Electronic Notes in Theoretical Computer Science 22 (1999)
URL: http://www.elsevier.nl/locate/entcs/volume22.html 26 pages

On the Semantic Foundations of Probabilistic Synchronous Reactive Programs

Christel Baier

Fakultät für Mathematik & Informatik
Universität Mannheim, 68131 Mannheim, Germany
baier@pi2.informatik.uni-mannheim.de

Edmund M. Clarke and Vasiliki Hartonas-Garmhausen

Department of Computer Science, Carnegie Mellon University
Pittsburgh, PA 15213, USA
{emc,hartonas}@cs.cmu.edu

Abstract

In this paper we consider synchronous parallel programs $\mathcal{P}$ that are composed by sequential randomized processes $\mathcal{S}_1, \ldots, \mathcal{S}_k$ which communicate via shared variables. First, we give an operational semantics for the sequential components $\mathcal{S}_i$ on the basis of a transition relation defined in the classical SOS-style à la Plotkin [Plo81] which we use to specify the behaviour of $\mathcal{P}$ by a Markov chain whose transitions stand for the cumulative effect of the activities of the components $\mathcal{S}_1, \ldots \mathcal{S}_k$ within one time step. Second, we provide a denotational semantics for $\mathcal{P}$ that also models $\mathcal{P}$ by a Markov chain. It is based on a (denotational) least fixed point semantics for the sequential components which formalizes the input/output behaviour of the sequential components within one time step. While the operational (declarative) semantics might be the one that a designer (who provides the input for the tool) has in mind, the denotational (procedural) semantics is the one that a compiler might use. We establish a consistency result stating that the Markov chains induced by the operational and denotational semantics are bisimilar in the sense of [LS91].

1 Introduction

In the literature, various algorithms for analyzing the quantitative temporal behaviour of probabilistic systems described by an abstract model (e.g. Markov chain or Markov decision process) have been proposed. E.g., methods that are designed for Markov chains are presented in [VW86,CY88,CC92,HJ94,HMP+94,CY95,BCH+97]. Such algorithms can serve as basis for a model checking tool [CE81,CES86] that takes as its input a probabilistic program $\mathcal{P}$ and its specification Φ (e.g. a temporal logical formula) and returns the answer "yes" or "no" depending on whether or not $\mathcal{P}$ meets its specification. The development of such tools requires an appropriate specification language for the program $\mathcal{P}$ together with a procedure that generates automatically the semantic model for $\mathcal{P}$ (e.g. a Markov chain). For instance, in the tool ProbVERUS [Har98,HCC99], a model checker for parallel randomized programs against *PCTL* formulas [HJ94] has been implemented where the input program $\mathcal{P}$ arises through the parallel composition of sequential randomized processes $\mathcal{S}_1, \ldots, \mathcal{S}_k$ that communicate via shared variables and are specified in an imparative C-like language. The parallel composition is *lazy synchronous* (in the style of [CGL94,Cam96]) which means that the sequential processes $\mathcal{S}_1, \ldots, \mathcal{S}_k$ work independently between the synchronization points. Each step of $\mathcal{P}$ is composed from the independent execution of sequences of activities of the sequential components $\mathcal{S}_1, \ldots, \mathcal{S}_k$ and is viewed to take one time unit. [1]

In this paper, we consider a specification language, similar to the one used in [Har98,HCC99], and present an operational and denotational semantics for the sequential processes which yield semantic descriptions of $\mathcal{P}$ by Markov chains. We establish a consistency result stating that the Markov chains obtained by the operational and denotational semantics are bisimilar.

The operational semantics for the sequential processes $\mathcal{S}_i$ is based on a formalization of the stepwise behaviour of $\mathcal{S}_i$ by an operational semantics in the classical SOS-style à la Plotkin [Plo81] using probability-labelled transitions of the form

$$\langle \mathit{stmt}, \sigma \rangle \rightarrow_q^{e_i} \langle \mathit{stmt}', \sigma' \rangle.$$

Here, *stmt*, *stmt'* are statements of the language used for specifying the behaviour of the sequential components, σ, σ' are interpretations for the variables that are under the control of $\mathcal{S}_i$ and e_i is the "environment" in which $\mathcal{S}_i$ works (i.e. e_i gives the values for the variables that are not under the control of $\mathcal{S}_i$). The value q is a real number in the interval $(0, 1]$ that denotes the probability for the above transition, i.e. the chance that the execution of the first command in *stmt* changes the values of the variables that are under the control of $\mathcal{S}_i$ according to σ' and leads to a (local) state where *stmt'* is the statement that $\mathcal{S}_i$ has to perform next; provided that the current values of the variables are given by σ and e_i. Thus, the first component *stmt* of a local state $\langle \mathit{stmt}, \sigma \rangle$ can be viewed as a control component for $\mathcal{S}_i$. We formalize the *one-time-step behaviour* of $\mathcal{S}_i$ in the environment e_i by the probabilities $\mathbf{P}_i^{e_i}(s_i, t_i)$ for $\mathcal{S}_i$ to move from the local state s_i to the local state t_i (where we deal with the probability measure in the Markov chain induced by the probability-labelled transition relation $\rightarrow^{e_i}$). As we suppose the sequential components $\mathcal{S}_1, \ldots, \mathcal{S}_k$ to act independently between the synchronization points the transition probability $\mathbf{P}(\bar{s}, \bar{t})$ for $\mathcal{P}$ to move from the global state $\bar{s}$ to the global state $\bar{t}$ within one time step is obtained by taking the product of the probabilities $\mathbf{P}_i^{e_i}(s_i, t_i)$. Here, the global states $\bar{s} = \langle s_1, \ldots, s_k \rangle$ and $\bar{t} = \langle t_1, \ldots, t_k \rangle$ are composed by the local states s_i, t_i for

[1] To avoid the typical reader/writer-problems, each program variable v is under the control of exactly one of the sequential components $\mathcal{S}_i$. All other components $\mathcal{S}_h$ can only read the current value of v at each synchronization point; but they do not have writing access to v.

the sequential processes $\mathcal{S}_i$. e_i denotes the environment for $\mathcal{S}_i$ that is given by the local states s_h, $h \neq i$.

The denotational semantics: The operational semantics formalizes the intuition about the behaviour of a randomized parallel program $\mathcal{P}$; thus, it will be the semantics that a designer (who provides the input for the tool) has in mind when he writes down the specifications for the sequential processes $\mathcal{S}_i$. On the other hand, this operational semantics is not adequate for a compiler since it uses statements as control components. For this reason, we take up the ideas of [CGL94,Cam96,Har98,HCC99] and provide an alternative semantics that uses integer-valued variables as control components for the sequential processes and can serve as basis for a compiler that computes the Markov chain for $\mathcal{P}$. The control components can be viewed as pointers to the locations at which the executions of the sequential processes are.

In a first step, we modify the statements for the sequential components by introducing special commands for these control variables. Like the operational semantics described above, this alternative semantics assigns a Markov chain to $\mathcal{P}$ but uses a denotational semantics $\mathcal{D}^{e_i}$ for the (modified) statements rather than the transition probabilities $\mathbf{P}_i^{e_i}(\cdot)$. Intuitively, $\mathcal{D}^{e_i}[\![\mathsf{stmt}]\!]$ describes the *probabilistic input/output behaviour* of stmt within one time step when executed in the "environment" e_i and can be viewed as the probabilistic and timed counterpart to the classical denotational input/output semantics for sequential (non-randomized, untimed) programs à la Scott. The definition of $\mathcal{D}^{e_i}[\![\mathsf{stmt}]\!]$ uses structural induction on the syntax of stmt which can be translated into a recursive procedure for computing $\mathcal{D}^{e_i}[\![\mathsf{stmt}]\!]$.

Consistency: At this stage, we have two semantic descriptions for $\mathcal{P}$: the operational (*declarative*) semantics that the designer has in mind and that is independent of any details about the compiler (e.g. the introduction of control variables and special commands for them into the source code for the sequential processes) and a denotational (*procedural*) semantics that a compiler might use to generate a Markov chain for $\mathcal{P}$. Thus, in the view of the designer, $\mathcal{P}$ meets the specification Φ iff the Markov chain induced by the operational semantics satisfies Φ while a tool (whose compiler uses the denotational semantics) returns the answer "$\mathcal{P}$ satisfies Φ" iff Φ is satisfied by the Markov chain induced by the denotational semantics. In Section 6 we establish a consistency result stating the *bisimulation equivalence* (in the sense of Larsen & Skou [LS91]) of the Markov chains induced by the operational and denotational semantics. This ensures the equivalence of the two Markov chains with respect to all properties that are expressed in a formalism which does not distinguish between bisimilar programs (such as $PCTL^*$ [ASB$^+$95]), and thus guarantees that the view of the designer is "consistent" with the calculations of the tool.

Organization of the paper: In Section 2 we briefly recall some basic notions concerning our model of *fully probabilistic systems.* Section 3 explains the syntax of parallel randomized programs. Sections 4 and 5 present the operational and denotational semantics respectively while Section 6 shows the consistency of them. Concluding remarks are given in Section 7.

2 Preliminaries: Fully probabilistic systems

In this section we briefly explain the model for probabilistic process that we use for the operational and denotational semantics. Our model is based on sequential discrete-time Markov chains where each state is associated with a distribution that gives the probabilities for the possible successor states. (For further details about the background in measure or probability theory see e.g. [Hal50,Fel68].)

Fully probabilistic systems: A *fully probabilistic system* is a pair $(S, \mathbf{P})$ consisting of a set S of *states* and a *transition probability function* $\mathbf{P} : S \times S \to [0,1]$ such that, for each $s \in S$, $\mathbf{P}(s,t) \neq 0$ for at most finitely many $t \in S$ and $\sum_{t \in S} \mathbf{P}(s,t) \leq 1$. If $C \subseteq S$ then we define $\mathbf{P}(s,C) = \sum_{t \in C} \mathbf{P}(s,t)$. A state $s \in S$ is called *terminal* iff $\mathbf{P}(s,S) = 0$. A state $s \in S$ is called *stochastic* iff $\mathbf{P}(s,S) = 1$; otherwise, s is called *substochastic*. $(S, \mathbf{P})$ is called *stochastic* iff all states are stochastic. Each fully probabilistic system $(S, \mathbf{P})$ can be "extended" to a stochastic fully probabilistic system $(S \cup \{\perp\}, \mathbf{P}_\perp)$ where $\perp \notin S$, $\mathbf{P}_\perp(s,t) = \mathbf{P}(s,t)$ if s, $t \in S$, and, for $s \in S$,

$$\mathbf{P}_\perp(s,\perp) \;=\; 1 - \mathbf{P}(s,S),\ \mathbf{P}_\perp(\perp,\perp) = 1 \text{ and } \mathbf{P}_\perp(\perp,s) = 0.$$

$(S \cup \{\perp\}, \mathbf{P}_\perp)$ is called the *stochastic extension* of $(S, \mathbf{P})$.

Paths can be viewed as execution sequences; they arise by resolving the probabilistic choices. Formally, a *path* in a fully probabilistic system $(S, \mathbf{P})$ is a nonempty (finite or infinite) sequence $\pi = s_0 s_1 s_2, \ldots$ where s_i are states in the stochastic extension $(S \cup \{\perp\}, \mathbf{P}_\perp)$ and $\mathbf{P}_\perp(s_{i-1}, s_i) > 0$, $i = 1, 2, \ldots$. The first state s_0 of π is denoted by *first*(π). If $\pi = s_0 s_1 s_2 \ldots$ and $s_k \in S$, $s_{k+1} = s_{k+2} = \ldots = \perp$ then we define *last*$(\pi) = s_k$. If $s_k \in S$ for all $k \geq 0$ then *last*(π) is undefined. $\pi(k)$ denotes the k-th state of π (i.e. if $\pi(k) = s_k$). $Path_\omega(s)$ denotes the set of infinite paths π with *first*$(\pi) = s$. If σ is a finite path then $Cyl(\sigma)$ denotes the *basic cylinder* induced by σ, i.e. $Cyl(\sigma)$ is the set of all infinite paths π where σ is a prefix of π.

The probability measure on fully probabilistic systems: For $s \in S$, let $\Sigma(s)$ be the smallest σ-field on $Path_\omega(s)$ which contains the basic cylinders $Cyl(\sigma)$ where σ ranges over all finite paths starting in s. The probability measure *Prob* on $\Sigma(s)$ is the unique measure with $Prob(Cyl(\sigma)) = \mathbf{P}(\sigma)$ where $\mathbf{P}(s_0 s_1 \ldots s_k) = \mathbf{P}_\perp(s_0, s_1) \cdot \mathbf{P}_\perp(s_1, s_2) \cdot \ldots \cdot \mathbf{P}_\perp(s_{k-1}, s_k)$.

Labelled fully probabilistic systems: In what follows, AP denotes a finite set of *atomic propositions*. A *labelled* fully probabilistic system is a tuple $(S, \mathbf{P}, L)$ consisting of a fully probabilistic system $(S, \mathbf{P})$ and a labelling $L : S \to 2^{AP}$. For the stochastic extension, we suppose $L(\perp) = \emptyset$.

Bisimulation equivalence: We recall the definition of bisimulation equivalence (reformulated for labelled fully probabilistic systems) à la Larsen & Skou [LS91]. A *bisimulation* for a labelled fully probabilistic system $(S, \mathbf{P}, L)$ is an equivalence relation R on S such that, if $(s, s') \in R$ then $L(s) = L(s')$ and $\mathbf{P}(s,C) = \mathbf{P}(s',C)$ for all equivalence classes $C \in S/R$. Two states s, s' are called *bisimilar* iff $(s,s') \in R$ for some bisimulation R.

Fully probabilistic processes: A *fully probabilistic process* denotes a tuple $(S, \mathbf{P}, s)$ consisting of a fully probabilistic system $(S, \mathbf{P})$ and an initial state $s \in S$. Similarly, a *labelled fully probabilistic process* denotes a tuple $\mathcal{M} = (S, \mathbf{P}, L, s_{init})$ consisting of a labelled fully probabilistic system $(S, \mathbf{P}, L)$ and an initial state $s_{init} \in S$. Two fully probabilistic processes $\mathcal{M}_1 = (S_1, \mathbf{P}_1, L_1, s_1)$ and $\mathcal{M}_2 = (S_2, \mathbf{P}_2, L_2, s_2)$ are said to be *bisimilar* (written $\mathcal{M}_1 \sim \mathcal{M}_2$) iff the initial states s_1 and s_2 are bisimilar in the "composed" system $(S_1 \uplus S_2, \mathbf{P}, L)$ where $\uplus$ denotes disjoint union, $\mathbf{P}(s, s') = \mathbf{P}_i(s, s')$ if $s, s' \in S_i$, $i = 1, 2$, $\mathbf{P}(s, s') = 0$ in all other cases, and $L(s) = L_i(s)$ if $s \in S_i$.

3 A parallel randomized language

In this section we explain the syntax of the specification language which is similar to the one used in ProbVERUS [Har98,HCC99]. [2] In our setting, a program $\mathcal{P}$ consists of sequential randomized components $\mathcal{S}_1, \ldots, \mathcal{S}_k$ that are executed in parallel and that communicate via shared variables where each variable is under the control of exactly one sequential component $\mathcal{S}_i$. The parallel composition is synchronous in a *lazy* style, i.e. within each (time) step of $\mathcal{P}$ (between the synchronization points), the sequential components work independently. Termination of one of the components $\mathcal{S}_i$ does not block the other components. The sequential processes $\mathcal{S}_i$ are specified by statements of an imperative (C-like) language with assignment, while-loops, conditional commands and

- a probabilistic choice operator $\texttt{pselect}(p_1 : \mathit{stmt}_1, \ldots, p_m : \mathit{stmt}_m)$ that assigns the probability p_i to the statement stmt_i
- the command `wait` that forces the component to be idle until the other sequential components are ready for synchronization.

One (time) step of $\mathcal{P}$ is composed by the parallel (independent) execution of sequences of commands between two wait commands. [3]

Types, variables, expressions and conditions: Let $\mathcal{T}$ be a finite set of *types* (i.e. finite sets of certain values) including the type $\mathit{Bool} = \{\mathit{tt}, \mathit{ff}\}$. For each type $T \in \mathcal{T}$ we have a finite set $\mathit{Op}(T)$ of operators $op : T_1 \times \ldots \times T_r \to T$ where $r \geq 1$ and $T_1, \ldots, T_r \in \mathcal{T}$. Let Var be a finite set of variables where each variable $v \in \mathit{Var}$ is associated with a type in $\mathcal{T}$, denoted $\mathit{Type}(v)$. *Expressions* of type T are built from the production system:

$$\mathit{expr} \ ::= \ \mathit{const} \ \mid \ v \ \mid \ op(\mathit{expr}_1, \ldots, \mathit{expr}_r)$$

where $\mathit{const} \in T$, $v \in \mathit{Var}$ with $\mathit{Type}(v) = T$, $op : T_1 \times \ldots \times T_r \to T$ is a r-ary operator in $\mathit{Op}(T)$, expr_i is an expression of type T_i. $\mathit{Expr}(T)$ denotes the set of expressions of type T, $\mathit{BExpr} = \mathit{Expr}(\mathit{Bool})$ the set of *boolean expressions* or *conditions*.

Evaluations, environments: Let $V \subseteq \mathit{Var}$ be a set of (typed) variables. An *evaluation* for V is a function $\sigma : V \to \bigcup_{T \in \mathcal{T}} T$, $v \mapsto \sigma.v$ that is type-consistent,

[2] The core language is a probabilistic variant of the language used in VERUS [Cam96] where the non-deterministic choice operator `select(...)` is replaced by a probabilistic choice operator `pselect(...)`. For simplicity, the real-time constructs like deadlines, time delays or periodic statements of [Cam96] are omitted but could be added as well.

[3] Here, termination is viewed as performing infinitely many `wait`'s.

i.e. $\sigma.v \in \mathit{Type}(v)$ for all $v \in V$. $\mathit{Eval}(V)$ denotes the set of evaluations for V. If σ is an evaluation, $n \geq 1$, $v_1, \ldots, v_n \in \mathit{Var}$ are pairwise distinct variables and $x_i \in \mathit{Type}(v_i)$, $i = 1, \ldots, n$ then $\sigma[v_1 := x_1, \ldots, v_n := x_n]$ denotes the evaluation that coincides with σ for all variables $w \notin \{v_1, \ldots, v_n\}$ and returns x_i for the variable v_i.[4] If $\sigma_i \in \mathit{Eval}(V_i)$, $i = 1, 2$, with $V_1 \cap V_2 = \emptyset$ then (σ_1, σ_2) denotes the evaluation for $V_1 \cup V_2$ with $(\sigma_1, \sigma_2).v = \sigma_i.v$ if $v \in V_i$, $i = 1, 2$. If $\sigma \in \mathit{Eval}(V)$, $W \subseteq V$ then $\sigma.W$ denotes the unique evaluation on W with $(\sigma.W).w = \sigma.w$ for all $w \in W$. Given an expression $\mathit{expr} \in \mathit{Expr}(T)$ and an evaluation σ for a superset of Var, $[\![\mathit{expr}]\!](\sigma)$ denotes the value of the expression expr when evaluated over σ.[5] An *environment* for $V \subseteq \mathit{Var}$ is an evalution e for a superset of $\mathit{Var} \setminus V$. Let $\mathit{Env}(V)$ denote the collection of all environments for V.

Statements: Statements over V are built from the following grammar.

$$\begin{aligned}
\mathit{stmt} ::= {} & \texttt{wait} \;\Big|\; \texttt{skip} \;\Big|\; v := \mathit{expr} \;\Big|\; \mathit{stmt}_1; \mathit{stmt}_2 \;\Big| \\
& \texttt{while}\ \mathit{cond}\ \{\mathit{stmt}\} \;\Big|\; \texttt{pselect}(p_1 : \mathit{stmt}_1, \ldots, p_m : \mathit{stmt}_m) \;\Big| \\
& \texttt{if}\ \mathit{cond}\ \texttt{then}\ \mathit{stmt}_1\ \texttt{else}\ \mathit{stmt}_2
\end{aligned}$$

where $v \in V$, $\mathit{expr} \in \mathit{Expr}(\mathit{Type}(v))$, $\mathit{cond} \in \mathit{BExpr}$, $m \geq 1$ is a natural number and $p_1, \ldots, p_m \in]0, 1]$ with $p_1 + \ldots + p_m = 1$. $\mathit{Stmt}(V)$ denotes the set of statements over V, Stmt the set of all statements. We define WStmt to be the set of statements that “start” with a wait command. Formally, WStmt is the smallest subset of Stmt such that $\texttt{wait} \in \mathit{WStmt}$ and, if $\mathit{wstmt} \in \mathit{WStmt}$ and $\mathit{stmt} \in \mathit{Stmt}$ then $\mathit{wstmt}; \mathit{stmt} \in \mathit{WStmt}$. We define $\mathit{Stmt}^+ = \mathit{Stmt} \cup \{\texttt{exit}\}$ and $\mathit{WStmt}^+ = \mathit{WStmt} \cup \{\texttt{exit}\}$ where $\texttt{exit}$ is an auxiliary statement that denotes termination. Let $\mathit{WStmt}(V) = \mathit{WStmt} \cap \mathit{Stmt}(V)$ and $\mathit{WStmt}^+(V) = \mathit{WStmt}(V) \cup \{\texttt{exit}\}$.

Sequential randomized components: A *sequential randomized component* is a tuple $\mathcal{S} = \langle V, \mathit{wstmt} \rangle$ consisting of a subset V of Var and a statement $\mathit{wstmt} \in \mathit{WStmt}(V)$.[6]

Parallel randomized programs: A *parallel randomized program* is a tuple $\mathcal{P} = \langle \bar{\sigma}, \mathcal{S}_1, \ldots, \mathcal{S}_k \rangle$ where $\bar{\sigma} \in \mathit{Eval}(\mathit{Var})$ is an *initial evaluation* and $\mathcal{S}_1, \ldots, \mathcal{S}_k$ are sequential randomized components such that, if $\mathcal{S}_i = \langle V_i, \mathit{wstmt}_i^0 \rangle$, $i = 1, \ldots, k$ then $V_i \cap V_h = \emptyset$ if $1 \leq i < h \leq k$, and $\mathit{Var} = \bigcup_{1 \leq i \leq k} V_i$.

Intuitively, $\mathcal{P} = \langle \bar{\sigma}, \mathcal{S}_1, \ldots, \mathcal{S}_k \rangle$ stands for the parallel execution of the sequential processes $\mathcal{S}_1, \ldots, \mathcal{S}_k$ between the wait commands. More precisely, each step of $\mathcal{P}$ is composed by the activities of the processes $\mathcal{S}_i$ between two `wait`'s. $\mathcal{S}_1, \ldots, \mathcal{S}_k$ synchronize at the `wait`'s, i.e. $\mathcal{S}_i$ reads the current values of the variables $v \in \mathit{Var} \setminus V_i$. At each `wait`, time increases by 1. Thus, we may assume that the time that passes

[4] I.e. $\sigma[v_1 := x_1, \ldots, v_n := x_n].w = \sigma.w$ if $w \notin \{v_1, \ldots, v_n\}$, $\sigma[v_1 := x_1, \ldots, v_n := x_n].v_i = x_i$.

[5] Formally, we define $[\![\mathit{expr}]\!]$ by structural induction: $[\![\mathit{const}]\!](\sigma) = \mathit{const}$, $[\![v]\!](\sigma) = \sigma.v$ and $[\![\mathit{op}(\mathit{expr}_1, \ldots, \mathit{expr}_r)]\!](\sigma) = \mathit{op}\,([\![\mathit{expr}_1]\!](\sigma), \ldots, [\![\mathit{expr}_r]\!](\sigma))$.

[6] Note that only the values of the variables $v \in V$ can be modified by $\mathcal{S}$; the variables $v \notin V$ can only be read by $\mathcal{S}$. The variables $w \in \mathit{Var} \setminus V$ might occur in the expression expr of an assignment or in the condition of a while-loop or conditional command.

between two `wait`'s is one time step. The initial evaluation $\bar{\sigma}$ gives the initial values of the variables, i.e. for $v \in \mathit{Var}$, $\bar{\sigma}.v \in \mathit{Type}(v)$ is the initial value of v.[7]

4 Operational semantics: the wait graph

We describe the behaviour of a parallel randomized program $\mathcal{P}$ by a Markov chain (with transition probability function $\mathbf{P}_{wg}$) that we derive from an operational semantics for the sequential processes $\mathcal{S}_1, \ldots, \mathcal{S}_k$. The transition probabilities $\mathbf{P}_{wg}(\bar{s}, \bar{t})$ assert that, from the global state $\bar{s}$, the global state $\bar{t}$ is reached within one time step with probability $\mathbf{P}_{wg}(\bar{s}, \bar{t})$. The resulting graph (whose nodes are the global states and whose edges are labelled with non-zero probabilities) is called the *wait graph* of $\mathcal{P}$ because each edge describes a possible behaviour of $\mathcal{P}$ between two `wait`'s.

Let $\mathcal{P} = \langle \bar{\sigma}, \mathcal{S}_1, \ldots, \mathcal{S}_k \rangle$ be a parallel randomized program. The *global states* of $\mathcal{P}$ are tuples $\bar{s} = \langle s_1, \ldots, s_k \rangle$ consisting of local states s_i for each of the sequential processes $\mathcal{S}_i$. The *local states* of $\mathcal{S}_i$ are pairs $s_i = \langle \mathit{wstmt}, \sigma \rangle$ where $\mathit{wstmt} \in \mathit{WStmt}^+(V_i)$ is the control component (that denotes the statement that $\mathcal{S}_i$ has to execute next when the local state of $\mathcal{S}_i$ is s_i) and σ is an interpretation for the variables $v \in V_i$ (i.e. $\sigma \in \mathit{Eval}(V_i)$). As $\mathcal{S}_1, \ldots, \mathcal{S}_k$ work independently between the synchronization points (the `wait`'s), the transition probabilities $\mathbf{P}_{wg}(\bar{s}, \bar{t})$ are given by the product of the probabilities $\mathbf{P}_i(s_i, t_i)$ for $\mathcal{S}_i$ to reach the local state t_i from s_i within one time step. Since the sequential components communicate via shared variables[8] the probabilities $\mathbf{P}_i(s_i, t_i)$ do not only depend on s_i but also on the local states s_h, $h \neq i$ (namely, on the interpretation of the variables $w \in V_h$, $h \neq i$). Thus, the transition probabilities for $\mathcal{P}$ are of the form

$$(*) \qquad \mathbf{P}_{wg}(\bar{s}, \bar{t}) \;=\; \prod_{1 \le i \le k} \mathbf{P}_i^{e_i}(s_i, t_i)$$

where e_i denotes the environment in which the component $\mathcal{S}_i$ works when the global state of $\mathcal{P}$ is $\bar{s}$. That is, e_i is the interpretation for the variables $w \in \mathit{Var} \setminus V_i$ in the global state $\bar{s}$, i.e. $e_i \in \mathit{Env}(V_i)$.

4.1 The one-time-step behaviour of the sequential processes

The transition probabilities $\mathbf{P}_i^{e_i}(s_i, t_i)$ in formula (*) describe the *one-time-step behaviour* of $\mathcal{S}_i$ in the environment e_i. In this section, we give a formal definition of these transition probabilities by means of an operational semantics of the statements over a fixed subset V of Var relative to an environment $e \in \mathit{Env}(V)$. More

[7] The requirement that the statements wstmt_i^0 belong to WStmt ensures that the computation of $\mathcal{P}$ starts with a synchronization. The condition $V_i \cap V_h = \emptyset$ avoids the typical writing problems for parallel processes with shared variables. Each variable can be written by at most one process while it can be read by all components $\mathcal{S}_1, \ldots, \mathcal{S}_k$. The requirement that all variables $v \in \mathit{Var}$ belong to some V_i ensures that all variables of $\mathcal{P}$ are under the control of a sequential component.

[8] Recall that in wstmt_i the variables $w \in \mathit{Var} \setminus V_i$ might occur in the expression of an assignment or in the condition of a while-loop or conditional command.

precisely, we define values $\mathbf{P}_V^e(s,t)$ that denote the probabilities to reach the local states $t = \langle wstmt', \sigma' \rangle$ from $s = \langle wstmt, \sigma \rangle$ by executing *wstmt* until the next wait command occurs or the execution of *wstmt* terminates. [9] The transition probabilities of the sequential processes $\mathcal{S}_i$ in formula (*) are obtained by $\mathbf{P}_i^{e_i}(s_i,t_i) = \mathbf{P}_{V_i}^{e_i}(s_i,t_i)$.

In order to formalize the cumulative effect of sequences of the commands that are executed within one time step (between two `wait`'s), we first describe the *stepwise behaviour* of the statements $stmt \in Stmt(V)$ (when executed in the environment e that gives the values for the variables $w \in Var \setminus V$). For this, we use transitions of the form $\langle stmt, \sigma \rangle \ \rightarrow_q^e \ \langle stmt', \sigma' \rangle$ that assert that – with probability q – the execution of the first command in *stmt* (where the current values of the variables are given by σ and e) leads to the intermediate state $\langle stmt', \sigma' \rangle$ in which *stmt'* has to be executed next and where the current value of the variables $v \in V$ is given by σ'. [10] Formally, we define the transition relation

$$\rightarrow^e \subseteq Stmt(V) \times Eval(V) \times]0,1] \times Stmt^+(V) \times Eval(V)$$

by the axioms and rules shown in Figure 1. [11] Most of the rules are self-explanatory. In the rule for `pselect` we sum up the probabilities p_l where $stmt_l = stmt'$. This is necessary because we did not make a syntactic restriction on the statements inside a probabilistic choice; thus, there might be more than one index l with $stmt_l = stmt$. For instance, we have the transition $\langle \texttt{pselect}(\frac{1}{3} : \texttt{skip}, \frac{2}{3} : \texttt{skip}), \sigma \rangle \ \rightarrow_1^e \ \langle \texttt{skip}, \sigma \rangle$ where the transition probability 1 is obtained from the sum $\frac{1}{3} + \frac{2}{3}$. The auxiliary symbol `exit` is needed to model terminating behaviour [12] and for the handling of sequential composition. [13]

We now use the transition relation $\rightarrow^{e_i}$ to formalize the behaviour of the sequential processes $\mathcal{S}_i$ within one time step. Let $V = V_i$ and $e = e_i$. If $\mathcal{S}_i$ is in the local state $s = \langle wstmt, \sigma \rangle$ then the behaviour in the next time step is formalized by a fully probabilistic process $TSB(wstmt, \sigma, e)$ [14] where

- the states are pairs $\langle stmt, \sigma \rangle$ with $stmt \in Stmt(V_i)$ and $\sigma \in Eval(V_i)$,

---

[9] For instance, if *wstmt* is of the form `wait`; *stmt*; `wait` where *stmt* does not contain any wait command then the one-time-step behaviour of $\mathcal{S}_i$ is given by the cumulative effect of the commands in *stmt* where the initial interpretation of the variables is given by σ and e_i.

[10] Intuitively, the first command denotes an "elementary step" such as an idling step (`skip` or `wait`), a variable assignment, the evaluation of the condition of a while-loop or a conditional command or resolving a probabilistic choice ("tossing a coin").

[11] Note that, for all pairs $(\langle stmt, \sigma \rangle, \langle stmt', \sigma' \rangle)$, there is at most one q where $\langle stmt, \sigma \rangle \ \rightarrow_q^e \ \langle stmt', \sigma' \rangle$.

[12] For instance, the outgoing transitions of $\langle \texttt{skip}, \ldots \rangle$, $\langle \texttt{wait}, \ldots \rangle$ and $\langle v := expr, \ldots \rangle$ lead to a local state of the form $\langle \texttt{exit}, \ldots \rangle$. Similarly, if *cond* is a condition that evaluates to false when interpreted over e and σ then the statement `while` *cond* {*stmt*} immediately terminates after the first "elementary step" (i.e. after the evaluation of *cond*); thus, with probability 1, we get the transition to the local state $\langle \texttt{exit}, \ldots \rangle$.

[13] E.g., if $\langle stmt_1, \sigma \rangle \rightarrow_q^{e_i} \langle \texttt{exit}, \sigma' \rangle$ (i.e. with probability q, $stmt_1$ terminates after performing the first command) then $\langle stmt_1; stmt_2, \sigma \rangle \rightarrow_q^e \langle stmt_2; \sigma' \rangle$ (i.e. with probability q, the execution of $stmt_2$ starts after the execution of the first command of $stmt_1$).

[14] The letters *TSB* stand for "time step behaviour".

$$\langle \texttt{wait}, \sigma\rangle \;\to_1^e \langle \texttt{exit}, \sigma\rangle \qquad \langle \texttt{skip}, \sigma\rangle \;\to_1^e\; \langle \texttt{exit}, \sigma\rangle$$

$$\langle v := expr, \sigma\rangle \;\to_1^e\; \langle \texttt{exit}, \sigma[v := [\![expr]\!](e,\sigma)]\rangle$$

$$\frac{q \;=\; \sum_{l\in I}\; p_l \text{ where } I = \{1 \le l \le m : stmt_l = stmt'\}}{\langle \texttt{pselect}(p_1 : stmt_1, \ldots, p_m : stmt_m), \sigma\rangle \;\to_q^e\; \langle stmt', \sigma\rangle}$$

$$\frac{[\![cond]\!](e,\sigma)}{\langle \texttt{if}\; cond\; \texttt{then}\; stmt_1\; \texttt{else}\; stmt_2, \sigma\rangle \;\to_1^e\; \langle stmt_1, \sigma\rangle}$$

$$\frac{\neg[\![cond]\!](e,\sigma)}{\langle \texttt{if}\; cond\; \texttt{then}\; stmt_1\; \texttt{else}\; stmt_2, \sigma\rangle \;\to_1^e \langle stmt_2, \sigma\rangle}$$

$$\frac{[\![cond]\!](e,\sigma)}{\langle \texttt{while}\; cond\; \{stmt\}, \sigma\rangle \;\to_1^e\; \langle stmt; \texttt{while}\; cond\; \{stmt\}, \sigma\rangle}$$

$$\frac{\neg[\![cond]\!](e,\sigma)}{\langle \texttt{while}\; cond\; \{stmt\}, \sigma\rangle \;\to_1^e\; \langle \texttt{exit}, \sigma\rangle}$$

$$\frac{\langle stmt_1, \sigma\rangle \;\to_q^e\; \langle stmt', \sigma'\rangle \text{ and } stmt' \neq \texttt{exit}}{\langle stmt_1; stmt_2, \sigma\rangle \;\to_q^e\; \langle stmt'; stmt_2, \sigma'\rangle}$$

$$\frac{\langle stmt_1, \sigma\rangle \;\to_q^e\; \langle \texttt{exit}, \sigma'\rangle}{\langle stmt_1; stmt_2, \sigma\rangle \;\to_q^e\; \langle stmt_2, \sigma'\rangle}$$

Fig. 1. The stepwise behaviour of the statements in the environment e

- all local states of the form $\langle wstmt', \ldots\rangle$ with $wstmt' \in WStmt(V_i)$ are viewed as terminal states; the outgoing transitions of $\langle wstmt', \ldots\rangle$ with respect to $\to^{e_i}$ are ignored (these transitions represent steps that are executed in the *next* time step),
- the root (initial state) is an auxiliary state $s_{init} = s_{init}(wstmt, \sigma, e_i)$ whose outgoing edges are given by the transitions from $\langle wstmt, \sigma\rangle$.

Formally, we define $TSB(wstmt, \sigma, e) = (S, \mathbf{P}, s_{init})$ as follows. The state space S consists of all pairs $\langle stmt', \sigma'\rangle \in Stmt^+(V) \times Eval(V)$ and an additional state $s_{init} = s_{init}(wstmt, \sigma, e)$, i.e. $S \;=\; Stmt^+(V) \times Eval(V) \cup \{s_{init}\}$. The transition probability function $\mathbf{P}$ is defined as follows. If $\langle stmt', \sigma'\rangle \to_q^e \langle stmt'', \sigma''\rangle$ and $stmt' \notin WStmt^+$ then $\mathbf{P}(\langle stmt', \sigma'\rangle, \langle stmt'', \sigma''\rangle) = q$. The probabilities for the outgoing transitions from the initial state are given by

$$\mathbf{P}(s_{init}, \langle stmt', \sigma'\rangle) = q \text{ if } \langle wstmt, \sigma\rangle \to_q^e \langle stmt', \sigma'\rangle.$$

We put $\mathbf{P}(\cdot) = 0$ in all remaining cases.

Remark: The additional initial state s_{init} is needed since the state $\langle wstmt, \sigma\rangle$ is terminal in $TSB(wstmt, \sigma, e)$. Recall that the outgoing transitions of $\langle wstmt', \ldots\rangle$ where $wstmt' \in WStmt(V)$ with respect to $\to^e$ are ignored. On the other hand, we cannot add the outgoing transitions of such states $\langle wstmt', \ldots\rangle$ as they describe

$\texttt{wait};$

$b := tt;$

$\texttt{pselect}(\ \frac{1}{3} : \texttt{wait};$

$\qquad\qquad \texttt{while}\ b \wedge \neg c\ \ \{$

$\qquad\qquad\qquad \texttt{pselect}(\frac{1}{2} : b := \mathit{ff}, \frac{1}{2} : b := tt);$

$\qquad\qquad\qquad \texttt{wait}\ \ \},$

$\qquad\quad \frac{2}{3} : \texttt{skip}\ \ );$

$b := \neg b$

Fig. 2. The statement *wstmt*

activities of the next time step. E.g., if $[\![\mathit{cond}]\!](e, \sigma)$ is true then, for the statement

$$\mathit{wstmt}\ =\ \texttt{wait}; \texttt{while}\ \mathit{cond}\ \{\texttt{wait}\},$$

we obtain the transition $\langle \mathit{wstmt}, \sigma\rangle \rightarrow^e_1 \langle \texttt{while}\ \mathit{cond}\ \{\texttt{wait}\}, \sigma\rangle \rightarrow^e_1 \langle \mathit{wstmt}, \sigma\rangle$. The behaviour of *wstmt* within one time step (i.e. the behaviour of *wstmt* before the second $\texttt{wait}$ inside the while-loop is reached) consists of these two steps rather than the loop shown on the right that describes an infinite behaviour. ■

$$\begin{tikzcd}
\langle \mathit{wstmt}, \sigma\rangle \arrow[d, bend right] \\
\langle \texttt{while}\ \mathit{cond}\ \{\texttt{wait}\}, \sigma\rangle \arrow[u, bend right]
\end{tikzcd}$$

Example: Let $\mathit{Var} = \{b, c\}$ with $\mathit{Type}(b) = \mathit{Type}(c) = \mathit{Bool}$ and $V = \{b\}$. We consider the statement $\mathit{wstmt} \in \mathit{WStmt}(V)$ of Figure 2. We write $[b = x]$ for the evaluation $\sigma \in \mathit{Eval}(V)$ with $\sigma.b = x$. Similarly, $[c = x]$ is the environment e for V with $e.c = x$. Figure 4 shows the process $\mathit{TSB}(\mathit{stmt}, [b = \mathit{ff}], e)$ where e is an

$$\begin{aligned}
\mathit{pstmt}' &= \texttt{pselect}(\tfrac{1}{2} : b := \mathit{ff}, \tfrac{1}{2} : b := tt) \\
\mathit{whilestmt} &= \texttt{while}\ b \wedge \neg c\ \ \{\mathit{pstmt}'; \texttt{wait}\} \\
\mathit{wstmt}' &= \texttt{wait}; \mathit{whilestmt} \\
\mathit{wstmt}'' &= \mathit{wstmt}'; b := \neg b \\
\mathit{pstmt} &= \texttt{pselect}(\tfrac{1}{3} : \mathit{wstmt}', \tfrac{2}{3} : \texttt{skip}); b := \neg b \\
\mathit{stmt} &= b := tt; \mathit{pstmt}
\end{aligned}$$

Fig. 3. “Substatements” of $\mathit{wstmt} = \texttt{wait}; \mathit{stmt}$

arbitrary environment for V and where the “substatements” of *wstmt* are denoted as shown in Figure 3. Figure 5 shows the system $\mathit{TSB}(\mathit{wstmt}'', [b = \mathit{tt}], [c = \mathit{ff}])$.

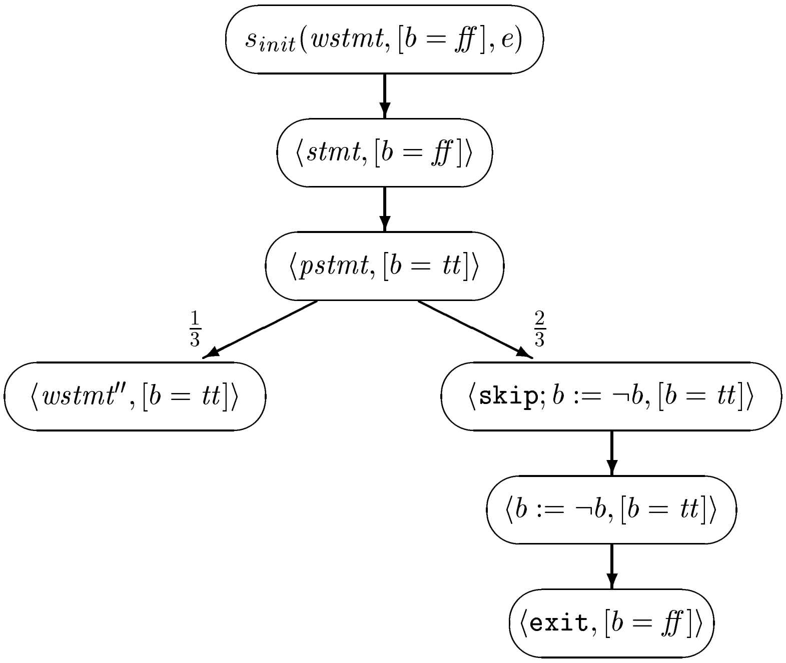

Fig. 4. The process $\mathit{TSB}(\mathit{wstmt}, [b = \mathit{ff}], e)$

Here, the condition $b \wedge \neg c$ of the while-loop is satisfied. Hence, by the rule for while-loops, $\langle \mathit{whilestmt}, [b = \mathit{tt}] \rangle \ \rightarrow_1^{[c=\mathit{ff}]} \ \langle \mathit{pstmt}'; \mathtt{wait}; \mathit{whilestmt}, [b = \mathit{tt}] \rangle$. Thus, by the rule for sequential composition:

$$\langle \mathit{whilestmt}; b := \neg b, [b = \mathit{tt}] \rangle \ \rightarrow_1^{[c=\mathit{ff}]} \ \langle \mathit{pstmt}'; \mathit{wstmt}'', [b = \mathit{tt}] \rangle.$$

Applying the rule for `pselect` and sequential composition yields

$$\langle \mathit{pstmt}'; \mathit{wstmt}'', [b = \mathit{tt}] \rangle \rightarrow_{\frac{1}{2}}^{[c=\mathit{ff}]} \langle b := x; \mathit{wstmt}'', [b = \mathit{tt}] \rangle$$

where $x \in \{\mathit{tt}, \mathit{ff}\}$. ■

The transition probabilities $\mathbf{P}_V^e(s,t)$: The cumulative effect of a statement $\mathit{wstmt} \in \mathit{WStmt}(V)$ within one time step (relative to an environment $e \in \mathit{Env}(V)$ and an initial evaluation $\sigma \in \mathit{Eval}(V)$) is obtained by taking the probabilities for the initial state s_{init} of $\mathit{TSB}(\mathit{wstmt}, \sigma, e)$ to reach the terminal states (i.e. the states of the form $\langle \mathit{wstmt}', \ldots \rangle$ or $\langle \mathtt{exit}, \ldots \rangle$). Formally, for $V \subseteq \mathit{Var}$, $e \in \mathit{Env}(V)$, σ, $\sigma' \in \mathit{Eval}(V)$ and $\mathit{wstmt} \in \mathit{WStmt}(V)$, $\mathit{wstmt}' \in \mathit{WStmt}^+(V)$, we define [15]

$$\mathbf{P}_V^e(\langle \mathit{wstmt}, \sigma \rangle, \langle \mathit{wstmt}', \sigma' \rangle) \ = \ \mathit{Prob}\left\{\pi \in \mathit{Path}_\omega(s_{init}) : \mathit{last}(\pi) = \langle \mathit{wstmt}', \sigma' \rangle\right\}.$$

For the special statement `exit`, we define $\mathbf{P}_V^e(\langle \mathtt{exit}, \sigma \rangle, \langle \mathtt{exit}, \sigma \rangle) = 1$, $\mathbf{P}_V^e(\langle \mathtt{exit}, \sigma \rangle, \langle \mathit{wstmt}', \sigma' \rangle) = 0$ if $\langle \mathit{wstmt}', \sigma' \rangle \neq \langle \mathtt{exit}, \sigma \rangle$. For instance, if wstmt, wstmt'' are as before (see Figure 2, 3 and 5) then

[15] Here, $\mathit{Prob}\{\ldots\}$ denotes the probability measure in $\mathit{TSB}(\mathit{wstmt}, \sigma, e)$ and s_{init} is the initial state of $\mathit{TSB}(\mathit{wstmt}, \sigma, e)$ (i.e. $s_{init} = s_{init}(\mathit{wstmt}, \sigma, e)$).

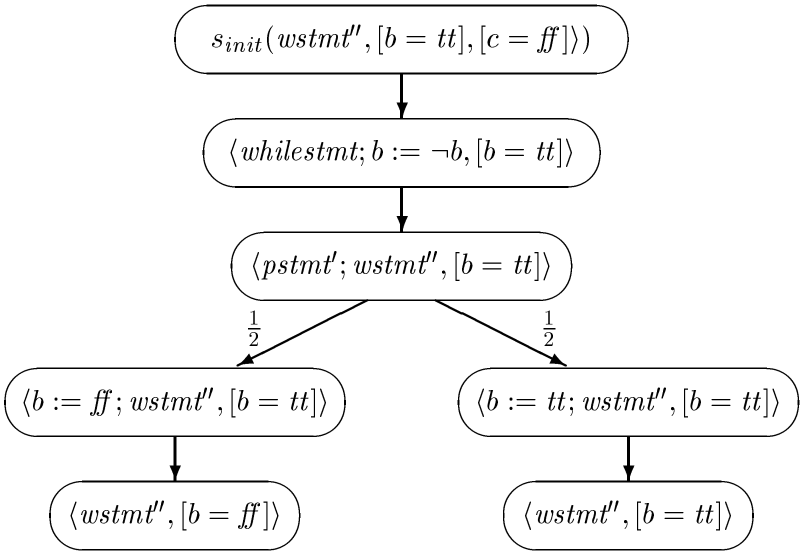

Fig. 5. The process $TSB(wstmt'', [b = tt], [c = ff])$

$$\mathbf{P}_V^e(\langle wstmt, [b = ff]\rangle, \langle \mathtt{exit}, [b = ff]\rangle) = \tfrac{2}{3},$$
$$\mathbf{P}_V^e(\langle wstmt, [b = ff]\rangle, \langle wstmt'', [b = tt]\rangle) = \tfrac{1}{3},$$
$$\mathbf{P}_V^{[c=ff]}(\langle wstmt'', [b = tt]\rangle, \langle wstmt'', [b = x]\rangle) = \tfrac{1}{2},$$
$$\mathbf{P}_V^{[c=y]}(\langle wstmt'', [b = x]\rangle, \langle \mathtt{exit}, [b = \neg x]\rangle) = 1$$

where $x \in \{ff, tt\}$, $y \in \{tt, x\}$. For all V, σ, e, $\mathbf{P}_V^e(\langle \mathtt{wait}, \sigma\rangle, \langle \mathtt{exit}, \sigma\rangle) = 1$.
Remark: Note that $1 - \sum_{wstmt',\sigma'} \mathbf{P}_V^e(\langle wstmt, \sigma\rangle, \langle wstmt', \sigma'\rangle)$ is the probability for *divergence.*[16] For instance, for the statement `wait; while` b `{skip}` and σ an evaluation for $V = \{b\}$ where $\sigma.b$ is true we have

$$\mathbf{P}_V^e(\langle \mathtt{wait; while}\ b\ \{\mathtt{skip}\}, \sigma\rangle, \langle wstmt', \sigma'\rangle) = 0$$

for all $wstmt'$ and σ'. Thus, the probability for divergence is 1. This reflects the fact that the while-loop never terminates and never reaches a state where the control component starts with a wait command. ■

4.2 The wait graph of a parallel randomized program

Let $\mathcal{P} = \langle \bar{\sigma}, \mathcal{S}_1, \ldots, \mathcal{S}_k\rangle$ be a parallel randomized program where $\mathcal{S}_i = \langle V_i, wstmt_i^0\rangle$. We define the wait graph of $\mathcal{P}$ to be a labelled fully probabilistic process where each global state consists of control components $wstmt_i \in WStmt^+(V_i)$ for each sequential process $\mathcal{S}_i$ and an evaluation for $Var = V_1 \cup \ldots \cup V_k$ that is composed by evaluations σ_i for V_i. The probability $\mathbf{P}_{wg}(\bar{s}, \bar{t})$ for $\mathcal{P}$ to move from $\bar{s} = \langle wstmt_1, \ldots, wstmt_k, \sigma_1, \ldots, \sigma_k\rangle$ to $\bar{t} = \langle wstmt'_1, \ldots, wstmt'_k, \sigma'_1, \ldots, \sigma'_k\rangle$ is the

[16] Here, divergence means the event of never reaching a terminal state (a "wait state" $\langle wstmt', \ldots\rangle$ or an "exit state" $\langle \mathtt{exit}, \ldots\rangle$).

product of the probabilities for $wstmt_i$ started in σ_i and executed in the environment $e_i = (\sigma_h)_{h \neq i}$ to reach $\langle wstmt'_i, \sigma'_i \rangle$ within one time step (cf. formula (*)).[17]

The wait graph: We use atomic propositions of the form $a_{v,x}$ where $v \in Var$ and $x \in Type(v)$. I.e. we deal with $AP = \{a_{v,x} : v \in Var, x \in Type(v)\}$. The intended meaning of $a_{v,x}$ is that the current value of v is x. Let $\mathcal{P} = \langle \bar{\sigma}, \mathcal{S}_1, \dots, \mathcal{S}_k \rangle$ be as before. The *wait graph* of $\mathcal{P}$ is the labelled fully probabilistic process $WG(\mathcal{P}) = (S_{wg}, \mathbf{P}_{wg}, L_{wg}, \bar{s}_{wg})$ where

$$S_{wg} = \big\{ \langle wstmt_1, \dots, wstmt_k, \sigma_1, \dots, \sigma_k \rangle : wstmt_i \in WStmt^+(V_i), \sigma_i \in Eval(V_i) \big\}$$

and the initial state is $\bar{s}_{wg} = \langle wstmt_1^0, \dots, wstmt_k^0, \bar{\sigma}.V_1, \dots, \bar{\sigma}.V_k \rangle$. The transition probability function $\mathbf{P}_{wg}$ is given by:

$$\begin{aligned} &\mathbf{P}_{wg}\left(\langle wstmt_1, \dots, wstmt_k, \sigma_1, \dots, \sigma_k \rangle, \langle wstmt'_1, \dots, wstmt'_k, \sigma'_1, \dots, \sigma'_k \rangle \right) \\ &= \prod_{1 \leq i \leq k} \mathbf{P}_{V_i}^{e_i}(\langle wstmt_i, \sigma_i \rangle, \langle wstmt'_i, \sigma'_i \rangle) \end{aligned}$$

where e_i is the environment for V_i that is composed by the evaluations σ_h, $h \neq i$, i.e. $e_i(v) = \sigma_h(v)$ if $v \in V_h$, $h \neq i$. The labelling function L_{wg} is given by:

$$L_{wg}(\langle wstmt_1, \dots, wstmt_k, \sigma_1, \dots, \sigma_k \rangle) = \bigcup_{1 \leq i \leq k} \{ a_{v, \sigma_i.v} : v \in V_i \}.$$

Example: Let $Var = \{b, c\}$, $Type(b) = Type(c) = Bool$. We consider the program $\mathcal{P} = \langle \bar{\sigma}, \mathcal{S}_1, \mathcal{S}_2 \rangle$ where $\bar{\sigma}.b = \mathit{ff}$ and $\bar{\sigma}.c = \mathit{ff}$ and $\mathcal{S}_1 = \langle V_1, wstmt_1^0 \rangle$ where $V_1 = \{b\}$ and $wstmt_1^0 = wstmt$ is as in Figure 2, $\mathcal{S}_2 = \langle V_2, wstmt_2^0 \rangle$ where $V_2 = \{c\}$ and $wstmt_2^0 = \mathtt{wait}; c := b$. The wait graph for $\mathcal{P}$ is shown in Figure 6. ■

5 Denotational semantics: the wait counter graph

For any automatic analysis of the behaviour of a parallel randomized program $\mathcal{P}$ (e.g. model checking against $PCTL$ specifications), the operational semantics (wait graph) is not adequate since the control components of $\mathcal{S}_1, \dots, \mathcal{S}_k$ are statements. In this section we give an alternative semantics for $\mathcal{P}$ which uses simpler control components. We follow the idea of [CGL94,Cam96,Har98] and use *wait counters* $\mathsf{wc}_1, \dots, \mathsf{wc}_k$ for the control components of $\mathcal{S}_1, \dots, \mathcal{S}_k$. wc_i is an integer variable whose current value is j iff the execution of $\mathcal{S}_i$ has reached the j-th occurrence of `wait` in $wstmt_i^0$. We associate with $\mathcal{P}$ the *wait counter graph* which is a fully probabilistic process whose states are tuples $\bar{s} = \langle s_1, \dots, s_k \rangle$ where $s_i \in Eval(V_i \cup \{\mathsf{wc}_i\})$, $i = 1, \dots, k$. I.e. in the wait counter graph, the control components are just interpretations of the wait counters. The wait counter graph is defined in a "denotational manner", using structural induction on the syntax of the statements $wstmt_i^0$ and a least fixed point operator for the handling of while-loops. This denotational approach can be used for an automatic procedure to obtain the wait counter graph of

[17] The fact that we multiply the probabilities $\mathbf{P}_{V_i}^{e_i}(\dots)$ for the individual moves of the sequential processes $\mathcal{S}_i$ reflects the assumption that $\mathcal{S}_1, \dots, \mathcal{S}_k$ work independently between the `wait`'s.

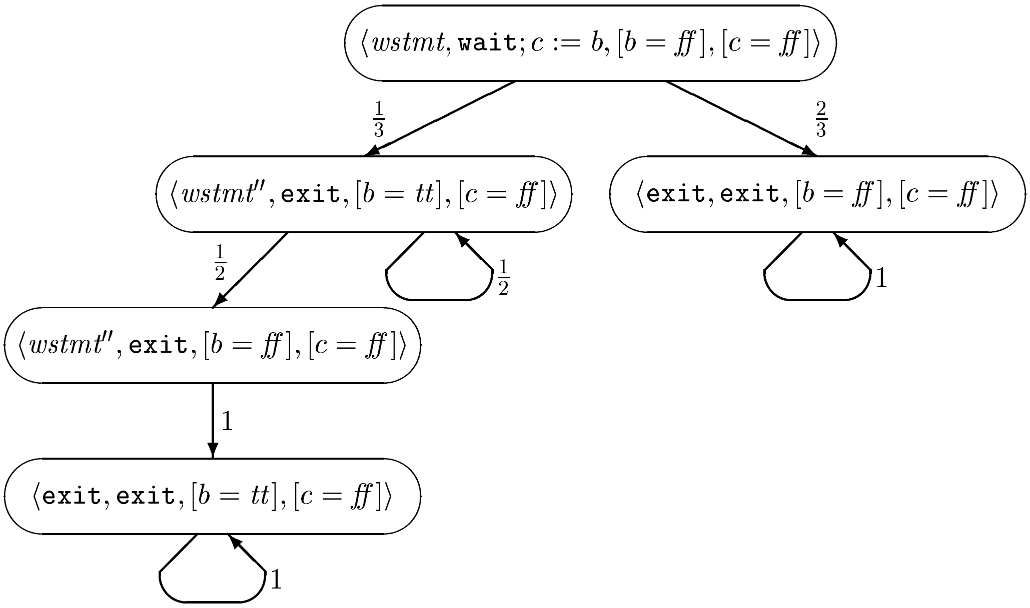

Fig. 6. The wait graph of $\mathcal{P}$

$\mathcal{P}$ where the least point operator for the while-loops is approximated by iteration (on the basis of Tarski's fixed point theorem).

The construction of the wait counter graph can be sketched as follows. In each statement $wstmt_i^0$, we replace the j-th occurrence of a wait command by wait_j. For these extended statements stmt [18], we give a denotational least fixed point semantics $\mathsf{stmt} \mapsto \mathcal{D}^e[\![\mathsf{stmt}]\!]$ (relative to an environment e) in the classical style à la Scott (Section 5.2). $\mathcal{D}^e[\![\mathsf{stmt}]\!]$ is a function that returns for each pair (s,t) of "local states" (interpretations of the variables of stmt, including the wait counter) the probability for stmt to reach t from s within one time step. Then, the one-time-step behaviour of $\mathcal{S}_i$ (relative to the environment e_i) in the local state s where the control is at the j-th wait command (i.e. $s.\mathsf{wc}_i = j$) is given by the function $t \mapsto \mathcal{D}^{e_i}[\![\pi_j(\mathsf{wstmt}_i^0)]\!](s,t)$ where $\pi_j(\mathsf{wstmt}_i^0)$ is the "substatement" of the extension wstmt_i^0 of $wstmt_i^0$ that starts with wait_j and that is obtained by unwinding all "relevant" while-loops. [19] The (global) transition probabilities $\mathbf{P}_{wcg}(\bar{s},\bar{t})$ for the wait counter graph are obtained by multiplying the probabilities $\mathcal{D}^{e_i}[\![\ldots]\!](s_i,t_i)$ for the individual moves of the sequential processes $\mathcal{S}_i$ within one time step.

5.1 Extended statements

The first step in the construction of the wait counter graph replaces each wait command by an indexed wait command; more precisely, the j-th occurrence of `wait` in $wstmt_i^0$ is replaced by wait_j. The index j is the value of the wait counter wc_i for $\mathcal{S}_i$ when the execution of $\mathcal{S}_i$ is at the j-th `wait` in $wstmt_i^0$. The introduction of

[18] The syntax of the extended statements arises from the syntax of the (ordinary) statements where the `wait` command is replaced by an indexed wait command wait_j, cf. Section 5.1.

[19] Here, "relevance" means that we consider those while-loops whose body contains wait_j.

$$
\begin{array}{l}
\mathsf{wait}_1; \\
b := tt; \\
\texttt{pselect}(\ \tfrac{1}{3} : \mathsf{wait}_2; \\
\qquad\qquad\quad \texttt{while}\ b \wedge \neg c\ \ \{ \\
\qquad\qquad\qquad \texttt{pselect}(\tfrac{1}{2} : b := \mathit{ff}, \tfrac{1}{2} : b := tt); \\
\qquad\qquad\qquad \mathsf{wait}_3\ \ \}, \\
\qquad\qquad \tfrac{2}{3} : \texttt{skip}\ \ ); \\
b := \neg b
\end{array}
$$

Fig. 7. The extension $\mathsf{ext}(\mathit{wstmt})$ of *wstmt*

these indexed wait commands leads to a new type of statements, called *extended* statements.

Syntax of extended statements: Let $V \subseteq \mathit{Var}$. $\mathsf{Stmt}(V)$ denotes the set of extended statements built from the following production system

$$
\begin{array}{rl}
\mathsf{stmt} ::= \mathsf{wait}_j \mid \texttt{skip} \mid v := \mathit{expr} \mid \mathsf{stmt}_1; \mathsf{stmt}_2 \mid \texttt{while}\ \mathit{cond}\ \ \{\mathsf{stmt}\} \mid \\
\texttt{pselect}(p_1 : \mathsf{stmt}_1, \ldots, p_m : \mathsf{stmt}_m) \mid \texttt{if}\ \mathit{cond}\ \texttt{then}\ \mathsf{stmt}_1\ \texttt{else}\ \mathsf{stmt}_2
\end{array}
$$

where j, $m \geq 1$ are natural numbers, $v \in V$, $\mathit{expr} \in \mathit{Expr}(\mathit{Type}(v))$, $\mathit{cond} \in \mathit{BExpr}$ and $p_1, \ldots, p_m$ are real numbers in $]0,1]$ with $p_1 + \ldots + p_m = 1$. We define $\mathsf{Stmt}^+(V) = \mathsf{Stmt}(V) \cup \{\texttt{exit}\}$.[20] An extended statement $\mathsf{stmt} \in \mathsf{Stmt}(V)$ is called *well-formed* iff, for each $j \geq 1$, the command wait_j occurs at most once in stmt. $\mathsf{WStmt}_j(V)$ (abbrev. WStmt_j) denotes the set of extended statements that "start" with wait_j. Let $\mathsf{WStmt}_\infty = \{\texttt{exit}\}$, $\mathsf{WStmt} = \bigcup_{j \geq 1} \mathsf{WStmt}_j$, $\mathsf{WStmt}^+ = \mathsf{WStmt} \cup \{\texttt{exit}\}$.

The extended statement $\mathsf{ext}(\mathit{stmt})$: Given $\mathit{stmt} \in \mathit{Stmt}^+(V)$, we transform *stmt* into a well-formed extended statement $\mathsf{ext}(\mathit{stmt}) \in \mathsf{Stmt}^+(V)$. $\mathsf{ext}(\mathit{stmt})$ arises from *stmt* by replacing the j-th occurrence of `wait` in *stmt* by the indexed wait command wait_j. E.g., the extension of the statement *wstmt* of Figure 2 is shown in Figure 7.

5.2 The probabilistic one time step denotations

We fix some subset V of *Var* and an environment e for V and give a denotational semantics $\mathcal{D}^e[\![\mathsf{stmt}]\!]$ for the extended statements $\mathsf{stmt} \in \mathsf{Stmt}^+(V)$ relative to an environment e for V. The basic idea is the use of a wait counter as control component whose current value is j if the control is at the indexed wait command $\texttt{wait}_j$.

[20] Intuitively, the auxiliary symbol `exit` corresponds to the indexed wait command wait_∞.

The denotational semantics $\mathcal{D}^e[\![\mathsf{stmt}]\!]$: Let wc be a "fresh" variable that does not belong to *Var*, called the *wait counter*. Let $\mathsf{stmt} \in \mathsf{Stmt}^+(V)$. We define a function

$$\mathcal{D}^e[\![\mathsf{stmt}]\!] : \mathit{Eval}(V \cup \{\mathsf{wc}\}) \times \mathit{Eval}(V \cup \{\mathsf{wc}\}) \to [0,1]$$

where $\mathcal{D}^e[\![\mathsf{stmt}]\!](s,s')$ returns the probability for stmt to reach s' from s within one time step. Thus, $\mathcal{D}^e[\![\mathsf{stmt}]\!]$ describes the input/output-behaviour of stmt within one time step: given the initial evaluation s (the input), within one time step, the execution of stmt leads with probability $\mathcal{D}^e[\![\mathsf{stmt}]\!](s,s')$ to the local state s' (the output).[21] We call $\mathcal{D}^e[\![\mathsf{stmt}]\!]$ the *probabilistic one-time step denotation* of stmt in the environment e. For extended statements whose first command is not a wait command (i.e. extended statements $\mathsf{stmt} \notin \mathsf{WStmt}$), one time step is the time that passes until a wait command is reached or stmt terminates. For $\mathsf{wstmt} \in \mathsf{WStmt}$, one time step is the time that passes between the first wait command (the first command in wstmt) and the next wait command or the termination of wstmt.

Recall that, for $s \in \mathit{Eval}(V \cup \{\mathsf{wc}\})$, $W \subseteq V \cup \{\mathsf{wc}\}$, $s.W$ is the unique evaluation $\sigma \in \mathit{Eval}(W)$ with $\sigma.w = s.w$ for all $w \in W$. Let $\mathit{Exit} = \{t \in \mathit{Eval}(V \cup \{\mathsf{wc}\}) : t.\mathsf{wc} = \infty\}$. We define $\mathcal{D}^e[\![\mathsf{stmt}]\!]$ by structural induction on the syntax of stmt.

- **Skip** and the **wait command:**

$$\mathcal{D}^e[\![\mathtt{skip}]\!](s, s[\mathsf{wc} := \infty]) \;=\; \mathcal{D}^e[\![\mathsf{wait}_j]\!](s, s[\mathsf{wc} := \infty]) \;=\; 1$$

  and $\mathcal{D}^e[\![\mathtt{skip}]\!](s,s') = \mathcal{D}^e[\![\mathsf{wait}_j]\!](s,s') = 0$ in all other cases.

- **Assignment for variables** $v \in V$:

$$\mathcal{D}^e[\![v := \mathit{expr}]\!](s,s') \;=\; \begin{cases} 1 : \text{if } s'.\mathsf{wc} = \infty,\ s'.v = [\![\mathit{expr}]\!](e,s) \\ \qquad \text{and } s.w = s'.w \text{ for all } w \in V \setminus \{v\} \\ 0 : \text{otherwise.} \end{cases}$$

Clearly, $\mathtt{skip}$, wait_j and $v := \mathit{expr}$ terminate after executing the first "elementary step" (an idling step in the cases $\mathtt{skip}$ and $\mathtt{wait}_j$; the evaluation of *expr* and a variable assignment in the case of $v := \mathit{expr}$). Thus, we have $s'.\mathsf{wc} = \infty$ for the successor state s' of s.

- **Probabilistic choice**: Let

$$A_l(s,s') \;=\; \begin{cases} \mathcal{D}^e[\![\mathsf{stmt}_l]\!](s,s') & : \text{if } \mathsf{stmt}_l \notin \mathsf{WStmt} \\ 1 & : \text{if } \mathsf{stmt}_l \in \mathsf{WStmt}_j,\ s' = s[\mathsf{wc} := j] \\ 0 & : \text{otherwise.} \end{cases}$$

  Then, $\mathcal{D}^e[\![\mathtt{pselect}(p_1 : \mathsf{stmt}_1, \ldots, p_m : \mathsf{stmt}_m)]\!](s,s') \;=\; \sum_{1 \le l \le m} p_l \cdot A_l(s,s')$.

[21] Thus, the function $\mathcal{D}^e$ can be viewed as the probabilistic and timed counterpart to the classical denotational semantics à la Scott that describes the input/output behaviour of sequential (non-randomized) programs.

- **Conditional commands:** $\mathcal{D}^e[\![\texttt{if } cond \texttt{ then } \mathsf{stmt}_1 \texttt{ else } \mathsf{stmt}_2]\!](s, s')$

$$= \begin{cases} \mathcal{D}^e[\![\mathsf{stmt}_1]\!](s,s') & : \text{if } [\![cond]\!](e,s) \text{ and } \mathsf{stmt}_1 \notin \mathsf{WStmt} \\ \mathcal{D}^e[\![\mathsf{stmt}_2]\!](s,s') & : \text{if } \neg[\![cond]\!](e,s) \text{ and } \mathsf{stmt}_2 \notin \mathsf{WStmt} \\ 1 & : \text{if } s' = s[\mathsf{wc} := j] \text{ and} \\ & \quad \text{either } \mathsf{stmt}_1 \in \mathsf{WStmt}_j \wedge [\![cond]\!](e,s) \\ & \quad \text{or } \mathsf{stmt}_2 \in \mathsf{WStmt}_j \wedge \neg[\![cond]\!](e,s) \end{cases}$$

and $\mathcal{D}^e[\![\texttt{if } \ldots]\!](s,s') = 0$ in all remaining cases.

- **While-loops:** $\mathcal{D}^e[\![\texttt{while } cond \ \{\mathsf{stmt}\}]\!] = \mathit{lfp}(\Omega)$ where $\mathit{lfp}(\cdot)$ denotes the least fixed point of $(\cdot)$ of the operator $\Omega : (\mathit{Eval}(V \cup \{\mathsf{wc}\})^2 \to [0,1]) \to (\mathit{Eval}(V \cup \{\mathsf{wc}\})^2 \to [0,1])$ which is defined as follows.[22]

$$\Omega(f)(s,s') = \begin{cases} \mathcal{D}^e[\![\mathsf{stmt}]\!](s,s') + \sum_{t \in Exit} \mathcal{D}^e[\![\mathsf{stmt}]\!](s,t) \cdot f(t,s') \\ \qquad : \text{if } [\![cond]\!](e,s),\ \mathsf{stmt} \notin \mathsf{WStmt} \text{ and } s'.\mathsf{wc} \neq \infty \\ \sum_{t \in Exit} \mathcal{D}^e[\![\mathsf{stmt}]\!](s,t) \cdot f(t,s') \\ \qquad : \text{if } [\![cond]\!](e,s),\ \mathsf{stmt} \notin \mathsf{WStmt} \text{ and } s'.\mathsf{wc} = \infty \\ 1 \qquad : \text{if } s.V = s'.V \text{ and} \\ \qquad\ \ \text{either } \neg[\![cond]\!](e,s) \ \wedge\ s'.\mathsf{wc} = \infty \\ \qquad\ \ \text{or } [\![cond]\!](e,s) \ \wedge\ s'.\mathsf{wc} = j \ \wedge\ \mathsf{stmt} \in \mathsf{WStmt}_j \end{cases}$$

and $\Omega(f)(s,s') = 0$ in all other cases.

[22] Note that, for all $s,\ t,\ s' \in \mathit{Eval}(V \cup \{\mathsf{wc}\})$ there exist constants $a_{s,t},\ b_{s,s'} \geq 0$ such that $\Omega(f)(s,s') = \sum_t a_{s,t} \cdot f(t,s') + b_{s,s'}$. Here, t ranges over all evaluations for $V \cup \{\mathsf{wc}\}$. For instance, $a_{s,t} = \mathcal{D}^e[\![\mathsf{stmt}]\!](s,t)$ if $[\![cond]\!](e,s)$ and $\mathsf{stmt} \notin \mathsf{WStmt}$, $a_{s,t} = 0$ and $b_{s,s'} = 1$ if $\neg[\![cond]\!](e,s)$ and $s'.\mathsf{wc} = \infty$. This yields the continuity of Ω with respect to the elementwise ordering $f \leq f'$ iff $f(s,s') \leq f'(s,s')$ for all $s,\ s' \in \mathit{Eval}(V \cup \{\mathsf{wc}\})$ on the function space $\mathit{Eval}(V \cup \{\mathsf{wc}\})^2 \to [0,1]$. Tarski's fixed point theorem yields the existence of a least fixed point.

- **Sequential composition:** $\mathcal{D}^e[\![\mathsf{stmt}_1;\mathsf{stmt}_2]\!](s,s')$

$$= \begin{cases} \mathcal{D}^e[\![\mathsf{stmt}_1]\!](s,s') \ + \ \mathcal{D}^e[\![\mathsf{stmt}_1]\!](s,s'[\mathsf{wc} := \infty]) \\ \qquad\qquad : \text{if } \mathsf{stmt}_2 \in \mathsf{WStmt}_j \text{ and } s'.\mathsf{wc} = j \\ \mathcal{D}^e[\![\mathsf{stmt}_1]\!](s,s') : \text{if } \mathsf{stmt}_2 \in \mathsf{WStmt}_j \text{ and } s'.\mathsf{wc} \neq j \\ \mathcal{D}^e[\![\mathsf{stmt}_1]\!](s,s') \ + \ \sum_{t\in Exit} \mathcal{D}^e[\![\mathsf{stmt}_1]\!](s,t)\cdot\mathcal{D}^e[\![\mathsf{stmt}_2]\!](t,s') \\ \qquad\qquad : \text{if } \mathsf{stmt}_2 \notin \mathsf{WStmt} \text{ and } s'.\mathsf{wc} \neq \infty \\ \sum_{t\in Exit} \mathcal{D}^e[\![\mathsf{stmt}_1]\!](s,t)\cdot\mathcal{D}^e[\![\mathsf{stmt}_2]\!](t,s') \\ \qquad\qquad : \text{if } \mathsf{stmt}_2 \notin \mathsf{WStmt} \text{ and } s'.\mathsf{wc} = \infty \end{cases}$$

  and $\mathcal{D}^e[\![\mathsf{stmt}_1;\mathsf{stmt}_2]\!](s,s') = 0$ in all remaining cases.

We give an informal explanation for the definition of $\mathcal{D}^e[\![\ldots]\!]$ for the probabilistic choice operator and while-loops. The arguments for conditional commands and sequential composition are similar and omitted here. In some cases we refer to the transition relation $\leadsto^e$ which describes the effect of the first commands ("elementary steps") of the extended statements. The exact definition of $\leadsto^e$ (which can be given in the SOS-style as in Figure 1) is omitted here.

Probabilistic choice: If $\mathtt{pselect}(\ldots)$ is a substatement of some well-formed extended statement then there is at most one index l where wait_j occurs in stmt_l. If there is no index l where $\mathsf{wait}_{s'.\mathsf{wc}}$ occurs in stmt_l then $\mathcal{D}^e[\![\mathtt{pselect}(\ldots)]\!](s,s') = 0$ (because s' cannot be reached from s). Now suppose that $s'.\mathsf{wc} = j$ and that wait_j occurs in stmt_l but not in any other of the extended statements stmt_i. (Thus, $A_i(s,s') = 0$ if $i \neq l$.) If wait_j is the first command of stmt_l then

$$\langle \mathtt{pselect}(\ldots, p_l : \mathsf{wait}_j;\mathsf{stmt},\ldots),\sigma\rangle \ \leadsto^e_{p_l} \ \langle \mathsf{wait}_j;\mathsf{stmt},\sigma\rangle.$$

and $\mathcal{D}^e[\![\mathtt{pselect}(\ldots)]\!](s,s[\mathsf{wc} := j]) = p_l$. If stmt_l does not start with a wait command (i.e. $\mathsf{stmt}_l \notin \mathsf{WStmt}$, $A_l(s,s') = \mathcal{D}^e[\![\mathsf{stmt}_l]\!](s,s')$) then the probability for s to reach s' with one time step when executing $\mathtt{pselect}(\ldots)$ is the same as for reaching s' from s when executing stmt_l under the condition that the outcome of resolving the probabilistic choice is stmt_l.

While-loops: If $[\![cond]\!](e,s)$ is wrong then the while-loop immediately terminates, i.e. $\langle \mathtt{while}\ cond\ \{\mathsf{stmt}\}, s.V\rangle \ \leadsto^e_1 \ \langle \mathtt{exit}, s.V\rangle$ which is reflected in the definition

$$\mathcal{D}^e[\![\mathtt{while}\ \ldots]\!](s,s') \ = \ \begin{cases} 1 : \text{if } s' = s[\mathsf{wc} := \infty] \\ 0 : \text{otherwise.} \end{cases}$$

Next we assume that $[\![cond]\!](e,s)$ is true. Then, we have the transition

$$\langle \mathtt{while}\ cond\ \{\mathsf{stmt}\}, s.V\rangle \ \leadsto^e_1 \ \langle \mathsf{stmt};\mathtt{while}\ cond\ \{\mathsf{stmt}\}, s.V\rangle.$$

If stmt starts with the wait command wait_j (i.e. $\mathsf{stmt} \in \mathsf{WStmt}_j$) then we get

$$\mathcal{D}^e[\![\texttt{while} \ \ldots]\!](s, s') \ = \ \begin{cases} 1 : \text{if } s' = s[\mathsf{wc} := j] \\ 0 : \text{otherwise} \end{cases}$$

Now we assume that the first command of stmt is not a wait command (i.e. $\mathsf{stmt} \notin \mathsf{WStmt}$). Let $t.\mathsf{wc} = \infty$ (i.e. $t \in \mathit{Exit}$). Then, $\mathcal{D}^e[\![\mathsf{stmt}]\!](s,t)$ is the probability for s to terminate in t within one time step when executing stmt. Hence,

$$\sum_{t \in \mathit{Exit}} \mathcal{D}^e[\![\mathsf{stmt}]\!](s,t) \cdot \mathcal{D}^e[\![\texttt{while} \ \ldots]\!](t, s')$$

denotes the probability for s to reach s' within one time step where the body stmt of the while-loop is executed at least once without passing any wait command.

First, let $s'.\mathsf{wc} = \infty$. The while-loop only terminates in s' when $[\![\mathit{cond}]\!](e, s')$ is wrong. Thus, each execution of $\texttt{while} \ \ldots$ that starts in s and terminates in state s' passes a state $t \in \mathit{Exit}$ such that the execution of the while-loop, when (re-)started in t, terminates in s'. Thus, if $[\![\mathit{cond}]\!](e, s)$, $\mathsf{stmt} \notin \mathsf{WStmt}$ and $s'.\mathsf{wc} = \infty$:

$$\mathcal{D}^e[\![\texttt{while} \ \ldots]\!](s, s') \ = \ \sum_{t \in \mathit{Exit}} \mathcal{D}^e[\![\mathsf{stmt}]\!](s,t) \cdot \mathcal{D}^e[\![\texttt{while} \ \ldots]\!](t, s')$$

Now we assume that $s'.\mathsf{wc} = j \neq \infty$. There are two possible cases for the while-loop to reach s' from s within one time step: either the first execution of stmt leads to s' without passing any wait command (with probability $\mathcal{D}^e[\![\mathsf{stmt}]\!](s, s')$) or the first execution of stmt leads to a state $t \in \mathit{Exit}$ without passing any wait command (with probability $\mathcal{D}^e[\![\mathsf{stmt}]\!](s, t)$) and the execution of the while-loop when (re-)started in t leads to s' within one time step (with probability $\mathcal{D}^e[\![\texttt{while} \ \ldots]\!](t, s')$). Thus, if $[\![\mathit{cond}]\!](e, s)$, $\mathsf{stmt} \notin \mathsf{WStmt}$ and $s'.\mathsf{wc} \neq \infty$ then

$$\mathcal{D}^e[\![\texttt{while} \ \ldots]\!](s, s') \ = \ \mathcal{D}^e[\![\mathsf{stmt}]\!](s, s') \ + \ \sum_{t \in \mathit{Exit}} \mathcal{D}^e[\![\mathsf{stmt}]\!](s,t) \cdot \mathcal{D}^e[\![\texttt{while} \ \ldots]\!](t, s').$$

Remark: $\mathcal{D}^e[\![\mathsf{stmt}]\!](s, s')$ does not depend on the value of the wait counter in s. I.e. $\mathcal{D}^e[\![\mathsf{stmt}]\!](s, s') = \mathcal{D}^e[\![\mathsf{stmt}]\!](t, s')$ for all s, t where $s.V = t.V$. [23] ■

5.3 The wait counter graph for parallel randomized programs

We now define the wait counter graph of $\mathcal{P}$ (where $\mathcal{P} = \langle \bar{\sigma}, \mathcal{S}_1, \ldots, \mathcal{S}_k \rangle$, $\mathcal{S}_i = \langle V_i, \mathit{wstmt}_i^0 \rangle$ are as before). The states are tuples $\bar{s} = \langle s_1, \ldots, s_k \rangle$ where s_i is the local state of $\mathcal{S}_i$, $i = 1, \ldots, k$. Let wc_i denote the wait counter for $\mathcal{S}_i$. The local states s_i are evaluations for $V_i \cup \{\mathsf{wc}_i\}$, i.e. they consist of a control component $s_i.\mathsf{wc}_i$ and an interpretation $s_i.V_i$ of the variables that are under the control of $\mathcal{S}_i$. Then,

[23] In the computation of the wait counter graph, the probabilities $\mathcal{D}^e[\![\mathsf{stmt}]\!](s, s')$ are only needed for those s and stmt where $s.\mathsf{wc} = j$ and $\mathsf{stmt} \in \mathsf{WStmt}_j$.

$\mathsf{ext}(wstmt_i^0) \in \mathsf{WStmt}(V_i)$ and $s_i.\mathsf{wc}_i \in Type(\mathsf{wc}_i) = \{1, \ldots, n_i\} \cup \{\infty\}$ where n_i is the number of `wait`'s in $wstmt_i^0$.

In the local state s_i where $s_i.\mathsf{wc}_i := j$, the sequential component $\mathcal{S}_i$ has to perform the (statement that coincides to the) extended "substatement" $\mathsf{wstmt}_{i,j}$ of $\mathsf{ext}(wstmt_i^0)$ that starts with wait_j. Thus, the (one time step) transition probabilities for $\mathcal{S}_i$ in the global state $\bar{s}$ are given by $\mathcal{D}^{e_i}[\![\mathsf{wstmt}_{i,s_i.\mathsf{wc}_i}]\!](s_i, t_i)$.

The statements $\pi_j(\mathsf{stmt})$: For stmt to be a well-formed extended statement that contains the wait command wait_j, we define an extended statement $\pi_j(\mathsf{stmt})$ that represents the "logical" substatement of stmt whose first command is wait_j and that arises by unwinding the while-loops whose body contains the command wait_j. Let stmt be a well-formed extended statement that contains the command wait_j. $\pi_j(\mathsf{stmt})$ is defined by structural induction.

- $\pi_j(\mathsf{wait}_j) \;=\; \mathsf{wait}_j$
- $\pi_j(\mathtt{pselect}(p_1 : \mathsf{stmt}_1, \ldots, p_m : \mathsf{stmt}_m) = \pi_j(\mathsf{stmt}_l)$ if wait_j occurs in stmt_l
- $\pi_j(\mathsf{stmt}_1; \mathsf{stmt}_2) \;=\; \begin{cases} \pi_j(\mathsf{stmt}_1); \mathsf{stmt}_2 & : \text{if } \mathsf{wait}_j \text{ occurs in } \mathsf{stmt}_1 \\ \pi_j(\mathsf{stmt}_2) & : \text{otherwise.} \end{cases}$
- $\pi_j(\mathtt{if}\ cond\ \mathtt{then}\ \mathsf{stmt}_1\ \mathtt{else}\ \mathsf{stmt}_2) \;=\; \pi_j(\mathsf{stmt}_l)$ if wait_j occurs in stmt_l
- $\pi_j(\mathtt{while}\ cond\ \{\mathsf{stmt}\}) \;=\; \pi_j(\mathsf{stmt});\ \mathtt{while}\ cond\ \{\mathsf{stmt}\}$

Moreover, we define $\pi_\infty(\mathsf{stmt}) = \mathtt{exit}$.[24] For example, consider the statement $\mathsf{ext}(wstmt)$ of Figure 7. The extended statements $\pi_2(\mathsf{ext}(wstmt))$ and $\pi_3(\mathsf{ext}(wstmt))$ are shown in Figure 8.

wait_2;	wait_3;
$\mathtt{while}\ b \wedge \neg c\ \{$	$\mathtt{while}\ b \wedge \neg c\ \{$
$\quad\mathtt{pselect}(\frac{1}{2} : b := f\!f, \frac{1}{2} : b := tt)$;	$\quad\mathtt{pselect}(\frac{1}{2} : b := f\!f, \frac{1}{2} : b := tt)$;
$\quad\mathsf{wait}_3\ \}$;	$\quad\mathsf{wait}_3\ \}$;
$b := \neg b$	$b := \neg b$

Fig. 8. The "unfoldings" $\pi_2(\mathsf{ext}(wstmt))$ and $\pi_3(\mathsf{ext}(wstmt))$

The wait counter graph: Let $\mathcal{P} = \langle \bar{\sigma}, \mathcal{S}_1, \ldots, \mathcal{S}_k \rangle$ be as before. The *wait counter graph* for $\mathcal{P}$ is the labelled fully probabilistic process

$$WCG(\mathcal{P}) = (S_{wcg}, \mathbf{P}_{wcg}, L_{wcg}, \overline{s}_{wcg})$$

[24] Note that we require stmt to be well-formed. Thus, the command wait_j occurs exactly once in stmt. Clearly, if $wstmt \in \mathsf{WStmt}$ then $\pi_1(\mathsf{ext}(wstmt)) = \mathsf{ext}(wstmt)$. In general, $\pi_j(\mathsf{stmt})$ is not well-formed as it might contain more than one occurrence of wait_j.

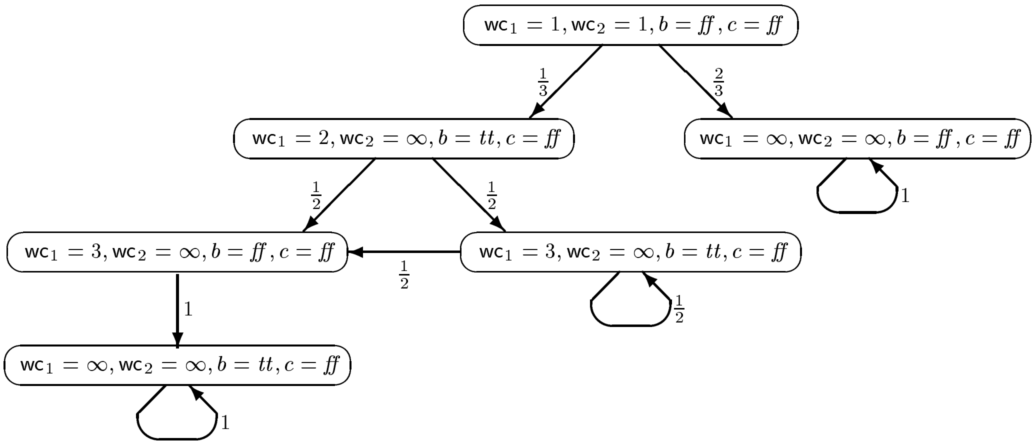

Fig. 9. The wait counter graph of $\mathcal{P}$

where $S_{wcg} = \mathit{Eval}(\mathit{Var} \cup \{\mathsf{wc}_1, \ldots, \mathsf{wc}_k\})$ and

$$\mathbf{P}_{wcg}(\langle s_1, \ldots, s_k \rangle, \langle s'_1, \ldots, s'_k \rangle) \;=\; \prod_{1 \le i \le k} \mathcal{D}^{e_i}[\![\; \pi_{s_i.\mathsf{wc}_i}(\mathsf{ext}(\mathit{wstmt}_i^0)) \;]\!](s_i, s'_i).$$

Here, e_i is the environment for $V_i \cup \{\mathsf{wc}_i\}$ that is composed by the evaluations $s_h.V_h$, $h \neq i$, i.e. $e_i.v = s_h.v$ for all $v \in V_h$, $h \neq i$. The initial state $\overline{s}_{wcg}$ is given by $\overline{s}_{wcg} = \langle s_1^0, \ldots, s_k^0 \rangle$ where $s_i^0.v = \bar{\sigma}.v$ for all $v \in V_i$ and $s_i^0.\mathsf{wc}_i = 1$. The labelling function L_{wcg} is given by $L_{wcg}(\langle s_1, \ldots, s_k \rangle) = \bigcup_{1 \le i \le k} \{a_{v, s_i.v} : v \in V_i\}$.

Example: Let $\mathcal{P} = \langle \bar{\sigma}, \mathcal{S}_1, \mathcal{S}_2 \rangle$ be as in Figure 6. I.e. we deal with two boolean variables b (under the control of $\mathcal{S}_1$) and c (under the control of $\mathcal{S}_2$) and the statements $\mathit{wstmt}_1^0 = \mathit{wstmt}$ as in Figure 2 for $\mathcal{S}_1$, $\mathit{wstmt}_2^0 = \mathtt{wait}; c := b$ for $\mathcal{S}_2$. The wait counter graph for $\mathcal{P}$ is shown in Figure 9 where we assume the initial interpretation $\bar{\sigma}.b = \mathit{ff}$ and $\bar{\sigma}.c = \mathit{ff}$. We briefly explain the outgoing transitions of the initial state $\langle \mathsf{wc}_1 = 1, \mathsf{wc}_2 = 1, b = \mathit{ff}, c = \mathit{ff} \rangle$ which stands short for the state $\bar{s} = \langle s_1, s_2 \rangle$ where $s_1.\mathsf{wc}_1 = s_2.\mathsf{wc}_2 = 1$, $s_1.b = s_2.c = \mathit{ff}$. We have to consider the environments e_1, e_2 where $e_1.b = e_2.c = \mathit{ff}$ and the evaluations σ_1, σ_2 where $\sigma_1.b = \sigma_2.c = \mathit{ff}$. For the extended statement $\pi_1(\mathsf{ext}(\mathit{wstmt}_1^0)) = \mathsf{ext}(\mathit{wstmt})$ (see Figure 7), we have:

$$\mathcal{D}^{[c=\mathit{ff}]}[\![\mathsf{ext}(\mathit{wstmt})]\!](s, s') \;=\; \begin{cases} 1/3 : \text{if } s' = s[\mathsf{wc} := 2, b := \mathit{tt}] \\ 2/3 : \text{if } s' = s[\mathsf{wc} := \infty, b := \mathit{ff}] \end{cases}$$

We have $\pi_1(\mathsf{ext}(\mathit{wstmt}_2^0)) = \mathsf{ext}(\mathit{wstmt}_2^0) = \mathsf{wait}_1 := 1; c := b$. Thus,

$$\mathcal{D}^{[b=\mathit{ff}]}[\![\mathsf{ext}(\mathit{wstmt}_2^0)]\!](s_2, s_2[c := \mathit{ff}, \mathsf{wc}_2 := \infty]) \;=\; 1.$$

For the initial state $\overline{s}_{wcg} = \langle \mathsf{wc}_1 = 1, \mathsf{wc}_2 = 1, b = \mathit{ff}, c = \mathit{ff} \rangle$ we obtain

$$\mathbf{P}_{wcg}(\overline{s}_{wcg}, \overline{t}) \;=\; \begin{cases} 1/3 : \text{if } \overline{t} = \langle \mathsf{wc}_1 = 2, \mathsf{wc}_2 = \infty, b = \mathit{tt}, c = \mathit{ff} \rangle \\ 2/3 : \text{if } \overline{t} = \langle \mathsf{wc}_1 = \infty, \mathsf{wc}_2 = \infty, b = \mathit{ff}, c = \mathit{ff} \rangle \end{cases}$$

and $\mathbf{P}_{wcg}(\overline{s}_{wcg}, \overline{t}) \;=\; 0$ in all other cases. ■

6 Consistency

In the previous section we gave a denotational semantics (the wait counter graph) of a parallel randomized program. Using iteration to approximate the least fixed operator used for while-loops, the definition of the wait counter graph can be used as an algorithm to compute the (denotational) semantics. The question arises in what way the operational semantics (the wait graph) and the denotational semantics (the wait counter graph) are related. In this section, we establish the consistency result for the operational and denotational semantics stating that the wait graph and the wait counter graph are bisimilar. [25]

To show that the wait graph and the wait counter graph are bisimilar we have to establish a bisimulation that relates the states of the wait graph and the states of the wait counter graph. First we observe that in general the wait graph and wait counter graph contains are not isomorphic (cf. Figure 6 and 9); more precisely, the wait graph might contain more states. This is due to the fact that there might be more than one extended statement that stem from the same statement. [26] We show that the relation that identifies the global state $\langle wstmt_1, \ldots, wstmt_k, \sigma_1, \ldots, \sigma_k\rangle$ of the wait graph with all states $\langle s_1, \ldots, s_k\rangle$ of the wait counter graph where $wstmt_i$ "corresponds" to $\pi_{s_i.\mathsf{wc}_i}(\mathsf{ext}(wstmt_i^0))$ and $s_i.V_i = \sigma_i$, $i = 1, \ldots, k$, is a bisimulation.
The statements $\phi(\mathsf{stmt})$: Let $\mathsf{stmt} \in \mathsf{Stmt}^+(V)$ be well-formed. We retransform stmt into a statement $\phi(\mathsf{stmt}) \in Stmt^+(V)$ by replacing all indexed wait commands wait_j by `wait`. Clearly, $\phi(\mathsf{ext}(wstmt)) = wstmt$. Let $wstmt$, $wstmt' \in WStmt^+(V)$ and $\sigma' \in Eval(V)$. We define

$$
\begin{aligned}
&States(wstmt, wstmt', \sigma')\\
&\quad = \{s \in Eval(V \cup \{\mathsf{wc}\}) : \phi\left(\pi_{s.\mathsf{wc}}(\mathsf{ext}(wstmt))\right) = wstmt', s.V = \sigma'\}.\text{[27]}
\end{aligned}
$$

Example: For the extension $\mathsf{ext}(wstmt)$ of $wstmt$ of Figure 2 (see also Figure 8), we have: $\phi(\pi_2(\mathsf{ext}(wstmt))) = \phi(\pi_3(\mathsf{ext}(wstmt))) = wstmt''$ and

$$States(wstmt, wstmt'', [b = \mathit{ff}]) = \{s_2, s_3\}$$

where the statement $wstmt''$ is as in Figure 3 and where s_2, $s_3 \in Eval(\{\mathsf{wc}, b\})$ with $s_i.\mathsf{wc} = i$ and $s_i.b = \mathit{ff}$. ■

---

[25] For the notion "consistency" see [BMC97].
[26] By dropping the indices for the wait commands, two extended statements might lead to the same statement. For instance, $\pi_2(\mathsf{ext}(wstmt))$ and $\pi_3(\mathsf{ext}(wstmt))$ (where $wstmt$ is as in Figure 2) correspond to the same statement $wstmt''$. Thus, the state $\langle wstmt'', \texttt{exit}, [b = tt], [c = \mathit{ff}]\rangle$ of the wait graph in Figure 6 is "represented" in the wait counter graph (see Figure 9) by the two states $\langle \mathsf{wc}_1 = 2, \mathsf{wc}_2 = \infty, b = tt, c = \mathit{ff}\rangle$ and $\langle \mathsf{wc}_1 = 3, \mathsf{wc}_2 = \infty, b = tt, c = \mathit{ff}\rangle$.
[27] Note that $States(wstmt, \texttt{exit}, \sigma') = \{s \in Eval(V \cup \{\mathsf{wc}\}) : s.\mathsf{wc} = \infty, s.V = \sigma'\}$.

Theorem 6.1 *Let $wstmt \in WStmt^+(V)$. Then, for all $s \in Eval(V \cup \{\mathsf{wc}\})$:*

$$\mathbf{P}^e_V(\langle \phi(\pi_{s.\mathsf{wc}}(\mathit{ext}(wstmt))), s.V\rangle, \langle wstmt', \sigma'\rangle) \;=\; \sum_{s' \in S'} \mathcal{D}^e[\![\pi_{s.\mathsf{wc}}(\mathit{ext}(wstmt))]\!](s, s')$$

where $S' = States(wstmt, wstmt', \sigma')$.

Proof (Sketch): Let $e \in Env(V)$. Using similar axioms and rules as in Figure 1, we define a transition relation $\rightsquigarrow^e \subseteq \mathsf{Stmt}(V) \times Eval(V) \times]0,1] \times \mathsf{Stmt}^+(V) \times Eval(V)$ for the extended statements over V that formalizes the stepwise baheviour. Let $\mathsf{stmt} \in \mathsf{Stmt}(V)$. We define a fully probabilistic process $\mathsf{TSB}(\mathsf{stmt}, \sigma, e) = (\mathsf{S}, \mathsf{P}, \mathsf{s}_{init})$ as follows. $\mathsf{S} = \mathsf{Stmt}^+(V) \times Eval(V) \,\cup\, \{\mathsf{s}_{init}(\mathsf{stmt}, \sigma, e)\}$ where $\mathsf{s}_{init} = \mathsf{s}_{init}(\mathsf{stmt}, \sigma, e)$ is the initial state. The transition probability matrix P is given by:

$\mathsf{P}(\langle \mathsf{stmt}', \sigma'\rangle, \langle \mathsf{stmt}'', \sigma''\rangle) = q$ iff $\langle \mathsf{stmt}', \sigma'\rangle \; \rightsquigarrow^e_q \; \langle \mathsf{stmt}'', \sigma''\rangle$ and $\mathsf{stmt}' \notin \mathsf{WStmt}^+$, $\mathsf{P}(\mathsf{s}_{init}, \langle \mathsf{stmt}', \sigma'\rangle) = q$ iff $\langle \mathsf{stmt}, \sigma\rangle \; \rightsquigarrow^e_q \; \langle \mathsf{stmt}', \sigma'\rangle$ and $\mathsf{P}(\cdot) = 0$ in all other cases. Then,

$$\mathsf{D}^e[\![\mathsf{stmt}]\!] : Eval(V) \to \big(\mathsf{WStmt}^+(V) \times Eval(V) \to [0,1]\big)$$

is given by by $\mathsf{D}^e[\![\mathsf{stmt}]\!](\sigma)(s) \;=\; Prob\,\{\pi \in Path_\omega(\mathsf{s}_{init}(\mathsf{stmt}, \sigma, e)) : last(\pi) = s\}$. Here, $Prob\{\ldots\}$ denotes the probability measure on $\mathsf{TSB}(\mathsf{stmt}, \sigma, e)$. Moreover, we put $\mathsf{D}^e[\![\mathtt{exit}]\!](\sigma)(\langle \mathtt{exit}, \sigma\rangle) = 1$ and $\mathsf{D}^e[\![\mathtt{exit}]\!](\sigma)(s) = 0$ if $s \neq \langle \mathtt{exit}, \sigma\rangle$. It can be shown that, if s, $s' \in Eval(V \cup \{\mathsf{wc}\})$ then

(I) $\quad \mathcal{D}^e[\![\mathsf{stmt}]\!](s, s') \;=\; \mathsf{D}^e[\![\mathsf{stmt}]\!](s.V)(\langle \pi_{s'.\mathsf{wc}}(\mathsf{stmt}), s'.V\rangle)$.

$TSB(\cdot)$ and $\mathsf{TSB}(\cdot)$ are viewed as *labelled* fully probabilistic processes with labels in $AP' = AP \cup Stmt^+(V)$. Here, the labelling L of $TSB(wstmt, \sigma, e)$ is given by $L(\langle stmt', \sigma'\rangle) = \{a_{v,\sigma'.v} : v \in V\} \cup \{stmt'\}$ and $L(s_{init}(wstmt, \sigma, e)) = \{a_{v,\sigma.v} : v \in V\} \cup \{wstmt\}$. Similarly, we define the labelling L of $\mathsf{TSB}(\mathsf{wstmt}, \sigma, e)$ by $\mathsf{L}(\langle \mathsf{stmt}', \sigma'\rangle) = \{a_{v,\sigma'.v} : v \in V\} \cup \{\phi(\mathsf{stmt}')\}$ and $\mathsf{L}(\mathsf{s}_{init}(\mathsf{wstmt}, \sigma, e)) = \{a_{v,\sigma.v} : v \in V\} \cup \{\phi(\mathsf{wstmt})\}$. Now we assume that $\phi(\mathsf{wstmt}) = wstmt$. It is easy to see that $TSB(wstmt, \sigma, e)$ and $\mathsf{TSB}(\mathsf{wstmt}, \sigma, e)$ are bisimilar. From this, we get

$$\text{(II)} \quad \mathbf{P}^e_V(\langle wstmt, \sigma\rangle, \langle wstmt', \sigma'\rangle) \;=\; \sum_{\mathsf{wstmt}' \in \phi^{-1}(wstmt')} \mathsf{D}^e[\![\mathsf{wstmt}]\!](\sigma)(\langle \mathsf{wstmt}', \sigma'\rangle).$$

Let $\mathcal{J} = \{j : \phi(\pi_j(\mathsf{wstmt})) = wstmt'\}$. By (II):

$$\mathbf{P}^e_V(\langle \phi(\pi_{s.\mathsf{wc}}(\mathsf{wstmt})), s.V\rangle, \langle wstmt', \sigma'\rangle) \;=\; \sum_{j \in \mathcal{J}} \mathsf{D}^e[\![\pi_{s.\mathsf{wc}}(\mathsf{stmt})]\!](s.V)(\langle \pi_j(\mathsf{stmt}), \sigma'\rangle).$$

Let $state(j, \sigma')$ be those evaluation $s' \in Eval(V \cup \{\mathsf{wc}\})$ with $s'.\mathsf{wc} = j$ and $s'.V = \sigma'$. Then, $States(stmt, wstmt', \sigma') = \{state(j, \sigma') : j \in \mathcal{J}\}$. Thus, by (I):

$$\begin{aligned}&\mathbf{P}^e_V(\langle \phi(\pi_{s.\mathsf{wc}}(\mathsf{stmt})), s.V\rangle, \langle wstmt', \sigma'\rangle)\\ &= \sum_{s' \in S'} \mathsf{D}^e[\![\pi_{s.\mathsf{wc}}(\mathsf{stmt})]\!](s.V)(\langle \pi_{s'.\mathsf{wc}}(\mathsf{stmt}), s'.V\rangle) \;=\; \sum_{s' \in S'} \mathcal{D}^e[\![\pi_{s.\mathsf{wc}}(\mathsf{stmt})]\!](s, s').\end{aligned}$$

This yields the claim. ■

Example: Let $wstmt \;=\; \mathtt{wait};\mathtt{pselect}(\frac{1}{3} : \mathtt{wait}, \frac{2}{3} : \mathtt{wait})$. Then,

$$\mathsf{ext}(\mathit{wstmt}) \;=\; \mathsf{wait}_1; \mathtt{pselect}(\tfrac{1}{3} : \mathsf{wait}_2, \tfrac{2}{3} : \mathsf{wait}_3).$$

Let $s \in \mathit{Eval}(V \cup \{\mathsf{wc}\})$, $s.\mathsf{wc} = 1$. Then, $\phi(\pi_{s.\mathsf{wc}}(\mathsf{ext}(\mathit{wstmt}))) = \mathit{wstmt}$ and

$$\mathcal{D}^e[\![\mathsf{ext}(\mathit{wstmt})]\!](s, s') \;=\; \begin{cases} 1/3 : \text{if } s' = s[\mathsf{wc} := 2] \\ 2/3 : \text{if } s' = s[\mathsf{wc} := 3] \\ 0 \quad\; : \text{otherwise} \end{cases}$$

Then, $S' \stackrel{\mathit{def}}{=} \mathit{States}(\mathit{wstmt}, \mathtt{wait}, s.V) = \{s_2, s_3\}$ where $s_j.\mathsf{wc} = j$, $s_j.V = s.V$. Thus,

$$\begin{aligned} \mathbf{P}^e_V(\langle \mathit{wstmt}, s.V\rangle, \langle \mathtt{wait}, s.V\rangle) \;&=\; 1 \;=\; \tfrac{1}{3} \;+\; \tfrac{2}{3} \\ &=\; \mathcal{D}^e[\![\mathsf{ext}(\mathit{wstmt})]\!](s, s_2) \;+\; \mathcal{D}^e[\![\mathsf{ext}(\mathit{wstmt})]\!](s, s_3). \end{aligned}$$

Note that, in the transformation of the above statement *wstmt* into an extended statement, the wait's in the two alternatives in the $\mathtt{pselect}(\cdot)$ command get different indices. Thus, when we use wait counters as control components then the state that is reached after resolving the probabilistic choice depends on whether we choose the left or right alternative. On the other hand, when we use statements as control components then from state $\langle \mathit{wstmt}, s.V\rangle$ we move to the state $\langle \mathtt{wait}, s.V\rangle$ independent on whether we choose the left or right alternative. ■

Theorem 6.2 *For each parallel randomized program $\mathcal{P}$, $\mathit{WG}(\mathcal{P}) \sim \mathit{WCG}(\mathcal{P})$.*

Proof: Let $\mathcal{P} = \langle \bar{\sigma}, \mathcal{S}_1, \ldots, \mathcal{S}_k\rangle$ be as before. Using Theorem 6.1, we get that $\{(\langle \mathit{wstmt}_1, \ldots, \mathit{wstmt}_k, \sigma_1, \ldots, \sigma_k\rangle, \langle s_1, \ldots, s_k\rangle) : s_i \in \mathit{States}(\mathit{wstmt}^0_i, \mathit{wstmt}_i, \sigma_i)\}$ is a bisimulation. ■

Example: We consider the wait graph (Figure 6) and wait counter counter graph (Figure 9) for the program $\mathcal{P} = \langle \bar{\sigma}, \mathcal{S}_1, \mathcal{S}_2\rangle$. Let R be the smallest equivalence relation on the states of the wait graph of $\mathcal{P}$ and the wait counter graph of $\mathcal{P}$ that relates the states as shown in Figure 10. Then, (as shown in the proof of Theorem

$\mathit{WG}(\mathcal{P})$	$\mathit{WCG}(\mathcal{P})$
$\langle \mathit{wstmt}, \mathtt{wait}; c := b, [b = \mathit{ff}], [c = \mathit{ff}]\rangle$	$\langle \mathsf{wc}_1 = 1, \mathsf{wc}_2 = 1, b = \mathit{ff}, c = \mathit{ff}\rangle$
$\langle \mathit{wstmt}'', \mathtt{exit}, [b = \mathit{tt}], [c = \mathit{ff}]\rangle$	$\langle \mathsf{wc}_1 = 2, \mathsf{wc}_2 = \infty, b = \mathit{tt}, c = \mathit{ff}\rangle$ $\langle \mathsf{wc}_1 = 3, \mathsf{wc}_2 = \infty, b = \mathit{tt}, c = \mathit{ff}\rangle$
$\langle \mathit{wstmt}'', \mathtt{exit}, [b = \mathit{ff}], [c = \mathit{ff}]\rangle$	$\langle \mathsf{wc}_1 = 3, \mathsf{wc}_2 = \infty, b = \mathit{ff}, c = \mathit{ff}\rangle$
$\langle \mathtt{exit}, \mathtt{exit}, [b = \mathit{tt}], [c = \mathit{ff}]\rangle$	$\langle \mathsf{wc}_1 = \infty, \mathsf{wc}_2 = \infty, b = \mathit{tt}, c = \mathit{ff}\rangle$
$\langle \mathtt{exit}, \mathtt{exit}, [b = \mathit{ff}], [c = \mathit{ff}]\rangle$	$\langle \mathsf{wc}_1 = \infty, \mathsf{wc}_2 = \infty, b = \mathit{ff}, c = \mathit{ff}\rangle$

Fig. 10. The bisimulation equivalence relation R

6.2) R is a bisimulation. ■

7 Conclusion

In this paper, we considered a specification language for parallel randomized programs $\mathcal{P}$ whose sequential components $\mathcal{S}_1, \ldots, \mathcal{S}_k$ are described in an imperative C-like language with while-loops, conditional commands and probabilistic choice. We described two semantic models for $\mathcal{P}$ that both yield a Markov chain for $\mathcal{P}$ and are based on an operational resp. denotational semantics for $\mathcal{S}_i$. Because of its declarative nature, the wait graph (the Markov chain obtained by the operational semantics) might be one that a designer has in mind. The denotational semantics is defined inductively and can easily be translated into a recursive procedure that can be implemented with multi-terminal BDDs [CFM$^+$93,BFG$^+$93]. Thus, the denotational semantics yields the theoretical foundations of a symbolic model checking tool like [Har98] that generates the wait counter graph for $\mathcal{P}$. In Theorem 6.2, we have established the bisimulation equivalence of the wait graph and wait counter graph. This guarantees that the calculations of a model checking tool (that works with the wait counter graph) are consistent with the view of the designer, provided that the underlying specification formalism is insensitive with respect to bisimulation equivalence (e.g. $PCTL^*$ [ASB$^+$95]).

It should be noted that the probabilistic one time step denotations could also be defined for (proper) statements rather than extended statements and used for the construction of a third Markov chain for a parallel randomized program $\mathcal{P}$. The resulting Markov chain would be isomorphic to the wait graph. Although the number of states in the wait graph (obtained by an operational or denotational semantics) is smaller than the number of states in the wait counter graph, its construction is not adequate for a verification tool since it uses statements as control components for the local states. [28]

References

[ASB$^+$95] A. Aziz, V. Singhal, F. Balarin, R. Brayton, A. Sangiovanni-Vincentelli: It usually works: The Temporal Logic of Stochastic Systems, Proc. CAV'95, LNCS, Vol. 939, pp 155-165, 1995.

[BFG$^+$93] I. Bahar, E. Frohm, C. Gaona, G. Hachtel, E. Macii, A. Padro, F. Somenzi: Algebraic Decision Diagrams and their Applications, Proc. ICCAD, pp 188-191, 1993.

[BCH$^+$97] C. Baier, E. Clarke, V. Hartonas-Garmhausen, M. Kwiatkowska, M. Ryan: Symbolic Model Checking for Probabilistic Processes, Proc. ICALP'97, Lecture Notes in Computer Science 1256, pp 430-440, 1997.

[28] The construction of the wait graph requires the representation of the global states $\langle wstmt_1, \ldots, wstmt_k, \ldots \rangle$ where the first k components range over certain (in general quite long) fragments of the source code for the sequential processes. Thus, the space needed for the wait graph is (in general) much more than the space complexity for the wait counter graph. Moreover, the cases where a global state of the wait graph is duplicated in the wait counter graph are rare.

[BMC97] C. Baier, M. Majster-Cederbaum: How to Interpret and Establish Consistency Results for Semantics of Concurrent Programming Languages, Fundamenta Informaticae, Vol. 29, No. 3, pp 225-256, 1997.

[Cam96] S. Campos: A Quantitative Approach to the Formal Verification of Real-Time Systems, Ph.D.Thesis, Carnegie Mellon University, 1996.

[CC92] L. Christoff, I. Christoff: Reasoning about Safety and Liveness Properties for Probabilistic Processes, Proc. 12th Conference on Foundations of Software Technology and Theoretical Computer Science, LNCS, Vol. 652, pp 342-355, 1992.

[CE81] E. Clarke, E.A. Emerson: Design and Synthesis of Synchronization Skeletons from Branching Time Temporal Logic, Proc. Workshop on Logics of Programs, LNCS, Vol. 131, pp 52-71, 1981.

[CES86] E. Clarke, A. Emerson, P. Sistla: Automatic Verification of Finite-State Concurrent Systems using Temporal Logic Specifications, ACM Trans. Programming Languages and Systems, 1(2), 1986.

[CFM+93] E. Clarke, M. Fujita, P. McGeer, J. Yang, X. Zhao: Multi-Terminal Binary Decision Diagrams: An Efficient Data Structure for Matrix Representation, In IWLS'93: International Workshop on Logic Synthesis, Tahoe City, 1993.

[CGL94] E. Clarke, O. Grumberg, D. Long: Model Checking and Abstraction, ACM Transactions on Programming Languages and Systems, Vol. 16, pp 1512-1542, 1994.

[CY88] C. Courcoubetis, M. Yannakakis: Verifying Temporal Properties of Finite-State Probabilistic Programs, Proc. FOCS'88, pp 338-345, 1988.

[CY95] C. Courcoubetis, M. Yannakakis: The Complexity of Probabilistic Verification, J. ACM, 42 (4), pp 857-907, 1995.

[Fel68] W. Feller: An Introduction to Probability Theory and its Applications, Wiley, Ney York, 1968.

[HMP+94] G. Hachtel, E. Macii, A. Padro, F. Somenzi: Probabilistic Analysis of Large Finite State Machines, Proc. ACM/IEEE DAC'94, pp 270-275, 1994.

[Hal50] P. Halmos: Measure Theory, Springer-Verlag, 1950.

[HJ94] H. Hansson, B. Jonsson: A Logic for Reasoning about Time and Probability, Formal Aspects of Computing, Vol. 6, pp 512-535, 1994.

[Har98] V. Hartonas-Garmhausen: Probabilistic Symbolic Model Checking with Engineering Models and Applications, Ph.D.Thesis, Carnegie Mellon University, 1998.

[HCC99] V. Hartonas-Garmhausen, S. Campos, E. Clarke: ProbVerus: Probabilistic Symbolic Model Checking, Proc. ARTS'99, LNCS 1601, pp 96-110, 1999.

[LS91] K. Larsen, A. Skou: Bisimulation through Probabilistic Testing, Information and Computation, Vol. 94, pp 1-28, 1991.

[Plo81] G. Plotkin: A Structural Approach to Operational Semantics, Report DAIMI FN-19, Aarhus University, September 1981.

[VW86] M. Vardi, P. Wolper: An Automata-Theoretic Approach to Automatic Program Verification, Proc. LICS'86, pp 332-344, 1986.