{yields(\tau_1 \circ \tau_2, \sigma, \rho_1 \bullet \rho_2)} \qquad \textit{(sequence)}$$

We say that τ *yields an* update set ν *in a state* σ *iff* $\nu = \{\rho | yields(\tau, \sigma, \rho)\}$*.*

The mst **skip** yields the update **skip**. Similarly, a primitive update mst $\oslash\psi$ yields the corresponding update $(\oslash, \psi)$. In the case the condition of a conditional mst $\phi \longrightarrow \tau$ is satisfied in the current mental state, the calculus yields one of the updates corresponding to the right hand side mst τ, otherwise the no-operation

skip update is yielded. A non-deterministic choice mst yields an update corresponding to either of its members and finally a sequential mst yields a sequence of updates corresponding to the first mst of the sequence and an update yielded by the second member of the sequence in a state resulting from application of the first update to the current mental state.

Notice, that the provided semantics of choice and sequence operators implies associativity of both. Hence, from this point on, instead of the strictly pairwise notation $\tau_1|(\tau_2|(\tau_3|(\cdots|\tau_k)))$, we simply write $\tau_1|\tau_2|\tau_2|\cdots|\tau_k$. Similarly for the sequence operation $\circ$.

The following definition articulates the denotational semantics of the notion of mental state transformer as an encoding of a function mapping mental states of a *BSM* to updates, i.e., transitions between them.

Definition 4 (mst functional semantics). *Let $\mathcal{M}_1, \ldots, \mathcal{M}_n$ be KR modules. A mental state transformer τ encodes a function $\mathfrak{f}_\tau : \sigma \mapsto \{\rho | yields(\tau, \sigma, \rho)\}$ over the space of mental states $\sigma = \langle \sigma_1, \ldots, \sigma_n \rangle \in S_1 \times \cdots \times S_n$.*

Subsequently, the semantics of a *BSM* agent is defined as a set of traces in the induced transition system enabled by the *BSM* agent program.

Definition 5 (BSM semantics). *A* BSM $\mathcal{A} = (\mathcal{M}_1, \ldots, \mathcal{M}_n, \mathcal{P})$ *can make a step from state σ to a state σ', iff $\sigma' = \sigma \bigoplus \rho$, s.t. $\rho \in \mathfrak{f}_\mathcal{P}(\sigma)$. We also say, that* $\mathcal{A}$ induces *a (possibly compound) transition $\sigma \xrightarrow{\rho} \sigma'$.*

A possibly infinite sequence of states $\sigma_1, \ldots, \sigma_i, \ldots$ is a run *of BSM $\mathcal{A}$, iff for each $i \geq 1$, $\mathcal{A}$ induces a transition $\sigma_i \rightarrow \sigma_{i+1}$.*

The semantics of an agent system characterised by a BSM $\mathcal{A}$, is a set of all runs of $\mathcal{A}$.

Additionally, we require the non-deterministic choice of a *BSM* interpreter to fulfil the *weak fairness condition*, similar to that in [7], for all the induced runs.

Condition 1 (weak fairness condition) *A computation run is weakly fair iff it is not the case that an update is always yielded from some point in time on but is never selected for execution.*

2.3 Jazzyk

Jazzyk is an interpreter of the *Jazzyk* programming language implementing the computational model of the *BSM* framework. Later in this paper, we use a more readable notation mixing the syntax of *Jazzyk* with that of the *BSM* mst's introduced above. when ϕ then τ encodes a conditional mst $\phi \longrightarrow \tau$. Symbols ; and , stand for choice | and sequence $\circ$ operators respectively. To facilitate operator precedence, mental state transformers can be grouped into compound structures, blocks, using curly braces {...}.

To better support source code modularity and re-usability, *Jazzyk* interpreter integrates a macro preprocessor, a powerful tool for structuring and modularising and encapsulating the source code and writing code templates.

For further details on the *Jazzyk* programming language and the macro preprocessor integration with *Jazzyk* interpreter, consult [10].

3 Probabilistic BSMs

In the plain *BSM* framework, the syntactic construct of a mental state transformer encodes a transition function over the space of mental states of a *BSM* (cf. Definition 4). Hence, an execution of a compound non-deterministic choice mst amounts to a non-deterministic selection of one of its components and its subsequent application to the current mental state of the agent. In order to enable a finer grained control over this selection process, in this section I introduce an extension of the *BSM* framework with specifications of a probability distributions over components of choice mst's.

The *P-BSM* formalism introduced below heavily builds on associativeness of *BSM* composition operators of non-deterministic choice and sequence. We also informally say that an mst τ *occurs* in a mst τ' iff τ' can be constructed from a set of mst's $\mathcal{T}$, s.t. $\tau \in \mathcal{T}$, by using composition operators as defined by the Definition 1.

Definition 6 (Probabilistic BSM). *A* Probabilistic Behavioural State Machine *(*P-BSM*)* $\mathcal{A}_p$ *is a tuple* $\mathcal{A}_p = (\mathcal{M}_1, \ldots, \mathcal{M}_n, \mathcal{P}, \Pi)$*, where* $\mathcal{A} = (\mathcal{M}_1, \ldots, \mathcal{M}_n, \mathcal{P})$ *is a* BSM *and* $\Pi : \tau \mapsto P_\tau$ *is a function assigning to each non-deterministic choice mst of the form* $\tau = \tau_1 | \cdots | \tau_k \in \mathcal{P}$ *occurring in* $\mathcal{P}$ *a discrete probability distribution function* $P_\tau{:}\tau_i \mapsto [0, 1]$*, s.t.* $\sum_{i=1}^{k} P_\tau(\tau_i) = 1$.

W.l.o.g. we assume that each mst occurring in the agent program $\mathcal{P}$ *can be uniquely identified (e.g. by its position in the agent program).*

The probability distribution function P_τ assigns to each component of a non-deterministic choice mst $\tau = \tau_1 | \tau_2 | \cdots | \tau_k$ a probability of its selection for application by a *BSM* interpreter.

Note, that because of the unique identification of mst's in an agent program $\mathcal{P}$, the function Π assigns two distinct discrete probability distributions P_{τ_1} and P_{τ_2} to choice mst's τ_1, τ_2 even when they share the syntactic form but occur as distinct components of $\mathcal{P}$.

To distinguish from the *BSM* formalism, we call mst's occurring in a *P-BSM* *probabilistic mental state transformers*. *BSM* mst's as defined in Section 2 will be called *plain*.

Similarly to plain mst's, the semantic counterpart of a probabilistic mst is a probabilistic update. A *probabilistic update* of a *P-BSM* $\mathcal{A}_p = (\mathcal{M}_1, \ldots, \mathcal{M}_n, \mathcal{P}, \Pi)$ is a tuple $p{:}\rho$, where $p \in \mathbb{R}$, s.t. $p \in [0, 1]$, is a probability and $\rho = (\oslash, \psi)$ is an update from the *BSM* $\mathcal{A} = (\mathcal{M}_1, \ldots, \mathcal{M}_n, \mathcal{P})$.

The semantics of a probabilistic mental state transformer in a state σ, formally defined by the $yields_p$ predicate below, is the probabilistic update set it yields in that mental state.

Definition 7 ($\mathbf{yields}_p$ calculus). *A probabilistic mental state transformer* τ *yields a probabilistic update* $p{:}\rho$ *in a state* σ*, iff* $yields_p(\tau, \sigma, p{:}\rho)$ *is derivable in the following calculus:*

$$\frac{\top}{yields_p(\mathbf{skip},\sigma,1{:}\mathbf{skip})} \qquad \frac{\top}{yields_p(\oslash\psi,\sigma,1{:}(\oslash,\psi))} \qquad \textit{(primitive)}$$

$$\frac{yields_p(\tau,\sigma,p{:}\rho),\, \sigma\models\phi}{yields_p(\phi\longrightarrow\tau,\sigma,p{:}\rho)} \qquad \frac{yields_p(\tau,\sigma,\theta,p{:}\rho),\, \sigma\not\models\phi}{yields_p(\phi\longrightarrow\tau_p,\sigma,1{:}\mathbf{skip})} \qquad \textit{(conditional)}$$

$$\frac{\tau{=}\tau_1|\cdots|\tau_k,\, \Pi(\tau){=}P_\tau,\, \forall 1{\leq}i{\leq}k{:}\; yields_p(\tau_i,\sigma,p_i{:}\rho_i)}{\forall 1{\leq}i{\leq}k{:}\; yields_p(\tau,\sigma,P_\tau(\tau_i){\cdot}p_i{:}\rho_i)} \qquad \textit{(choice)}$$

$$\frac{\tau{=}\tau_1\circ\cdots\circ\tau_k,\, \forall 1{\leq}i{\leq}k{:}\; yields_p(\tau_i,\sigma_i,p_i{:}\rho_i)\wedge\sigma_{i+1}{=}\sigma_i\oplus\rho_i}{yields(\tau,\sigma_1,\prod_{i=1}^{k}p_i{:}\rho_1\bullet\cdots\bullet\rho_k)} \qquad \textit{(sequence)}$$

The modification of the plain *BSM yields* calculus introduced above for primitive and conditional mst's is rather straightforward. A plain primitive mst yields the associated primitive update for which there's no probability of execution specified. A conditional mst yields probabilistic updates of its right hand side if the left hand side query condition is satisfied. It amounts to a **skip** mst otherwise. The function Π associates a discrete probability distribution function with each non-deterministic choice mst and thus modifies the probability of application of the probabilistic updates yielded by its components accordingly. Finally, similarly to the plain *yields* calculus, a sequence of probabilistic mst's yields sequences of updates of its components, however the joint application probability equals to the conditional probability of selecting the particular sequence of updates. The following example illustrates the sequence rule of the probabilistic $yields_p$ calculus.

Example 1. Consider the following mst: $(0.3{:}\tau_1 \,|\, 0.7{:}\tau_2) \circ (0.6{:}\tau_3 \,|\, 0.4{:}\tau_4)$. Let's assume that for each of the component mst's τ_i, we have $yields_p(\tau_i, \sigma, p_i{:}\rho_i)$ in a state σ. The plain *yields* calculus yields the following sequences of updates $\rho_1 \bullet \rho_3$, $\rho_1 \bullet \rho_4$, $\rho_2 \bullet \rho_3$ and $\rho_2 \bullet \rho_4$. The probability of selection of each of them, however, equals to the conditional probability of choosing an update from the second component of the sequence, provided that the choice from the first one was already made. I.e. the probabilistic $yields_p$ calculus results in the following sequences of probabilistic updates $0.18{:}(\rho_1 \bullet \rho_3)$, $0.12{:}(\rho_1 \bullet \rho_4)$, $0.42{:}(\rho_2 \bullet \rho_3)$ and $0.28{:}(\rho_2 \bullet \rho_4)$.

The corresponding adaptation of the mst functional semantics straightforwardly follows.

Definition 8 (probabilistic mst functional semantics). *Let* $\mathcal{A}_p = (\mathcal{M}_1, \ldots, \mathcal{M}_n, \mathcal{P}, \Pi)$ *be a* P-BSM. *A probabilistic mental state transformer* τ *encodes a transition function* $\mathfrak{fp}_\tau : \sigma \mapsto \{p : \rho | yields_p(\tau, \sigma, p : \rho)\}$ *over the space of mental states* $\sigma = \langle \sigma_1, \ldots, \sigma_n \rangle \in S_1 \times \cdots \times S_n$.

According to the Definition 6, each mst occurring in a *P-BSM* agent program can be uniquely identified. Consequently, also each probabilistic update yielded by the program can be uniquely identified by the mst it corresponds to. The consequence is, that w.l.o.g. we can assume that even when two probabilistic updates $p_1{:}\rho_1$, $p_2{:}\rho_2$ yielded by the agent program $\mathcal{P}$ in a state σ share their syntactic form (i.e. $p_1 = p_2$ and ρ_1, ρ_2 encode the same plain *BSM* update) they both independently occur in the probabilistic update set $\mathfrak{fp}(\sigma)$.

The following lemma shows, that the semantics of probabilistic mst's embodied by the $yields_p$ calculus can be understood as *an encoding of a probability distribution*, or *a probabilistic policy* over updates yielded by the underlying plain mst. Moreover, it also implies that composition of probabilistic mst's maintains their nature as probability distributions.

Lemma 1. *Let $\mathcal{A}_p = (\mathcal{M}_1, \ldots, \mathcal{M}_n, \mathcal{P}, \Pi)$ be a* P-BSM. *For every mental state transformer τ occurring in $\mathcal{P}$ and a mental state σ of $\mathcal{A}_p$, we have*

$$\sum_{p:\rho \in \mathfrak{fp}_\tau(\sigma)} p = 1 \tag{1}$$

Proof. Cf. Appendix A.

Finally, the semantics of a *P-BSM* agent is defined as a set of traces in the induced transition system enabled by the *P-BSM* agent program.

Definition 9 (BSM semantics). *A* P-BSM $\mathcal{A}_p = (\mathcal{M}_1, \ldots, \mathcal{M}_n, \mathcal{P}, \Pi)$ *can make a step from state σ to a state σ' with probability p, iff $\sigma' = \sigma \bigoplus \rho$, s.t. $p{:}\rho \in \mathfrak{fp}_\tau(\sigma)$. We also say, that with a probability p, $\mathcal{A}_p$ induces a (possibly compound) transition $\sigma \xrightarrow{p:\rho} \sigma'$.*

A possibly infinite sequence of states $\omega = \sigma_1, \ldots, \sigma_i, \ldots$ is a run of P-BSM *$\mathcal{A}_p$, iff for each $i \geq 1$, $\mathcal{A}$ induces the transition $\sigma_i \xrightarrow{p_i:\rho_i} \sigma_{i+1}$ with probability p_i.*

Let $pref(\omega)$ denote the set of all finite prefixes of a possibly infinite computation run ω and $|.|$ the length of a finite run. $P(\omega) = \prod_{i=1}^{|\omega|} p_i$ is then the probability of the finite run ω.

The semantics of an agent system characterised by a P-BSM *$\mathcal{A}_p$, is a set of all runs ω of $\mathcal{A}_p$, s.t. all of their finite prefixes $\omega' \in pref(\omega)$ have probability $P(\omega') > 0$.*

Informally, the semantics of an agent system is a set of runs involving only transitions induced by updates with a non-zero selection probability.

Additionally, we require an admissible *P-BSM* interpreter to fulfil the following specialisation of the weak fairness condition, for all the induced runs.

Condition 2 (P-BSM weak fairness condition) *Let ω be a possibly infinite computation run of a* P-BSM *$\mathcal{A}_p$. Let also $freq_{p:\rho}(\omega')$ be the number of transitions induced by the update $p{:}\rho$ along a finite prefix of $\omega' \in pref(\omega)$.*

We say that ω is weakly fair w.r.t. *$\mathcal{A}_p$ iff for all updates $p{:}\rho$ we have, that if from some point on $p{:}\rho$ is always yielded in states along ω, 1) it also occurs on ω infinitely often, and 2) for the sequence of finite prefixes of ω ordered according to their length holds*

$$\liminf_{\substack{|\omega'| \to \infty \\ \omega' \in pref(\omega)}} \frac{freq_{p:\rho}(\omega')}{|\omega'|} \geq p$$

Similarly to the plain *BSM* weak fairness Condition 1, the above stated Condition 2 embodies a minimal requirement on admissible *P-BSM* interpreters. It admits only *P-BSM* interpreters which honor the intended probabilistic semantics of the non-deterministic choice selection of the $yields_p$ calculus. The first part of the requirement is a consequence of the plain *BSM* weak fairness condition (Condition 1), while the second states that in sufficiently long computation runs, the frequency of occurrence of an always yielded probabilistic update corresponds to its selection probability in each single step.

4 Jazzyk(P)

Jazzyk is a programming language instantiating the plain *BSM* theoretical framework introduced in [10]. This section informally describes its extension *Jazzyk(P)*, an instantiation of the framework of *Probabilistic Behavioural State Machines* introduced in Section 3 above.

Jazzyk(P) syntax differs from that of *Jazzyk* only in specification of probability distributions over choice mst's. *Jazzyk(P)* allows for explicit labellings of choice mst members by their individual application probabilities. Consider the following labelled choice mst $p_1{:}\tau_1\,;\, p_2{:}\tau_2\,;\, p_3{:}\tau_3\,;\, p_4{:}\tau_4$ in the *Jazzyk(P)* notation. Each $p_i \in [0,1]$ denotes the probability of selection of mst τ_i by the interpreter. Furthermore, to ensure that the labelling denotes a probability distribution over τ_i's, *Jazzyk(P)* parser requires that $\sum_{i=1}^{k} p_i = 1$ for every choice mst $p_1{:}\tau_1\,;\ldots;\, p_k{:}\tau_k$ occurring in the considered agent program. Similarly to *Jazzyk*, during the program interpretation phase, *Jazzyk(P)* interpreter proceeds in a top-down manner subsequently considering nested mst's from the main agent program, finally down to primitive update formulae. When the original *Jazzyk* interpreter faces a selection from a non-deterministic choice mst, it randomly selects one of them assuming a discrete uniform probability distribution. I.e., the probability of selecting from a choice mst with k members is $\frac{1}{k}$ for each of them. The extended interpreter *Jazzyk(P)* respects the specified selection probabilities: it generates a random number $p \in [0,1]$ and selects τ_s, s.t. $\sum_{i=1}^{s-1} p_i \leq p \leq \sum_{i=1}^{s} p_i$.

For convenience, *Jazzyk(P)* enables use of incomplete labellings. An *incompletely labelled* non-deterministic choice mst is one containing at least one member mst without an explicit probability specification such as $p_1{:}\tau_1\,;\, p_2{:}\tau_2\,;\, \tau_3\,;\, \tau_4$. In such a case, the *Jazzyk(P)* parser automatically completes the distribution by uniformly dividing the remaining probability range to unlabelled mst's. I.e., provided an incompletely labelled choice mst with k members, out of which $m < k$ are labelled $(p_1{:}\tau_1\,;\ldots;\, p_m{:}\tau_m\,;\, \tau_{m+1}\,;\cdots;\, \tau_k)$, it assigns probability $p = \frac{1-\sum_{i=1}^{m} p_i}{k-m}$ to the remaining mst's $\tau_{m+1},\ldots,\tau_k$.

The Listing 1 provides an example of a *Jazzyk(P)* code snippet adapted from the *Jazzbot* project [6]. Consider a BDI-style virtual agent (bot) in a simulated 3D environment. The bot moves around a virtual building and searches for items which it picks up and delivers to a particular place in the environment. Upon encountering an unfriendly agent (attacker), it executes an emergency behaviour,

Listing 1 Example of *Jazzyk(P)* syntax.

```
when ⊨bel [{ threatened }] then {
        /* ***Emergency modus operandi*** */

        /* Detect the enemy's position */
    0.7 : when ⊨bel [{ attacker(Id) }] and ⊨env [{ eye see Id player Pos }]
    then ⊕map [{ positions[Id] = Pos }] ;

    /* Check the camera sensor */
    0.2 : when ⊨env [{ eye see Id Type Pos }] then {
        ⊕bel [{ see(Id, Type) }] ,
        ⊕map [{ objects[Pos].addIfNotPresent(Id) }]
    }

    /* Check the body health sensor */
    when ⊨env [{ body health X }] then ⊕bel [{ health(X). }] ;
} else {
    /* ***Normal mode of perception*** */

    /* Check the body health sensor */
    when ⊨env [{ body health X }] then ⊕bel [{ health(X). }] ;

    /* Check the camera sensor */
    when ⊨env [{ eye see Id Type Pos }] then {
        ⊕bel [{ see(Id, Type) }] ,
        ⊕map [{ positions[Id] = Pos }]
    }
}
```

such as running away until it feels safe again. The agent consists of several KR modules *bel*, *map* and *env* respectively representing its beliefs about the environment and itself, the map of the environment and an interface to its sensors and actuators, i.e. the body. The corresponding query and update operators $\models$ and $\oplus$ are sub-scripted with the KR module label they correspond to.

The Listing 1 provides a piece of code for perception of the bot. In the normal mode of operation, the bot in a single step queries either its camera, or its body health status sensor with the same probability of selection for each of them, i.e., 0.5. However, in the case of emergency, the bot focuses more on escaping the attacker, therefore, in order to retrieve the attacker's position, it queries the camera sensor more often (selection probability $p = 0.7$) than sensing objects around it ($p = 0.2$). Checking it's own body health is of the least importance ($p = 0.1$), however not completely negligible.

In an implemented program, however, the Listing 1 would be rewritten using the macro facility of the *Jazzyk* interpreter and reduced to a more concise code shown in the Listing 2.

5 Discussion

Probabilistic Behavioural State Machines, and in turn *Jazzyk(P)*, allow for labelling of alternatives in non-deterministic choice mental state transformers, thus providing a specification of a probability distribution over the set of enabled transitions for the next step in agent's life-cycle. Besides the, in the field

Listing 2 Example of focusing bot's attention during emergency situations rewritten with reusable macros.

```
when ⊨bel [{ threatened }] then {
    /* ***Emergency modus operandi*** */
    0.7 : DETECT_ENEMY_POSITION ;
    0.2 : SENSE_CAMERA ;
          SENSE_HEALTH
} else {
    /* ***Normal mode of perception*** */
    SENSE_HEALTH ;
    SENSE_CAMERA
}
```

of rule based agent programming languages conventional, Plotkin style operational semantics [13], the *BSM* semantics allows a functional view on mental state transformers (cf. Definition 8). In turn, the informal reading of a *P-BSM* choice mst's can be seen as a specification of the probability with which the next transition will be chosen from the function denoted by the particular member mst. In other words, *a probability of applying the member mst function to the current mental state*.

The proposed extension allows a finer grained steering of the interpreter's non-deterministic selection mechanism and has applications across several niches of methodologies for rule based agent oriented programming languages. Our analyses and first experiments in the context of the *Jazzbot* application have shown that labelling probabilistic mst's is a useful means to contextually focus agent's perception (cf. Listing 2). Similarly, the labelling technique is useful in contexts, when it is necessary to execute certain behaviours with a certain given frequency. For example, approximatelly in about every 5th step broadcast a ping message to peers of the agent's team. Finally, the technique can be used when the agent developer has an informal intuition that preferring more frequent execution of certain behaviours over others (e.g. cheaper, but less efficient over resource intensive, but rather powerful) might suffice, or even perform better in a given context. Of course writing such programs makes sense only when a rigorous analysis of the situation is impossible, or undesirable and at the same time a suboptimal performance of the agent system is acceptable.

An instance of the latter technique for modifying the main control cycle of the agent program is what I call *adjustable deliberation*. Consider the following *Jazzyk BSM* code for the main control cycle of an agent adapted from [12]:

```
PERCEIVE ; HANDLE_GOALS ; ACT
```

The macros PERCEIVE, HANDLE_GOALS and ACT encode behaviours for perception (similar to that in the Listing 1), goal commitment strategy implementation and action selection respectively. In the case of emergency, as described in the example in Section 4 above, it might be useful to slightly neglect deliberation about agent's goals, in favour of an intensive environment observation and quick reaction selection. The following reformulation of the agent's control cycle demonstrates the simple program modification:

```
when ⊨bel [{ emergency }] then { PERCEIVE ; HANDLE_GOALS ; ACT }
                          else { 0.4 : PERCEIVE ; HANDLE_GOALS ; 0.4 : ACT }
```

The underlying semantic model of *Behavioural State Machines* framework is a labelled transition system [11]. In consequence, the underlying semantic model of the *P-BSM* framework is a discrete probabilistic labelled transition system, i.e., a structure similar to a *Markov chain* [8]. This similarity suggest a relationship of the *P-BSM* underlying semantic structure to various types of *Markov models* (cf. e.g. [9]), however a more extensive exploration of this relationship is beyond the scope of this paper.

In the field of agent oriented programming languages, recently Hindriks et al. [5] introduced an extension of the *GOAL* language [4], where a quantitative numeric value is associated with execution of an action leading from a mental state m to another mental state m'. I.e., a triple of a precondition ϕ (partially describing m), an action a and a post-condition ψ (describing m') is labelled with a utility value $U(\phi, a, \psi)$. Subsequently, in each deliberation cycle, the interpreter selects the action with the highest expected future utility w.r.t. agent's goals.

The approach of Hindriks et al. focuses on estimating aggregate utility values of bounded future evolutions of the agent system, i.e., evaluating possible future courses of evolution, plans, the agent can consider, and subsequently choosing an action advancing the system evolution along the best path. The *P-BSM*, on the other hand, is concerned only with selection of the next action resulting from the bottom-up propagation of probabilistic choices through the nested structure, a decision tree, of the agent program. So, while the approach of Hindriks et al. can be seen as a step towards look-ahead like reactive planning, *P-BSM* remains a purely reactive approach to programming cognitive agents. Informally, except for the nested structuring of agent programs (the distinguishing practical feature of the *BSM* framework w.r.t. to other theoretically founded agent programming languages), the *P-BSM* framework could be emulated by the approach of Hindriks et al. with the look-ahead planning bound of the length one.

6 Conclusion

The main contribution of the presented paper is introduction of *Probabilistic Behavioural State Machines* framework with the associated agent programming language *Jazzyk(P)*. The proposed extension of the plain *BSM* framework is a result of practical experience with *BSM* case-studies [6] and introduces a straightforward and pragmatic, yet quite a powerful, extension of the *BSM* framework. However, the presented paper presents only first steps towards a more rigorous approach to dealing with underspecification in agent oriented programming by means of probabilistic action selection.

Underspecification of agent programs is in general inevitable. However, in situations when a suboptimal performance is tolerable, providing the agent program interpreter with a heuristics for steering its choices can lead to rapid development of more efficient and robust agent systems.

References

1. Rafael H. Bordini, Mehdi Dastani, Jürgen Dix, and Amal El Fallah Seghrouchni. *Multi-Agent Programming Languages, Platforms and Applications*, volume 15 of *Multiagent Systems, Artificial Societies, and Simulated Organizations*. Kluwer Academic Publishers, 2005.
2. Rafael H. Bordini, Jomi F. Hübner, and Renata Vieira. *Jason and the Golden Fleece of Agent-Oriented Programming*, chapter 1, pages 3–37. Volume 15 of *Multiagent Systems, Artificial Societies, and Simulated Organizations* [1], 2005.
3. Mehdi Dastani, M. Birna van Riemsdijk, and John-Jules Meyer. *Programming Multi-Agent Systems in 3APL*, chapter 2, pages 39–68. Volume 15 of *Multiagent Systems, Artificial Societies, and Simulated Organizations* [1], 2005.
4. Frank S. de Boer, Koen V. Hindriks, Wiebe van der Hoek, and John-Jules Ch. Meyer. A verification framework for agent programming with declarative goals. *J. Applied Logic*, 5(2):277–302, 2007.
5. Koen V. Hindriks, Catholijn M. Jonker, and Wouter Pasman. Exploring heuristic action selection in agent programming. In Koen Hindriks, Alexander Pokahr, and Sebastian Sardina, editors, *Proceedings of the Sixth International Workshop on Programming Multi-Agent Systems, ProMAS'08, Estoril, Portugal*, volume 5442 of *LNAI*, 2008.
6. Michael Köster, Peter Novák, David Mainzer, and Bernd Fuhrmann. Two case studies for Jazzyk BSM. In *Proceedings of Agents for Games and Simulations, AGS 2009, AAMAS 2009 collocated workshop, to appear*, 2009.
7. Zohar Manna and Amir Pnueli. *The temporal logic of reactive and concurrent systems*. Springer-Verlag New York, Inc., New York, NY, USA, 1992.
8. Andrey Andreyevich Markov. Extension of the law of large numbers to dependent quantities (in Russian). *Izvestiya Fiziko-matematicheskogo obschestva pri Kazanskom Universitete*, (15):135–156, 1906.
9. S. P. Meyn and R. L. Tweedie. *Markov Chains and Stochastic Stability*. Springer-Verlag, London, 1993.
10. Peter Novák. Jazzyk: A programming language for hybrid agents with heterogeneous knowledge representations. In *Proceedings of the Sixth International Workshop on Programming Multi-Agent Systems, ProMAS'08*, volume 5442 of *LNAI*, pages 72–87, May 2008.
11. Peter Novák and Wojciech Jamroga. Code patterns for agent-oriented programming. In *Proceedings of The Eighth International Conference on Autonomous Agents and Multi-Agent Systems, AAMAS 2009, to appear*, 2009.
12. Peter Novák and Michael Köster. Designing goal-oriented reactive behaviours. In *Proceedings of the 6th International Cognitive Robotics Workshop, CogRob 2008, July 21-22 in Patras, Greece*, pages 24–31, July 2008.
13. Gordon D. Plotkin. A Structural Approach to Operational Semantics. Technical Report DAIMI FN-19, University of Aarhus, 1981.
14. Alexander Pokahr, Lars Braubach, and Winfried Lamersdorf. *Jadex: A BDI Reasoning Engine*, chapter 6, pages 149–174. Volume 15 of *Multiagent Systems, Artificial Societies, and Simulated Organizations* [1], 2005.
15. Anand S. Rao and Michael P. Georgeff. An Abstract Architecture for Rational Agents. In *KR*, pages 439–449, 1992.
16. Michael Winikoff. *JACK^{TM} Intelligent Agents: An Industrial Strength Platform*, chapter 7, pages 175–193. Volume 15 of *Multiagent Systems, Artificial Societies, and Simulated Organizations* [1], 2005.

A Proofs

Proof (Proof of Lemma 1). The proof follows by induction on nesting depth of mst's. The nesting depth of an mst is the maximal number of steps required to derive $\mathit{yields}_p(\tau, \sigma, p{:}\rho)$ in the yields_p calculus for all σ from $\mathcal{A}_p$ and all $p{:}\rho$ yielded by τ.

depth $= 1$**:** Equation 1 is trivially satisfied for primitive updates from $\mathcal{A}_p$ of the form **skip** and $\oslash\psi$.

depth $= 2$**:** let's assume $\tau_1, \ldots, \tau_k$ are primitive mst's yielding $1{:}\rho_1, \ldots, 1{:}\rho_k$ in a state σ respectively, and ϕ be a query formula. We recognise three cases:

conditional in the case $\sigma \models \phi$, we have $\mathit{yields}_p(\phi \longrightarrow \tau_1, \sigma, 1{:}\rho_1)$. Similarly for $\sigma \not\models \phi$, we have $\mathit{yields}_p(\phi \longrightarrow \tau_1, \sigma, 1{:}\mathbf{skip})$, hence Equation 1 is satisfied in both cases.

choice according to Definition 7 for each $1 \leq i \leq k$ we have

$$\mathit{yields}_p(\tau_1 | \cdots | \tau_k, \sigma, P_{\tau_1|\cdots|\tau_k}(\tau_i){:}\rho_i)$$

where $\mathit{\Pi}(\tau_1 | \cdots | \tau_k) = P_{\tau_1|\cdots|\tau_k}$. Since $P_{\tau_1|\cdots|\tau_k}$ is a discrete probability distribution (cf. Definition 6) over the elements $\tau_1, \ldots, \tau_k$, we have

$$\sum_{1 \leq i \leq k} P_{\tau_1|\cdots|\tau_k}(\tau_i) = 1$$

hence Equation 1 is satisfied as well.

sequence for the sequence mst, we have $\mathit{yields}_p(\tau_1 \circ \cdots \circ \tau_k, \sigma, 1{:}(\rho_1 \bullet \cdots \bullet \rho_k))$, so Equation 1 is trivially satisfied again.

depth $= n$**:** assume Equation 1 holds for mst's of nesting depth $n-1$, we show it holds also for mst's of depth n. Again we assume that ϕ is a query formula of $\mathcal{A}_p$ and $\tau_1, \ldots, \tau_k$, are compound mst's of maximal nesting depth $n-1$ yielding sets of updates $\mathfrak{fp}_{\tau_1}(\sigma), \ldots, \mathfrak{fp}_{\tau_k}(\sigma)$ in a mental state σ respectively. Similarly to the previous step, we recognise three cases:

conditional according to the derivability of ϕ w.r.t. σ, for the conditional mst $\phi \longrightarrow \tau_1$ we have either $\mathfrak{fp}_{\phi \longrightarrow \tau_1}(\sigma) = \mathfrak{fp}_{\tau_1}(\sigma)$, or $\mathfrak{fp}_{\phi \longrightarrow \tau_1}(\sigma) = \{1{:}\mathbf{skip}\}$. For the latter case, Equation 1 is trivially satisfied and since τ_1 is of maximal nesting depth $n-1$, we have $\sum_{p:\rho \in \mathfrak{fp}_{\phi \longrightarrow \tau_1}(\sigma)} p = \sum_{p:\rho \in \mathfrak{fp}_{\tau_1}(\sigma)} p = 1$ as well.

choice let $P_{\tau_1|\cdots|\tau_k}$ be the probability distribution function assigned to the choice mst $\tau_1 | \cdots | \tau_k$ by the function $\mathit{\Pi}$. We have

$$\mathfrak{fp}_{\tau_1|\cdots|\tau_k}(\sigma) = \left\{ p{:}\rho \middle| \exists 0 \leq i \leq k : \mathit{yields}_p(\tau_i, \sigma, p_i{:}\rho) \wedge p = P_{\tau_1|\cdots|\tau_k}(\tau_i) \cdot p_i \right\}$$

Subsequently,

$$\sum_{p:\rho \in \mathfrak{fp}_{\tau_1|\cdots|\tau_k}(\sigma)} p = \sum_{0 \leq i \leq k} \left(P_{\tau_1|\cdots|\tau_k}(\tau_i) \cdot \sum_{p:\rho \in \mathfrak{fp}_{\tau_i}(\sigma)} p \right)$$

However, because of the induction assumption that Equation 1 holds for mst's τ_i with maximal nesting depth $n-1$, for all i $\sum_{p:\rho\in\mathfrak{fp}_{\tau_i}(\sigma)} p = 1$, and since $P_{\tau_1|\cdots|\tau_k}$ is a discrete probability distribution function, we finally arrive to

$$\sum_{p:\rho\in\mathfrak{fp}_{\tau_1|\cdots|\tau_k}(\sigma)} p = \sum_{0\leq i\leq k} P_{\tau_1|\cdots|\tau_k}(\tau_i) = 1$$

sequence for the sequence mst $\tau_1 \circ \cdots \circ \tau_k$, we have

$$\mathfrak{fp}_{\tau_1\circ\cdots\circ\tau_k}(\sigma) = \left\{ \prod_{i=1}^{k} p_i{:}(\rho_1 \bullet \cdots \bullet \rho_k) | \forall 1 \leq i \leq k : yields_p(\tau_i, \sigma, p_i{:}\rho_i) \right\}$$

and subsequently

$$\sum_{p:\rho\in\mathfrak{fp}_{\tau_1\circ\cdots\circ\tau_k}(\sigma)} p = \sum_{\prod_{i=1}^{k} p_i:(\rho_1\bullet\cdots\bullet\rho_k)\in\mathfrak{fp}_{\tau_1\circ\cdots\circ\tau_k}(\sigma)} \prod_{i=1}^{k} p_i \tag{2}$$

Observe, that if we fix the update sequence suffix $\rho_2 \bullet \cdots \bullet \rho_k$, the sum 2 can be rewritten as

$$\left(\sum_{p_1:\rho_1\in\mathfrak{fp}_{\tau_1}(\sigma)} p_1 \right) \cdot \left(\sum_{\prod_{i=2}^{k} p_i:(\rho_2\bullet\cdots\bullet\rho_k)\in\mathfrak{fp}_{\tau_2\circ\cdots\circ\tau_k}(\sigma)} \prod_{i=2}^{k} p_i \right)$$

Finally, by reformulation of the sum of products 2 as a product of sums and by applying the induction assumption for the mst's $\tau_1, \ldots, \tau_k$ of nesting depth $n-1$, we arrive to

$$\prod_{i=1}^{k} \sum_{p:\rho\in\mathfrak{fp}_{\tau_i}(\sigma)} p = \prod_{i=1}^{k} 1 = 1$$

Hence, Equation 1 is satisfied. □