Probabilities in Action Systems

Kaisa Sere* Elena A. Troubitsyna†

Abstract

Action systems combined with the refinement calculus is a powerful tool for the design of parallel and distributed systems in a stepwise manner. The basic idea behind action systems is that actions are atomic entities that are chosen for execution nondeterministically. In many situations, especially when designing control systems, some preliminary information regarding probabilities of execution of certain actions is given. Within action systems we cannot take this information into account and hence we get only a pessimistic estimation of system behavior. The formalism is based on the predicate transformer semantics.

Recently the notion of probabilistic predicate transformers has been introduced to reason about probabilistic behavior of programs as well as refinement. We propose a way of combining the probabilistic predicate transformer theory with the action systems formalism and identify a class of action systems where such merging is possible. We *embed* a probabilistic actions, i.e. actions where a choice of action is performed probabilistically, into a nondeterministic action system. Thus in a probabilistic action system some of the nondeterministic choices are replaced by probabilistic choices. We show that the introduction of probabilistic action systems still allows us to perform refinement of action systems in an ordinary way. Thus we can reason about the correctness of probabilistic tasks within the refinement calculus for action systems.

1 Introduction

Action systems is a formalism introduced by Back and Kurki-Suonio [3] to reason about behavior of parallel and distributed systems. The basic entities of action systems are *actions* that can take place in the system. The actions are atomic, i.e. an execution of any action cannot be interfered by the other actions of the considered action system. The requirement of atomicity allows us to claim that a parallel execution of an action system establishes the same result as a sequential execution. Hence it is possible to apply *the refinement calculus* [1, 4] for stepwise refinement of parallel systems.

Within refinement of the action systems we can, for instance, reason about the total correctness of a parallel program. Parallelism is introduced by superposition of action systems and refining of atomicity of actions [4].

The nondeterninistic choice of an action to be executed is one of the basic concepts of the action systems. In many cases we posses some additional information regarding system behavior. Many regularities in the system behavior are expressed by defining of the probabilities of certain events. Using of this information within the action systems formalism is cumbersome. On the other hand, the well-developed formalism is very attractive for reasoning about parallel program. The idea of a compromise solution came after a recent introduction of the probabilistic predicate transformer notion [6, 7]. To our knowledge there is one similar formalism, the extension of the UNITY formalism to analyse the probabilistic behavior of parallel programs [8]. In UNITY the actions are more restrictive than ours, consisting only of a multiple assignment statement.

*Åbo Akademi University, Department of Computer Science, FIN-20520 Turku, Finland, e-mail: Kaisa.Sere@abo.fi

†Turku Center for Computer Science, Lemminkäisenkatu 14, FIN-20520 Turku, Finland, e-mail: etroubit@abo.fi

In this paper we present the way of embedding probabilistic information in the action system frame. We start by describing the basis of the action system formalism and the probabilistic predicate transformer notion. Then we present the basic idea of the transition from the nondeterministic system to the system with the probabilistic behavior. As the next step, we demonstrate an application of the suggested approach to the analysis of a system with a critical section. The discussion of the obtained results and the perspectives of this research are analysed in the conclusion.

2 Action systems

An action system statement has the following form:

$$\mathcal{A} = \texttt{begin var } x := x_0; \textbf{do } A1 \,[\!]\, \ldots \,[\!]\, A_m \textbf{ od end} : z \tag{2.1}$$

We denoted the *local variables* of action system A as x, *initialized local variables* as x_0, the *global variables* as z, and $A1$,...,A_m are the *actions* of A.

Each action has the form

$$A_i = g_i \rightarrow S_i \tag{2.2}$$

Here g_i is the *guard* of the action and S_i is the *statement* (or *body*) of the action. The guard of action A is denoted by gA and the statement of it by sA. Thus action A can be written in the following way $A{=}gA \rightarrow sA$. The body of an action can be an arbitrary statement. It may terminate or not. The local and global variables of action system are distinct. They form the *state variables* y, $y{=}x \cup z$. The set of state variables accessed in action A is denoted vA.

An action system describes the system behavior on the global level. If described system does not terminate then so does corresponding action system. The execution of action system terminates when no action is enabled anymore.

Let us consider an execution of action system. As we already mentioned an action system can be implemented in the parallel manner. Parallel execution of actions A_i and A_j is possible if these actions do not have common state variables ($vA_i \cap vAj = \emptyset$) and both are enabled. Obviously the parallel execution of actions under the indicated restrictions produces exactly the same result as a sequential execution.

To reason about *the correctness* of a program we consider the program as *a predicate transformer*. A predicate transformer is a function from *predicates* to predicates. Suppose that S is the set of states which a program can move between. A predicate is a set of such states. The predicates are partially ordered by inclusion: one predicate *implies* another if it is contained in the other. A predicate over final states of program is denoted as *postcondition*. A predicate over initial states is denoted as *a precondition*. The weakest precondition indicates all the initial states starting from which the program is guaranteed to terminate in a state satisfying the postcondition.

The standard predicate-transformer semantics is defined as

$$\mathcal{D}S := \mathbf{P}S \rightarrow \mathbf{P}S$$

where $\mathcal{D}S$ is the function space.

The notion of correctness used here is *total correctness*. It means that a program *prog* must terminate in the final state Q whenever it starts in the initial state satisfying $wp(prog, Q)$.

Total correctness is generalized in the *Refinement Calculus* to a notion of *correctness preserving refinement* between programs. A refinement preserves *the correctness of a program*. A *refinement* ordering $\sqsubseteq$ is derived pointwise from the ordering on predicates. In terms of weakest preconditions, the refinement $prog \sqsubseteq prog'$ between *prog* and $prog'$ holds iff

$$(\forall Q.wp(prog, Q) \subseteq wp(prog', Q))$$

3 Probabilistic Predicate Transformers

In real life very often designer is not interested in the total correctness of a program but rather in getting a certain result with a certain defined probability. Randomized algorithms compromise absolute correctness for the sake of performance. For example, when incorrect behavior of a control system does not lead to a disaster it is possible to have the vanishingly small probability of incorrect behavior as well. In all these cases the system is considered correct which is definitely not the case if we stay within the standard predicate transformer model.

To reason about the probabilistic behavior the notion of standard predicate was generalized by Morgan et.al. [6]. Let us sketch the way of doing this. It is known that standard predicates can be defined as functions from states to $\{0,1\}$ with pointwise $\leq$ -ordering. Probabilistic predicates generalize this representation: a probabilistic predicate is defined as a function from states to the non-negative reals $\mathbf{R}_{\geq}$. Taking this under consideration a probabilistic program can be considered as function from a probabilistic postcondition to a probabilistic weakest precondition.

Following [7] we use the square brackets to show that we are considering probabilistic predicates. In this case $[true]$ and $[false]$ are the constant functions 1 and 0 over the program variables. The standard logical operations on the predicates have the correspondence with arithmetic operations as indicated in Fig.1 (There all the arithmetic operations are applied pointwise).

logic	arithmetic
$[\neg Q]$	$1 - [Q]$
$[Q \vee Q\prime]$	$[Q] \sqcup [Q\prime]$, or
	$[Q] + [Q\prime]$ if $Q, Q\prime$ disjoint
$[Q \wedge Q\prime]$	$[Q] \sqcap [Q\prime]$, or
	$[Q] \times [Q\prime]$
$Q \mathrel{\equiv\!\!>} Q\prime$	$[Q] \leq [Q\prime]$ in all states
$Q \equiv Q\prime$	$[Q] = [Q\prime]$ in all states
$Q \mathrel{<\!\!\equiv} Q\prime$	$[Q] \geq [Q\prime]$ in all states

Figure 1. *Correspondence between logical and arithmetical operations*

More formally the space of probabilistic predicates over S is defined as

$$\mathcal{P}S := (S \to \mathbf{R}_{\geq}, \mathrel{\equiv\!\!>}),$$

where $\mathbf{R}_{\geq}$ denote the non-negative reals, and the entailment relation $\mathrel{\equiv\!\!>}$ is inherited pointwise from the normal $\leq$ ordering on $\mathbf{R}_{\geq}$. The probabilistic predicate transformer model for programs is

$$\mathcal{J}S := (\mathcal{P}S \to \mathcal{P}S, \sqsubseteq),$$

where the *refinement* order $\sqsubseteq$ is derived pointwise from the entailment on $\mathcal{P}S$.

In a probabilistic program the statement *probabilistic choice* is introduced. This statement looks as follows:

$$prog \ {}_P\!\oplus \ prog'$$

The statement $prog$ is chosen for execution with probability P and the statement $prog'$ is chosen for execution with probability $1 - P$

The weakest precondition semantics is presented on the Fig.2. Probabilistic choice is considered as a *deterministic* statement. A program is deterministic if it is free of demonic non-determinism unless it aborts.

$$
\begin{array}{rcl}
wp(\mathbf{abort}, Q) & := & 0 \\
wp(\mathbf{skip}, Q) & := & \mathrm{Q} \\
wp(x := E, Q) & := & \mathrm{Q[x:=E]} \\
wp(prog; prog', Q) & := & wp(prog, wp(prog', Q)) \\
wp(\mathbf{if}\ B\ \mathbf{then}\ prog\ \mathbf{else}\ prog', Q) & := & [B] \times wp(prog, Q) + [\neg B] \times wp(prog', Q) \\
wp(prog \parallel prog', Q) & := & wp(prog, Q) \sqcap wp(prog', Q) \\
wp(prog_P \oplus prog', Q) & := & P \times wp(prog, Q) + (1 - P) \times wp(prog', Q) \\
(\mu X \dot{\mathcal{C}}) & := & \text{least fixed point of } cntx : \mathcal{J}S \to \mathcal{J}S, \\
 & & \text{defined } wp(\mathcal{C} = cntx.(wp(X)).
\end{array}
$$

Here $\sqcap$ denotes a pointwise minimum between predicates.

Figure 2. *The weakest precondition semantics for probabilistic program*

The following expression can be applied to calculate the weakest precondition of a loop if the loop has a deterministic body.

$$wp(\mathbf{do}\ B \to body\ \mathbf{od}, Q) \equiv \sum_{i=1}^{\infty} h_i.([\neg B] \times Q), \tag{3.1}$$

where $h.\gamma := [B] \times wp(body, \gamma)$.

4 Deterministic action system

Suppose that we specified the behavior of some system within the action systems formalism. To make a probabilistic analysis of the system behavior we should transform the action system statement (2.1) to a statement which can be analysed applying the formulae of section 3. The syntactical form of the action systems enforces us to write the following statement to perform probabilistic analysis of system

$$\mathbf{do}\ A1\ {}_{p_1}\oplus A2\ {}_{p_2}\oplus\ \ldots\ \oplus A_n\ \mathbf{od} \tag{4.1}$$

where $A_i, i = 1..n$ are the actions in the form (2.2). We claim that the behavior of the system depicted by the statement (4.1) must correspond to the behavior of the initial non-deterministic action system (2.1). This requirement can be applied to determine the meaning of the probabilities $p_i, i = 1..n$.

Let us make the following observation. Suppose that actions are chosen for execution regardless at their enabledness. The choice of an unenabled action is equivalent to **skip**. If we consider infinitely many iterations then this situation is not constraining. But usually only finite number of iterations is at interest. Moreover, the number of iterations might be restricted by a considered final condition. Therefore if by chance the system chooses the unenabled action on a certain iteration then the system consumes the iteration without changing the state space. It is clear that in this case the behavior of the statement (4.1) is not equivalent to the behavior of the initial statement (2.1).

The previous discussion leads us to the idea of the introduction of a probabilistic choice between enabled actions. It means that the probabilities introduced in the statement (4.1) must indicate the chance to have a corresponding action enabled.

Let us give the short example. The statement

$$\mathbf{do}\ A1\ {}_{0.5}\oplus\ {}_{0.5}\ A2\ \mathbf{od} \tag{4.2}$$

is equivalent to the statement

$$\mathbf{do}\ sA1\ {}_{0.5}\oplus\ {}_{0.5}\ sA2\ \mathbf{od} \tag{4.3}$$

Thus on each iteration step either the environment or the system by itself makes the probabilistic choice

$$gA1, gA2 := true, false \ {}_{0.5}\oplus_{0.5}\ gA1, gA2 := false, true \tag{4.4}$$

and the system reflects a certain choice by the execution of the corresponding action body as shown in (4.3). It means that we take the point of view that as soon as the action is chosen it is enabled.

In the general case the probabilities of the state in which the guard is true is distinct from the probability of the state in which decision about execution of a certain body is done. To demonstrate this issue we observe the following situations. For brevity we consider a probabilistic choice between two actions, i.e. $n = 2$. We investigate the expression

$$\textbf{do } A1 \ {}_{p_1}\oplus_{p_2}\ A2 \textbf{ od} \tag{4.5}$$

We consider two cases.

1. $gA1 \wedge gA2 =$ **false** *in all states.* (Here gA_i is the guard of action A_i). In this case the actions are purely excluded. An execution of the body of any of them takes place if and only if *the only* guard of corresponding action is true. In this case the sum of probabilities is equal to unit and the probabilistic choice of the actions can be replaces by probabilistic choice of their body. It means that we can rewrite the initial statement in the following form:

$$\begin{array}{ll}\textbf{do} & \neg Q \ \rightarrow \\ & sA1 \ {}_{p_1}\oplus_{p_2}\ sA2 \\ \textbf{od} & \end{array} \tag{4.6}$$

where Q is a predicate which is intended to be established by the probabilistic counterpart of the considered action system either at termination or at a certain step of computation (if the considered action system does not terminate).

2. $gA_1 \wedge gA_2 =$ **true** *in some states.* In this situation the sum of the probabilities p_1 and p_2 can be greater than unit. It is caused by the fact that the probability of getting both actions enabled is taken into account. In this case we suggest to introduce an additional action $A1\_2$. The probability of getting this action enabled is denoted as p_{1_2}. This action should depict explicitly the situation that both actions are enabled. Then we calculate the corresponding probability as $p_{1\_2} = p_1 \times p_2$. Considering this discussion we rewrite the initial statement in the following manner

$$\begin{array}{lllll}\textbf{do} & & & & \\ & gA1 \wedge gA2 & \rightarrow & sA1 \ {}_{?}\oplus_{?}\ sA2 & \text{A1\_2} \\ {}_{p_{1\_2}}\oplus & gA1 & \rightarrow & sA1 & A1 \\ {}_{p'_1}\oplus_{p'_2} & gA2 & \rightarrow & sA2 & A2 \\ \textbf{od} & & & & \end{array} \tag{4.7}$$

The probabilities of the choice between actions are changed, besides the question about the probabilities presented in the body of the action $A1\_2$ is left open. To normalise the probabilities of choice of actions, (i.e. establish the expression $p'_1 + p'_2 + p_{1\_2} = 1$) we can calculate them on the base of the following equalities:

$$\begin{array}{c} p'_1 + p'_2 + p_1 \times p_2 = 1 \\ \dfrac{p'_1}{p'_2{}'} = \dfrac{p_1}{p_2} \end{array} \tag{4.8}$$

To answer the second question let us observe the following facts:
a). The probabilities of execution of the bodies p_{sA1} and p_{sA2} form the full group of events (i.e. the sum of probabilities must be equal to unit).
b). The probability of execution of a body should respect the relation of the probabilities of the states where the guards of the corresponding actions hold.

Thus we get the following equations to calculate the probabilities p_{sA1} and p_{sA2}:

$$p_{sA1} + p_{sA2} = 1$$

$$\frac{p_{sA1}}{p_{sA2}} = \frac{p_1}{p_2} \qquad (4.9\ a)$$

In the general case when a choice from n actions is performed we get the following system of equations:

$$\begin{array}{rcl} p_{sA1} + p_{sA2} + \cdots + p_{sA\ n} & = & 1 \\ \frac{p_{sA1}}{p_{sA2}} & = & \frac{p_1}{p_2} \\ \frac{p_{sA2}}{p_{sA3}} & = & \frac{p_2}{p_3} \\ \vdots & & \\ \frac{p_{sA\ n-1}}{p_{sA\ n}} & = & \frac{p_{n-1}}{p_n} \end{array} \qquad (4.9\ b)$$

Now all necessarily probabilities are calculated and we get rid of the guards obtaining the following statement

$$\begin{array}{lll} \textbf{do } \neg Q & & \\ & S_1 \ {}_{p_{sA1}}\!\oplus\!{}_{p_{sA2}}\ S_2 & \text{A1\_2} \\ {}_{p_{1\_2}}\!\oplus & S_1 & A1 \\ {}_{p'_1}\!\oplus & S_2 & A2 \\ \textbf{od} & & \end{array} \qquad (4.10)$$

The application of the formula (3.1) for the calculation of the weakest precondition of a loop with deterministic body as well as the formula for the weakest precondition of probabilistic choice (Fig.2) leads to the calculation of the weakest precondition of a deterministic action system.

5 Example

Let us consider the following system. We have two jobs: *job1* and *job2*. They request for a shared *resource*. Sometimes they ask for the resource one after the other and sometimes requests for the resource arise at the same time. In case only one of the jobs applies for the resource it always gets it. Any successful resource request is accompanied by discharging of the executed request and sending *respond* confirming the successful transaction. If both jobs request for the resource simultaneously then the following sequence of manipulations is performed:

- The system gives the resource to one of the jobs (the choice of the lucky job is non-deterministic).
- The system increments the counter of mistakes of the job which does not get the resource.
- The system sends the positive and negative responds to the lucky and unlucky jobs correspondingly.
- The system discards both requests.

The work of the system terminates when both counters of mistakes exceed their thresholds.

To describe the system behavior as an action system it is necessary to determine the list of essential actions which can take place in the system. The designed list of actions looks as follows:

- The action *ENV* simulates the arising of requests for the resource.
- The actions *A1* and *A2* model the request for the resource and the obtaining of the resource by one of the jobs *job1* and *job2* correspondingly.
- The action *A3* depicts the requests of the resource issued by both actions simultaneously.

In order to make one more step towards the formal specification of our task within the action system formalism we have to introduce variables and formulate the guards and the bodies of the actions in the terms of the introduced variables. The analysis of the task description shows that we have the following entities: the applications for the resource issued by *job 1* or *job2* (the variables

a1 and *a2* correspondingly), the responds confirming successful transactions (the variables *r1* and *r2*), the counters of mistakes of each job (the variables *c1* and *c2* respectively) and the values of the thresholds for *job 1* and *job2* (the global variables *tr1* and *tr2*).

We assume that in the initial state the jobs are not requesting for the resource and the counters are set in zero. The operations of establishing such initial state are presented in the initialization *INIT*.

The performed analysis leads us to the development of action system $\mathcal{A}$ below:

$$
\begin{array}{lllll}
\mathcal{A}: & & & & \\
\textbf{begin} & & & & \\
 & a1, a2, c1, c2 := false, false, 0, 0 & & & INIT \\
\textbf{do} & & & & \\
 & \neg a1 \wedge \neg a2 \wedge \neg(c1 \geq tr1 \wedge c2 \geq tr2) & \rightarrow & a1 := true \parallel a2 := true \parallel & \\
 & & & a1, a2 := true, true & ENV \\
[\!] & a1 \wedge \neg a2 \wedge \neg(c1 \geq tr1 \wedge c2 \geq tr2) & \rightarrow & a1, r1 := false, true & A1 \\
[\!] & a2 \wedge \neg a1 \wedge \neg(c1 \geq tr1 \wedge c2 \geq tr2) & \rightarrow & a2, r2 := false, true & A2 \\
[\!] & a1 \wedge a2 \wedge \neg(c1 \geq tr1 \wedge c2 \geq tr2) & \rightarrow & a1, a2, r1, r2, c2 := false, & \\
 & & & false, true, false, c2 + 1 [\!] & \\
 & & & a1, a2, r1, r2, c1 := false, & \\
 & & & false, false, true, c1 + 1 & A3 \\
\textbf{od} & & & & \\
\textbf{end} & & & &
\end{array}
\tag{5.1}
$$

In many cases the schedule of the request arising is known or can be established after observations. Suppose that such information is given. Now we know the probability of requesting for the resource by each of the jobs. The probability both jobs requesting the resource simultaneously is easy to calculate. Hence, we know the probability of enabledness of the actions *A1*, *A2* and *A3*. These probabilities are provided from the environment, hence the environment is presented in the system *indirectly*. We consider the action *ENV* as the external environment (the probabilities provider) and introduce the probabilities in the action system (5.1). We suppose that on each step the environment chooses one of the actions for execution preserving the values of the probabilities.

To depict this fact we get rid of the action *ENV* and introduce probabilities in the action system $\mathcal{A}$. The preliminary result of embedding the probabilistic information is the action system (5.2).

$$
\begin{array}{ll}
\mathcal{A}': & \\
 & \textbf{do} \\
 & \quad A1^{pr}{}_{P1} \ \oplus \ A2^{pr}{}_{P2} \ \oplus \ A3^{pr} \\
 & \textbf{od}
\end{array}
\tag{5.2}
$$

We use the subscript *pr* to emphasize that the action with the probabilistic behavior can differ from the original action. In our case the body of the action $A3$ should be transformed to the deterministic form. To make it so, we distribute the probabilistic behavior in the body of the action $A3$ as has been shown in the section 4. Using formula (4.9 *a*) we can rewrite the action system $\mathcal{A}'$ as follows:

$$
\begin{array}{llllll}
\mathcal{A}': & & & & & \\
\textbf{do} & & & & & \\
 & a1 \wedge \neg a2 \wedge \neg(c1 \geq tr1 \wedge c2 \geq tr2) & \rightarrow & a1, r1 := false, true & A1^{pr} & \\
{}_{P1}\oplus & a2 \wedge \neg a1 \wedge \neg(c1 \geq tr1 \wedge c2 \geq tr2) & \rightarrow & a2, r2 := false, true & A2^{pr} & \\
{}_{P2}\oplus & a1 \wedge a2 \wedge \neg(c1 \geq tr1 \wedge c2 \geq tr2) & \rightarrow & a1, a2, r1, r2, c2 := false, & & (5.3) \\
 & & & \quad false, true, false, c2 + 1 \ {}_{P2'}\oplus & & \\
 & & & a1, a2, r1, r2, c1 := false, & & \\
 & & & \quad false, false, true, c1 + 1 & A3^{pr} & \\
\textbf{od} & & & & &
\end{array}
$$

where

$$\begin{aligned} P_1' &= 1 - \frac{P_2}{P_1+P_2} \\ P_2' &= \frac{P_2}{P_1+P_2} \end{aligned}$$

As the next step in the transformation of the action system (5.2), we get rid of the guards of the actions present in (5.3) in the same manner as we transformed the action system (4.7) to the statement (4.10). The guard of the loop is formed according to the purpose of the probabilistic analysis. The result of this transformation is the following statement:

$$\begin{array}{ll} \textbf{do} & \neg(c1 \geq tr1 \wedge c2 \geq tr2) \ \rightarrow \\ & (a1, r1 := false, true) \ {}_{P1}\oplus \ (a2, r2 := false, true) \ {}_{P2}\oplus \\ & (a1, a2, r1, r2, c2 := false, false, true, false, c2+1) \\ & \qquad {}_{P_1\times P_2\times P_2'}\oplus \\ & (a1, a2, r1, r2, c1 := false, false, false, true, c1+1); \\ & n \ := \ n \ + \ 1 \\ \textbf{od} & \end{array} \tag{5.4}$$

At this point we have reached the goal of the performed transformation chain, i.e. we obtained the statement (5.4) which can be analysed applying the formula (3.1). In our case, the next step in the analysis of the probabilistic behavior of the system is to estimate the probability of termination during the execution of N iterations: $[n \leq N \wedge (c1 \geq tr1 \wedge c2 \geq tr2)]$. The purpose of the analysis forms the guard of the statement (5.4).

To calculate the probability of termination of the statement (5.4) during the N iteration steps let us note the following facts. There are four types of events that can take place:
Event1: job1 gets the resource. The probability of this event equals P_1.
Event2: job2 gets the resource. The corresponding probability is equal to P_2.
Event3: job1 and job2 apply for the resource and job1 gets it. The probability of this case is $P_1 \times P_2 \times P_2'$.
Event4: job1 and job2 apply for the resource and job2 obtains it. This event happens with the probability $P_1 \times P_2 \times P_1'$.

Let $l = tr1 + tr2$. It is clear that termination can occur not earlier than on the l-th iteration. Thus referring to the formula (3.1) we can conclude that $(\forall i : 0..l-1.h_i = 0)$. The probability of termination on the l-th step calculated by means of (3.1) looks as follows

$$h_l = (P_1 \times P_2)^l \times C_l^{tr1}(P_2')^{tr2}(P_1')^{tr1}$$

where $tr1$, $tr2$ are the values of the thresholds, P_1, P_2, P_1', P_2' are the probabilities appeared in (5.3) and $C_m^k = \frac{m!}{k!(m-k)!}$

To calculate h_{l+1} we can note that the system terminates on the $l+1$-th iteration, if one of the two possible sequences of events occurred:
The last event is $Event4$ and then either $Event3$ occurs $tr2$ times and $Event1$ or $Event2$ occurs once during the first l steps of iteration
or
$Event3$ is the last event and then either $Event4$ occurs $tr1$ times and the $Event1$ or $Event2$ occurs once during the first l steps of iteration.

$$h_{l+1} = C_l^1(P_1 \times P_2 \times P_2')^{tr2}(P_1 \times P_2 \times P_1')^{tr1} \times (C_{l-1}^{tr2}(1 - P_1 \times P_2 \times P_2') + C_{l-1}^{tr1}(1 - P_1 \times P_2 \times P_1'))$$

Proceeding in this manner we get on the N-th step

$$h_N = C_{N-1}^{N-l}(P_1 \times P_2 \times P_2')^{tr2}(P_1 \times P_2 \times P_1')^{tr1} \times (C_{l-1}^{tr2}(1 - P_1 \times P_2 \times P_2')^{N-l} + C_{l-1}^{tr1}(1 - P_1 \times P_2 \times P_1')^{N-l})$$

An application of the formula (3.1) for the calculation of the weakest preconditions of loop leads to the following result:

$$\begin{array}{l} wp(\textbf{do} \ \ldots \ \textbf{od}) = \\ (P_1 \times P_2)^l \times P_2'^{tr2} \times P_1'^{tr1}(C_l^{tr1} + C_{l-1}^{tr2}\Sigma_{i=1}^{N-l}C_l^i(1 - P_1 \times P_2 \times P_2')^i + \\ C_{l-1}^{tr1}\Sigma_{i=1}^{N-l}C_l^i(1 - P_1 \times P_2 \times P_1')^i \end{array} \tag{5.5}$$

Formula (5.5) expresses the probability of termination occurred during the execution of N iteration steps. We obtained the required probability as a function from the number of iterations and the values of the thresholds. Hence, in case the probabilistic information is provided, it is possible to obtain a certain value of the investigated probability by manipulating the values of the thresholds and the number of steps.

6 Conclusions

In many situations the design of a system is performed in the following way: at first a system is specified in a purely nondeterministic manner. No information is available or any additional information is not taken into consideration for the abstraction purposes. As the next design step, either additional information becomes available as a result of observations or initially provided information is taken into account. The action system formalism is very convenient to specify the system behavior abstractly. To put the additional information in the abstract specification it is necessary to build a bridge between the demonic nondeterminism and the probabilistic nondeterminism. We tried to describe a way of transformation from the abstract specification within the action system frame to the probabilistic specification. Using probabilistic information we show the chain of transformations from the action system to the deterministic statement. The basis of this transformation is the distribution of the probabilistic information to get rid of the nondeterminism. The idea of getting rid of the demonic nondeterminism is similar to the Rao concept [8]. Our approach employs the following point of view: the nondeterministic choice between actions is substituted by the probabilistic choice of the actions bodies. The formulas for the calculation of the corresponding probabilities are suggested. We consider the probability of action choice as the probability of action enabledness.

The described approach was applied to the analysis of a system with a critical section. Initially we specified the task as an action system. Then, applying the suggested rules we transformed it to the deterministic system. As a result we obtained a function depicting the probabilistic behavior of the system. This expression is formulated in terms of the system parameters. This result can be used to correct the parameters of the system to reach required behavior.

The probabilistic predicate transformer model restricts the merging of the probabilistic and nondeterministic behavior. As the next step, we are going to investigate the possible constrains, an application of which will allow to analyse the systems with combined behavior.

References

[1] R. J. R. Back. A calculus of refinement for program derivation. *Acta Informatica,* 25:593-624,1988.

[2] R. Back. *Correctness Preserving Program Refinements: Proof Theory and Applications,* volume 131 of *Mathematical Center Tracts.* Mathematical Centre, Amstredam, 1980

[3] R. J. R. Back and R. Kurki-Suonio. Decentralization of process nets with centralized control. In *Proc. of the 2nd ACM SIGACT–SIGOPS Symp. on Principles of Distributed Computing,* pages 131–142, 1983.

[4] R. J. R. Back and K. Sere. From modular systems to action systems. Proc. of *Formal Methods Europe'94,* Spain, October 1994. *Lecture Notes in Computer Science.* Springer–Verlag, 1994.

[5] M. Butler, E. Sekerinski, and K. Sere. An Action System Approach to the Steam Boiler Problem. In Jean-Raymond Abrial, Egon Borger and Hans Langmaack, editors, *Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control, Lecture Notes in Computer Science* Vol. 1165. Springer-Verlag, 1996. To appear.

[6] C.C. Morgan, A.K. McIver, K. Seidel. Probabilistic Predicate Transformers. *Technical report PRG-TR-5-95,* Programming Research Group, January 1995.

[7] K. Seidel, C.C. Morgan and A.K. McIver. An introduction to probabilistic predicate transformers. *Technical report PRG-TR-6-96*, Programming Research Group, 1996.

[8] J.R.Rao Extensions of the UNITY Methodology: Compositionality, Fairness and Probability in Parallelism. *Lecture Notes in Computer Science*. Vol. 908. Springer–Verlag, 1995