PCSP# Denotational Semantics with an Application in Sports Analytics

Zhaoyu Liu[1], Murong Ma[1], Kan Jiang[1], Zhe Hou[2], Ling Shi[3], Jin Song Dong[1]

[1] National University of Singapore, Singapore
[2] Griffith University, Australia
[3] Nanyang Technological University, Singapore

Abstract. The paper introduces probabilistic CSP# (PCSP#) as a formal language for modeling probabilistic systems, emphasizing its foundational role in concurrent and parallel computation. PCSP# is an extension of communicating sequential programs (CSP#) that enables the modeling of probabilistic behaviors and uncertainties in system designs. We provide the formal syntax and a denotational semantics for PCSP# based on the UTP framework. We also introduce the Process Analysis Toolkit (PAT) as a formal modeling and verification tool for complex concurrent and probabilistic systems, particularly focusing on its extension CSP# and its variant, PCSP#. The paper explores the applications of PCSP# in sports analytics, specifically in modeling tennis matches. It demonstrates how PCSP# can be used to analyze match outcomes, develop strategies, and provide training recommendations based on past match data. Experimental results from historical data in the past decade show that formal methods applied to sports analytics yield high-performance and accurate models that are explainable and suitable for strategy and training recommendations.

Keywords: Denotational Semantics · System Modeling · Probabilistic Model Checking · Sports Analytics

1 Dedication

During his undergraduate studies, Prof. Jin Song Dong (the last author of this paper), the leader of the NUS Formal Methods group, studied Jim's book, "Software Engineering Mathematics" [47], where Jin Song encountered the Z notation [50] and was deeply impressed by its precision and coherence. Jim's prolific contributions, reflected in both papers and presentations, left a lasting imprint on Jin Song's research endeavors. Their first encounter occurred in 1996 at the FME conference in Oxford, marking the commencement of numerous subsequent meetings across various international locales. Jim consistently ranks as a top candidate for keynote addresses at major conferences organized by Jin Song, a testament to the enduring impact of his insights. Furthermore, Jin Song holds dear the cherished moments spent with Jim and their family during their early gatherings in Singapore. Jin Song and his team would like to dedicate this paper to honor the occasion of Jim's retirement from the University of York in September 2024.

2 Introduction

Communicating Sequential Processes (CSP) [14] is a formal language for describing patterns of interaction in concurrent systems. Developed by Tony Hoare in the late 1970s, CSP serves as a foundational model for studying concurrent and parallel computation. The language provides a rigorous framework for specifying the behavior of concurrent processes, emphasizing communication and synchronization between them.

In CSP, concurrent processes are represented as independent entities that communicate by sending and receiving messages over channels. These channels serve as communication pathways, enabling processes to exchange information and coordinate their activities. The language allows for the definition of complex systems by composing simpler processes and specifying their interactions through communication events. Key concepts in CSP include process behavior, communication primitives, and synchronization mechanisms. Processes in CSP exhibit sequential behavior, executing a series of steps in a defined order, while communication primitives such as channel input and output operations facilitate message passing between processes. Synchronization mechanisms, such as synchronization barriers and process synchronization, ensure that processes coordinate their activities in a coherent manner.

CSP has found applications in various domains, including concurrent programming, distributed systems, and formal verification. Its formal semantics enables rigorous analysis of system properties, such as deadlock freedom, liveliness, and safety [31]. Additionally, CSP has inspired the development of other concurrency models and programming languages, contributing to the advancement of concurrent computing theory and practice.

A prominent application of CSP is the Process Analysis Toolkit (PAT) [43], which is a comprehensive software tool designed for the formal modeling, analysis, and verification of concurrent systems. PAT provides a user-friendly environment for specifying and analyzing complex systems using formal methods. PAT supports various formal modeling languages, including CSP, allowing users to describe concurrent systems as sets of interacting processes. These processes communicate and synchronize their actions through channels, enabling the modeling of intricate systems with concurrency and communication patterns.

PAT has been widely used in academia and industry for the formal verification of concurrent systems, including communication protocols, distributed systems, and hardware designs. Its user-friendly interface and powerful analysis capabilities make it a valuable tool for researchers, engineers, and developers working in the field of concurrent and distributed computing. Notably, PAT extends the CSP language with C# to make the modeling task user-friendly for programmers, resulting in communicating sequential programs (CSP#) [41]. Moreover, PAT has various modules that extend CSP# further to variants such as Stateful Timed CSP# [42] and probabilistic CSP# (PCSP#) [44]. This paper focuses on the latter and its applications.

PCSP# allows users to specify probabilistic behaviors and uncertainties in system designs. It enables the representation of systems that exhibit stochas-

tic or non-deterministic behaviors, such as probabilistic transitions, uncertain actions, and random events. This probabilistic extension enhances the expressiveness of CSP#, enabling the modeling of a wider range of real-world systems, including stochastic processes, randomized algorithms, and probabilistic protocols. The key features include probabilistic transitions, uncertain actions and random events. PCSP# is widely used in probabilistic systems modeling, reliability analysis, fault tolerance analysis, security analysis, performance evaluation, and so on. This paper demonstrates its application from a new angle: sports analytics. We show how PCSP# is used to model a tennis match, considering a variety of actions in the game and how they transit to each other. The probabilities are mined from past match data. Such a model enables in-depth analysis of the tennis match, including match outcome prediction, strategy analysis and training recommendations.

In what follows we will describe the syntax and semantics of PCSP#, and then we will give a detailed case study on tennis analytics. Similar techniques can be easily applied to other racket sports.

3 The Probabilistic CSP# Language (PCSP#)

A PCSP# model comprises definitions of constants, shared variables, channels, and processes. Constants are defined by keyword #**define** followed by a name and a value, shared variables are declared with keyword **var** followed by a name and an initial value with global scope, and channels are declared using keyword **channel** with a name. The process is an extension of CSP# with probabilistic multi-choice. The syntax is defined as follows.

Program	::=	$PCSP\#Par^*$	– model
$PCSP\#Par$	::=	#**define** *cons* v	– constant
	|	**var** $x = v_1$	– variable
	|	**channel** ch	– channel
	|	*ProcDel*	– process
ProcDel	::=	$Proc(ParDel) = P \mid Proc = P$	– process declaration
ParDel	::=	$i \mid i, ParDel$	– parameter
P, P_i, Q	::=	$Stop \mid Skip$	– primitives
	|	$a \rightarrow P$	– event prefixing
	|	$ch!exp \rightarrow P$	– channel output
	|	$ch?m \rightarrow Proc(m)$	– channel input
	|	$e\{prog\} \rightarrow P$	– data operation prefix
	|	$[b]P$	– state guard
	|	$P \,\square\, Q \mid P \sqcap Q$	– choices
	|	$P \,;\, Q$	– sequence
	|	$P \setminus X_1$	– hiding
	|	$P \parallel_{(X_1,X_2)} Q \mid P \mid\mid\mid_{X_2} Q$	– parallel/interleaving
	|	$N \mid \mu N \cdot P$	– recursion
	|	**pcase**$\{pr_0 : P_0;\ pr_1 : P_1;\ \cdots;\ pr_k : P_k\}$	– *probabilistic multi − choices*

where *cons* is an identifier for a constant whose value is v, x is an identifier for a variable whose initial value is v_1, *ch* is an identifier for a channel, *Proc* is a process name, i is an identifier for a variable denoting a process formal parameter, P, P_i, Q range over processes, a is an action name, *exp* is an arithmetic expression, m is a bound variable, e is the name of a non-communicating event denoting sequential program *prog* which updates global shared variables, b is a Boolean expression, X_1 is a set of actions, X_2 is a set of synchronous channel inputs and outputs, N is an identifier, and pr_i is a positive integer to express the probability weight. The syntax of program *prog* can be referred to [36].

Process *Stop* communicates nothing and process *Skip* terminates successfully. Process $a \rightarrow P$ engages in action a first and then behaves as process P. Synchronous channels communicate through handshaking. Specifically, a process $ch!exp \rightarrow P$ which is ready to perform an output through *ch* will be enabled if another process $ch?m \rightarrow Proc(m)$ is ready to receive an input through the same channel *ch* simultaneously, and *vice versa*. The expression *exp* in $ch!exp$ is evaluated atomically with the occurrence of the output. Process $e\{prog\} \rightarrow P$ generates an event e, executes *prog atomically*, and after that process behaves as P. Note that global shared variables are updated in the atomic execution of the program *prog*. Process $[b]P$ waits until condition b becomes *true* and then behaves as P. The condition checking is conducted atomically with the occurrence of the first event or state transition in P. Two choices are supported, i.e., $P \,\Box\, Q$ for external choice, and $P \sqcap Q$ for internal choice. Sequential composition $P \,;\, Q$ behaves as P until P terminates and then behaves as Q. Process $P \setminus X_1$ hides all occurrences of events in X_1. In process $P \parallel_{(X_1,X_2)} Q$, P and Q run in parallel and communicate through multi-part event synchronization (i.e., common events in X_1 and synchronous channels in X_2). In contrast, in process $P \,|||_{X_2}\, Q$, P and Q run independently (except for communications through synchronous channels in X_2). Lastly, in probabilistic multi-choices process $\mathbf{pcase}\{pr_0 : P_0;\ pr_1 : P_1;\ \cdots;\ pr_k : P_k\}$, with probability $\frac{pr_i}{pr_0+pr_1+\cdots pr_k}$, the process behaves as P_i.

4 The Denotational Semantics for PCSP#

The Unifying Theories of Programming (UTP) [15] offers a mathematical framework to describe and analyze programs across various paradigms, such as imperative, concurrent and so on. For each programming paradigm, a program is modeled as a relation between its initial and subsequent (intermediate or final) states. Relations are represented as predicates over observational variables to capture all aspects of program behaviours.

In the UTP framework, theories of programming paradigms contain three essential elements: *alphabet*, *signature* and *healthiness conditions*. The *alphabet* is a set of observational variables that can be used to record external observations of the program behaviour. These variables include both inputs to and outputs from the program. Variables of initial observations are undashed, representing the state before program execution, and variables of subsequent observations are

dashed, representing the state after execution. The *signature* defines the syntax of the rules to represent the elements of a theory. The *healthiness conditions* are a selection of laws identifying valid predicates that characterise a theory. A healthiness condition is associated with observational variables in the alphabet. It is defined by an idempotent function on predicates. A healthy program represented by predicate P satisfies healthiness condition ϕ if it is a fixed point of ϕ: $P = \phi(P)$.

4.1 Semantic model

To address the challenge of designing an appropriate semantic model which can cover different paradigms like communications, shared variables and probabilistic choices for the denotational semantics of PCSP#, we blend probability with communication events and states containing shared variables. We introduce *mixed* traces to record the interactions of processes with the global environment and its associated probability.

Observational Variables The following variables are introduced in the alphabet of observations of PCSP# process behaviours. Some of them (i.e., ok, ok', $wait$, $wait'$, ref, and ref') are similar to those in the denotational semantics of CSP# [35]. The key difference is that probabilities are recorded in the traces in PCSP#.

- ok, ok': $\mathsf{Boolean}$ are used to model the stability of a process. For example, when ok is *true* before a process's execution, it indicates that the process is ready to start. When ok is *true* after a program's execution, it indicates that the process has reached a stable state.
- $wait$, $wait'$: $\mathsf{Boolean}$ distinguish the intermediate observations of waiting states from the observations of final states. For example, when $wait$ is *false*, it records that the execution of the previous process has finished and the current process may start. When $wait'$ is *false*, it records that the next observation is in a terminated state.
- ref, ref': $\mathbb{P}\, Event$ denote a set of actions and channel inputs/outputs that can be refused before or after the observation. The set $Event$ denotes all possible actions and channel input/output directions (e.g., $ch?, ch!$). An input direction $ch?$ denotes any input through channel ch, and a channel output direction $ch!$ denotes any output through channel ch. The set Act denotes all possible actions, similar to its definitions in CSP# [36].
- tr, tr': $\mathrm{seq}((\mathsf{S} \times \mathsf{S}^{\perp} \times \mathbb{P}) \cup (\mathsf{S} \times \mathsf{E} \times \mathbb{P}))$ record a finite sequence of observations on the interaction of the processes with the global environment. It is a 3-tuple consisting of pre-state, event/post-state and associated probability.
  - S is the set of all possible mappings (states), and a state $s : \mathsf{VAR} \to \mathsf{int}$ is a total function which maps global shared variables names from VAR into values of integer int. Notice that the types of variable values and channel messages are integer in our proposed semantics.

- E is the set of all possible events, including actions, channel inputs or outputs, synchronous channel communications and event τ_p which indicates the probabilistic choice selection.
- $\mathbb{P}$ is the set of probabilities with the range of [0..1].
- $\mathsf{S} \times \mathsf{S}^{\perp} \times \mathbb{P}$ is the set of 3-tuples, where each tuple consists of a pre-state recording the initial variable values before the observation, a post-state recording the final values after the observation, and its associated probability. $\mathsf{S}^{\perp} \mathrel{\widehat{=}} \mathsf{S} \cup \{\perp\}$ represents all states, where the improper state $\perp$ indicates non-termination.
- $\mathsf{S} \times \mathsf{E} \times \mathbb{P}$ denotes a set of occurring events under the pre-states with corresponding probabilities. The reason of recording the pre-state is that the value of the expression which may contain shared variables in a channel output shall be evaluated under this state. Additionally, we record the probability for the occurrence of the event.

Healthiness Conditions In PCSP#, processes satisfy the following healthiness conditions.

$\mathbf{H}_p = \mathbf{R1} \circ \mathbf{R2} \circ \mathbf{R3} \circ \mathbf{CSP1} \circ \mathbf{CSP2}$

R1 ensures that a process can only extend the record and never change the past history of the observations.

$\mathbf{R1}(P) \;=\; P \wedge tr \leq tr'$

R2 restricts that the execution of a process is independent of the history before its activation.

$\mathbf{R2}(P(tr, tr')) \;=\; \sqcap_s \, P(s, s \frown (tr' - tr))$

R3 characterises that a process cannot start if its predecessor has not finished, or otherwise, the values of all observational variables are unchanged.

$\mathbf{R3}(P) \;=\; II \lhd wait \rhd P$

where $P \lhd b \rhd Q \mathrel{\widehat{=}} b \wedge P \vee \neg b \wedge Q$ and $II \;\mathrel{\widehat{=}}\; (\neg ok \wedge tr \leq tr') \vee (ok' \wedge tr' = tr \wedge wait' = wait \wedge ref' = ref)$.

CSP1 captures that when a process is in a divergent state, it can only arbitrarily extend the trace.

$\mathbf{CSP1}(P) \;=\; (\neg ok \wedge tr \leq tr') \vee P$

CSP2 states that if an observation of a process is valid when ok' is false, then the observation should also be valid when ok' is true.

$\mathbf{CSP2}(P) \;=\; P \;;\; ((ok \Rightarrow ok') \wedge tr' = tr \wedge wait' = wait \wedge ref' = ref)$

Although PCSP# satisfies the same healthiness conditions of CSP#, observational variables tr, tr' in our semantic model record additional information for probability. We adopt the same names for the idempotent functions used in CSP# for consistency.

4.2 Semantics of Processes

We first illustrate our semantic definitions of core processes: event prefixing, synchronous channel output/input, data operation prefixing, probabilistic multi-choices, and parallel composition. These process operators are frequently used to

specify complex probabilistic systems involving channel-based communications, shared variables, and probabilistic behaviour. We further present the semantics of other process operators and refinement at the end.

Event prefixing Process $a \rightarrow P$ engages in action a first and afterwards behaves as process P. Action a occurs instantaneously, and may require simultaneous participation by more than one processes, which is captured when the process is in a waiting state $\neg wait'$, it cannot refuse this action a (represented by predicate $a \notin ref'$). When a process performs action a and terminates with its trace extended with the observation (represented by predicate $tr' = tr \frown \langle (s, a, 1) \rangle$). Note that we record the probability of the action occurrence is 1 since there is no probabilistic choice for selection.

$$a \rightarrow P \mathrel{\widehat{=}} \mathbf{H}_p \left(ok' \wedge \begin{pmatrix} a \notin ref' \wedge tr' = tr \\ \lhd wait' \rhd \\ \exists\, s \in \mathsf{S} \cdot tr' = tr \frown \langle (s, a, 1) \rangle \end{pmatrix} \right) ;\ P$$

Synchronous channel output/input PCSP# allows messages to be sent or received synchronously through channels. In the pairwise synchronisation, a synchronous channel communication $ch.exp$ can take place only if an output $ch!exp$ is enabled and a corresponding input $ch?m$ is also ready. There are two possible behaviours: when a process is waiting to communicate on channel ch (represented by $\neg wait'$), it cannot refuse any channel input over ch provided by the environment to perform a channel communication (represented by $ch? \notin ref'$) for channel output process or any channel output provided by the environment (represented by $ch! \notin ref'$) for channel input process; or a process sends the output/receives a message through ch and terminates without divergence. Note that probability recorded in the trace is 1 and the definition of semantics function $\mathcal{A}$ for arithmetic expressions can be referred to [36].

$$ch!exp \rightarrow P \mathrel{\widehat{=}} \mathbf{H}_p \left(ok' \wedge \begin{pmatrix} ch? \notin ref' \wedge tr' = tr \\ \lhd wait' \rhd \\ \exists\, s \in \mathsf{S} \cdot tr' = tr \frown \langle (s, ch!\mathcal{A}[\![exp]\!](s), 1) \rangle \end{pmatrix} \right) ;\ P$$

$$ch?m \rightarrow Proc(m) \mathrel{\widehat{=}}$$
$$\exists\, v \in \mathsf{int} \cdot \left(\mathbf{H}_p \left(ok' \wedge \begin{pmatrix} ch! \notin ref' \wedge tr' = tr \\ \lhd wait' \rhd \\ \exists\, s \in \mathsf{S}\dot{t}r' = tr \frown \langle (s, ch?v, 1) \rangle \end{pmatrix} \right) ;\ Proc(v) \right)$$

Data Operation Prefixing In process $e\{prog\} \rightarrow P$, the sequential program $prog$ is executed atomically and its observation is the updates on shared variables after the execution of all programs. If the program does not terminate (represented by $(s, \bot) \in \mathcal{C}[\![prog]\!]$), then the process enters a waiting state, and its trace is extended with the record of non-termination. On the other hand, if the evaluation succeeds and terminates, then the process terminates and the state transition is recorded in the trace. In both cases, the probability is recorded as 1 in the extended trace. Function $\mathcal{C}$ defines the semantics of the sequential

programs which can be referred to [36]. Note that the non-communicating event e is not recorded in the trace since such an event would not synchronise with other events and it is a label to denote the data operation.

$e\{prog\} \rightarrow P \mathrel{\widehat{=}}$

$$\mathbf{H}_p \left(ok' \wedge \exists\, s \in \mathsf{S} \cdot \begin{pmatrix} wait' \wedge tr' = tr \frown \langle (s, \bot, 1) \rangle \\ \lhd (s, \bot) \in \mathcal{C}[\![prog]\!] \rhd \\ \neg wait' \wedge \exists\, s' \in \mathsf{S} \cdot (tr' = tr \frown \langle (s, s', 1) \rangle \\ \wedge (s, s') \in \mathcal{C}[\![prog]\!]) \end{pmatrix} \right) ;\ P$$

Probabilistic multi-choices In process $\mathbf{pcase}\{pr_0 : P_0;\ pr_1 : P_1;\ \cdots;\ pr_k : P_k\}$, if **pcase** is activated, then it transmits to a process via event τ_p with a corresponding probability. Event τ_p denotes the probabilistic choice selection which will not synchronize with other events. The semantics covers all cases of reaching the successive states following different probability weights. For example, after the execution, the process moves into process P_0 with the probability $\frac{pr_0}{pr}$. In this case, the trace is extended with the tuple $(s, \tau_p, \frac{pr_0}{pr})$. Note that $pr = \sum_{i=0}^{k} pr_i$ is the sum of all the probability weights.

$\mathbf{pcase}\{pr_0 : P_0;\ pr_1 : P_1;\ \cdots;\ pr_k : P_k\} \mathrel{\widehat{=}}$

$$\bigvee_{i \in \{0..k\}} \left(\mathbf{H}_p \left(ok' \wedge \begin{pmatrix} tr' = tr \\ \lhd wait' \rhd \\ \exists\, s \in \mathsf{S} \dot{t} r' = tr \frown \langle (s, \tau_p, \frac{pr_i}{pr}) \rangle \end{pmatrix} \right) ;\ P_i \right)$$

Parallel Composition Probabilistic multi-choices process executes independently when running in parallel with other processes. Thus, the semantics definition of parallel composition $P \parallel_{(X_1, X_2)} Q$ in PCSP# is similar to that in CSP#. The trace execution covers three aspects: (1) common actions of P and Q require simultaneous participation, (2) synchronous channel output in one process occurs simultaneously with the corresponding channel input in the other process, and (3) other kinds of events occur independently. Note that set X_1 contains common actions and set X_2 covers all synchronous channel inputs and outputs.

$P \parallel_{(X_1, X_2)} Q \mathrel{\widehat{=}}$

$$\mathbf{H}_p \left(\begin{matrix} \exists\, 0.ok, 0.wait, \\ 0.ref, 0.tr, \\ 1.ok, 1.wait, \\ 1.ref, 1.tr \end{matrix} \cdot \begin{pmatrix} P[0.ok, 0.wait, 0.ref, 0.tr / ok', wait', ref', tr'] \wedge \\ Q[1.ok, 1.wait, 1.ref, 1.tr / ok', wait', ref', tr'] \wedge \\ M(X_1, X_2) \end{pmatrix} \right)$$

where $M(X_1, X_2) \mathrel{\widehat{=}}$

$$\begin{pmatrix} (ok' = 0.ok \wedge 1.ok) \wedge \\ (wait' = 0.wait \vee 1.wait) \wedge \\ (ref' = (0.ref \cap 1.ref \cap X_2) \cup ((0.ref \cup 1.ref) \cap X_1) \\ \qquad \cup ((0.ref \cap 1.ref) - X_1 - X_2)) \\ (tr' - tr \in (0.tr - tr \parallel_{X_1} 1.tr - tr)) \end{pmatrix}$$

The merge predicate $M(X_1, X_2)$ encapsulates four types of behaviours in a parallel composition. Firstly, the composition diverges if either process diverges (expressed as $ok' = 0.ok \wedge 1.ok$). Secondly, the composition terminates if both

processes terminate ($wait' = 0.wait \vee 1.wait$). Thirdly, the composition handles synchronous channel outputs/inputs that are refused by both processes ($0.ref \cap 1.ref \cap X_2$), all actions in the set X_1 that are refused by both processes ($(0.ref \cup 1.ref) \cap X_1$), and events not in the set X_1 and X_2 but refused by both processes ($(0.ref \cap 1.ref) - X_1 - X_2$). Lastly, the trace of the composition belongs to the set of traces produced by the *trace synchronisation* function $\|_{X_1}$ as elaborated below.

Function $\|_{X_1}$ models how to merge two individual traces into a set of all possible traces; there are five cases covering both traces are empty, one of the trace is empty and both traces are non-empty. In the following definitions, s_1, s_1', s_2, s_2' are representative elements of variable states with termination, a, a_1, a_2 are representative elements of actions, ch is a representative element of channel names, τ_p is the event denoting the probabilistic choice selection, v, v_1, and v_2 are values with integer type, and p_1 and p_2 are probabilities with range of [0..1].

- Firstly, function $\|_{X_1}$ is symmetric, i.e., $t_1 \|_{X_1} t_2 = t_2 \|_{X_1} t_1$.
- The first case covers (**case-1**) two scenarios, (1) if both input traces are empty, the result is a set of an empty sequence; (2) if only one input trace is empty, the result is determined based on the first observation of that non-empty trace: if that observation is an action in the set X_1 which requires synchronisation, then the result is a set containing only an empty sequence, or otherwise, the first observation is recorded in the merged trace.
  case-1
  $1^\circ \quad \langle\rangle \|_{X_1} \langle\rangle = \{\langle\rangle\}$
  $2^\circ \quad \langle (s_1, h, p_1)\rangle \frown t \|_{X_1} \langle\rangle = \begin{cases} \{\langle\rangle\} & \text{if } h \in X_1 \\ \{\langle (s_1, h, p_1)\rangle \frown l \mid l \in t \|_{X_1} \langle\rangle\} & \text{otherwise} \end{cases}$
  where $h \in \{a, ch?v, ch!v, ch.v, \tau_p, s_1', \bot\}$
- The second case (**case-2**) covers a communication is over a synchronous channel. There are two aspects, (1) if the first observations of two input traces match (see Definition 1 below), then a synchronisation may occur (denoted by the set $\mathcal{G}_1$) or at this moment a synchronisation does not occur (denoted by the set $\mathcal{G}_2$); (2) Otherwise, either channel communication executes. Function $\mathsf{match_p}$ returns *true* if channel input and output from two processes respectively are enabled under the same pre-state with the same value.

  Definition 1 (Match). *Given two 3-tuples $tp_1 = (s_1, h_1, p_1)$ and $tp_2 = (s_2, h_2, p_2)$, where $h_1 \in \{ch?v_1, ch!v_1, ch.v_1\}$, $h_2 \in \{ch?v_2, ch!v_2, ch.v_2\}$, we say that they are matched if $s_1 = s_2$, $\{h_1, h_2\} = \{ch?v_1, ch!v_1\}$, and $v_1 = v_2$ are satisfied, denoted as $\mathsf{match_p}(tp_1, tp_2)$.*

  case-2 $\langle (s_1, h_1, p_1)\rangle \frown t_1 \|_{X_1} \langle (s_2, h_2, p_2)\rangle \frown t_2 =$
  $\begin{cases} \mathcal{G}_1 \cup \mathcal{G}_2 & \textit{if } \mathsf{match_p}((s_1, h_1, p_1), (s_2, h_2, p_2)) \\ \mathcal{G}_2 & \textit{otherwise} \end{cases}$
  where $h_1 \in \{ch?v_1, ch!v_1, ch.v_1\}$, $h_2 \in \{ch?v_2, ch!v_2, ch.v_2\}$, $\mathcal{G}_1 \mathrel{\widehat{=}} \{\langle (s_1, ch.v, p_1 \times p_2)\rangle \frown l \mid l \in t_1 \|_{X_1} t_2\}$, and $\mathcal{G}_2 \mathrel{\widehat{=}} \{\langle (s_1, h_1, p_1)\rangle \frown l \mid l \in t_1 \|_{X_1} \langle (s_2, h_2, p_2)\rangle \frown t_2\} \cup \{\langle (s_2, h_2, p_2)\rangle \frown l \mid l \in \langle (s_1, h_1, p_1)\rangle \frown t_1 \|_{X_1} t_2\}$.

– The third case (**case-3**) considers the synchronisation of two actions (a_1 and a_2). There are four scenarios with respect to the initial states (s_1 and s_2) and actions from the first observations of two traces: (1) if both actions are in the set X_1 but different or actions under different pre-states, then the result is a set containing only an empty sequence; (2) if actions from X_1 are the same and under the same pre-state, then a synchronisation occurs; (3) if one of the actions is not in X_1, the execution of the action is postponed to occur; and (4) if both actions are not in X_1, then either action can occur.

case-3 $\langle (s_1, a_1, p_1)\rangle \frown t_1 \parallel_{X_1} \langle (s_2, a_2, p_2)\rangle \frown t_2 =$

$$\begin{cases} \{\langle\rangle\} & a_1, a_2 \in X_1 \wedge \\ & (a_1 \neq a_2 \vee s_1 \neq s_2) \\ \{\langle (s_1, a_1, p_1 \times p_2)\rangle \frown l \mid l \in t_1 \parallel_{X_1} t_2\} & a_1, a_2 \in X_1 \\ & \wedge a_1 = a_2 \wedge s_1 = s_2 \\ \{\langle (s_2, a_2, p_2)\rangle \frown l \mid l \in \langle (s_1, a_1, p_1)\rangle \frown t_1 \parallel_{X_1} t_2\} & a_1 \in X_1 \wedge a_2 \notin X_1 \\ \{\langle (s_1, a_1, p_1)\rangle \frown l \mid l \in t_1 \parallel_{X_1} \langle (s_2, a_2, p_2)\rangle \frown t_2\} & \\ \cup & a_1 \notin X_1 \wedge a_2 \notin X_1 \\ \{\langle (s_2, a_2, p_2)\rangle \frown l \mid l \in \langle (s_1, a_1, p_1)\rangle \frown t_1 \parallel_{X_1} t_2\} & \end{cases}$$

– The fourth case (**case-4**) deals with the situation that the first observation of one input trace is in a waiting state (captured by $\perp$). The waiting state is due to the evaluation of the sequential program does not terminate. There are three scenarios for consideration: (1) if both observations are in waiting states, the result is a set of either observation, (2) if the other first observation is an action requiring the synchronisation ($h \in X_1$), the result contains the waiting observation only, (3) or otherwise, either observation from two processes occurs.

case-4

$1^\circ \langle (s_1, \perp, p_1)\rangle \parallel_{X_1} \langle (s_2, \perp, p_2)\rangle = \{\langle (s_1, \perp, p_1)\rangle, \langle (s_2, \perp, p_2)\rangle\}$

$2^\circ \langle (s_1, \perp, p_1)\rangle \parallel_{X_1} \langle (s_2, h, p_2)\rangle \frown t =$

$$\begin{cases} \{\langle (s_1, \perp, p_1)\rangle\} & \textit{if } h \in X_1 \\ \{\langle (s_1, \perp, p_1)\rangle\} \cup \{\langle (s_2, h, p_2)\rangle \frown l \mid l \in \langle (s_1, \perp, p_1)\rangle \parallel_{X_1} t\} & \textit{otherwise} \end{cases}$$

where $h \in \{a, ch?v, ch!v, ch.v, \tau_p, s_2'\}$

– The last case (**case-5**) defines the case when the first observation of one trace is an action a, a post-state s_1', or a τ_p event, and the other is a channel input $ch?v$, output $ch!v$, communication $ch.v$, a τ_p event, or a post-state s_2'. The merged observation depends on the action a in set X_1 or not, (1) if so, then its occurrence is postponed ($\mathcal{G}_3$), (2) or otherwise, either observation from two processes occurs ($\mathcal{G}_3 \cup \mathcal{G}_4$).

case-5 $\langle (s_1, h_1, p_1)\rangle \frown t_1 \parallel_{X_1} \langle (s_2, h_2, p_2)\rangle \frown t_2 = \begin{cases} \mathcal{G}_3 & \textit{if } h_1 \in X_1 \\ \mathcal{G}_3 \cup \mathcal{G}_4 & \textit{otherwise} \end{cases}$

where $h_1 \in \{a, \tau_p, s_1'\}$, $h_2 \in \{ch?v, ch!v, ch.v, \tau_p, s_2'\}$, $\mathcal{G}_3 \mathrel{\widehat{=}} \{\langle (s_2, h_2, p_2)\rangle \frown l \mid l \in \langle (s_1, h_1, p_1)\rangle \frown t_1 \parallel_{X_1} t_2\}$, and $\mathcal{G}_4 \mathrel{\widehat{=}} \{\langle (s_1, h_1, p_1)\rangle \frown l \mid l \in t_1 \parallel_{X_1} \langle (s_2, h_2, p_2)\rangle \frown t_2\}$.

Other processes and refinement The semantics of other processes is the same as those counterparts in the CSP# model [36] except state guard ($[b]P$). For process $[b]P$, Boolean expression b is evaluated simultaneously with the occurrence of the first event of process P. Like CSP#, in some situation, the process P behaves like *Skip*. Thus, no state could be observed to judge the truth of condition b. To address this, we construct process $\widehat{P}$ from P by adding a stuttering step.
$[b]P \mathrel{\widehat{=}} \widehat{P} \triangleleft (\mathcal{B}(b)(\pi_1(head(tr' - tr))) \wedge tr < tr') \triangleright Stop$
$\widehat{P} \mathrel{\widehat{=}} P \wedge tr < tr' \vee P(tr, tr) \wedge \exists\, s \in \mathsf{S} \cdot tr' - tr = \langle (s, s, 1) \rangle$

Refinement is a powerful technique for developing software systems that are correct by construction. In the UTP theory, it is expressed as an implementation satisfying a specification.

Definition 2 (Refinement). *Let P and Q be predicates for PCSP# processes with the same shared variable state space, the refinement $P \sqsupseteq Q$ holds iff $[P \Rightarrow Q]$.*

In the above definition, universal quantification implication means that for all observational variables, $\forall\, ok, ok', wait, wait', ref, ref', tr, tr' \cdot P \Rightarrow Q$ holds. In our current definition, the refinement ordering is strong; every observation that satisfies P must also satisfy Q. It requires that the associated probability in every observation of the trace shall be the same. The discussion on the probability comparison in the definition of refinement will be our future work.

5 Sports Analytics Using Probabilistic Model Checking

Sports analytics refers to the utilization of data science, artificial intelligence (AI), psychology, and Internet of Things (IoT) devices to enhance sports performance, strategic planning, and decision-making processes. It involves the collection, processing, and interpretation of data from various sources such as video recordings and scouting reports. This data aids in assessing both individual player and team performances, mitigating the risk of injuries, and assisting coaches in making well-informed decisions during both gameplay and training sessions. In this study, we employ Probabilistic Model Checking (PMC), a technique commonly employed in reliability analysis for intricate safety systems. We explain how this methodology can be adapted to sports strategy analytics, thereby augmenting the likelihood of achieving victory by considering the reliability of a player's specific sub-skill sets.

5.1 Overview

This section takes tennis as a paradigmatic example, our methodology has the potential for generalization across other racket sports like table tennis and badminton, as well as team-based sports such as soccer, basketball, and American football. In tennis, an individual's overall probability of winning is typically

contingent upon the reliability of their constituent skills, such as serving, returning, forehand, and backhand [6]. Furthermore, players may exhibit varied playing styles in response to different opponents. Our approach involves the modeling of tennis matches utilizing Markov Decision Processes (MDP) incorporating shot-by-shot actions, thus encapsulating intricate behaviors, including non-deterministic occurrences like coin tosses to determine the serving player and strategic choices such as body serves. The model leverages diverse match-related information, including player archetypes and potential actions, with probability distributions and success rates of these actions derived from historical data. Using the MDP model, we perform probabilistic reasoning for in-depth strategic analysis, such as deep strategy analytics for performance enhancement, facilitated by the Process Analysis Toolkit (PAT) model checker [43]. Although there is existing work on modeling tennis using MDP, our method, to the best of our knowledge, is the first to apply probabilistic model checking (PMC) for analytical purposes in this domain.

To rigorously evaluate the efficacy of our proposed strategies, we employ the MDP model to predict match outcomes and analyze alterations in winning probabilities ensuing from the implementation of different strategies. Our predictions are validated using data sourced from professional tennis matches (ATP and WTA) spanning the last decade. Furthermore, we execute experiments to directly compare our suggested strategies with actual strategic adaptations observed in past match data. The empirical findings demonstrate a strong correlation between the majority of our recommended strategies and the strategic adjustments applied by elite players, resulting in notable enhancements in their win rates.

5.2 The proposed approach

In this section, we introduce our methodology for tennis strategy analytics. The overview of our approach is given in Fig. 1. Initially, data is collected from online repositories, and video analytics methodologies are employed to enable automated data fusion. Subsequently, we model a tennis match as a Markov Decision Process (MDP), thereby enabling the simulation of matches involving any player pairing. The developed MDP model is instantiated in the PCSP# language and facilitates the prediction of match outcomes and the execution of strategy analytics via probabilistic model checking.

Data mining The dataset utilized in our investigation comprises comprehensive shot-by-shot descriptions for both players, crucial for constructing tennis models. To acquire this data, information is gathered from various sources.

- *Online data source.* Initially, our dataset is sourced from tennisabstract.com, an online repository that aggregates and annotates over 10,000 ATP and WTA matches dating back to 1959, encompassing comprehensive match data. Below illustrates an instance of a shot-by-shot record for a single point:

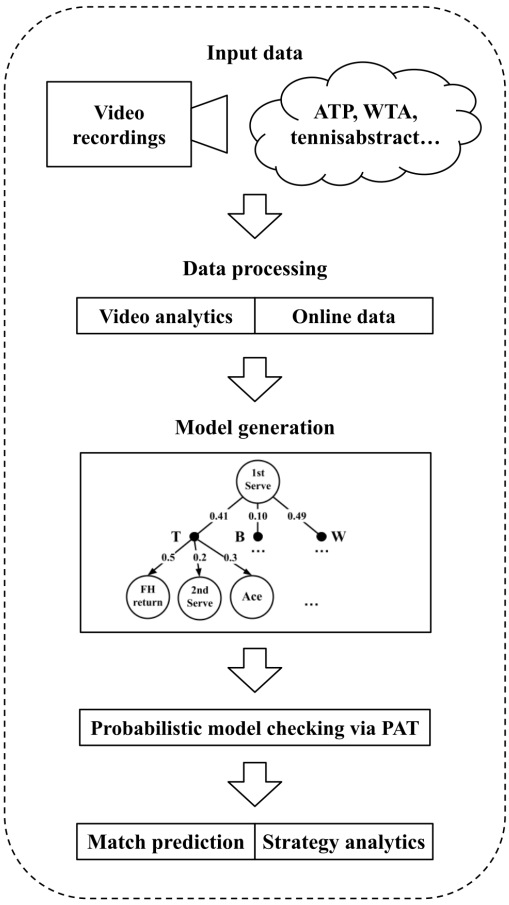

Fig. 1. The system pipeline of our approach.

```
1st serve wide; forehand return crosscourt;
backhand crosscourt; forehand down the middle;
forehand inside-out; forehand crosscourt;
backhand crosscourt; forehand crosscourt;
backhand down the line; backhand forced error.
```

We have developed a C# program to autonomously retrieve the matches of interest and extract data from the specified website. Furthermore, we have crafted a parser to extract related data, compute corresponding probabilities for individual sub-skills, and seamlessly integrate the outcomes into our pipeline as input parameters for the PCSP# model. Additionally, the program can automatically invoke the PAT console, facilitating the launch of the APT toolkit and the extraction of the desired results.

– *Video analytics.* In addition to data sourced from online repositories, there remains a dearth of detailed shot-by-shot information for many matches. To augment data coverage and cater to athletes at all levels, including college and junior players, we have devised a transformer-based seq2seq model capable of accurately recognizing intricate action sequences from fast-paced videos. Notably, this model can be trained end-to-end on a single GPU.
Moreover, recent advancements in Large Language Models (LLMs) have expanded their capabilities to perform multi-modal inference, enabling textual, visual, and auditory inputs. Specifically, GPT-4 exhibits proficiency in analyzing video content by processing image sequences. Initial evaluations on sports datasets showcase GPT-4's adeptness in various video comprehension tasks, such as sports genre classification, contextual understanding, and action identification, albeit with certain limitations in fine-grained classification and temporal analysis.

5.3 Modeling tennis in MDP

The task of modeling tennis matches is intricate, necessitating a delicate balance between precision and efficiency. An optimal model must accurately reflect the dynamic conditions of matches while also maintaining an appropriate level of abstraction to facilitate explainability and efficient analysis. Our modeling approach adeptly addresses these requirements by giving an expressive representation coupled with analytical capabilities.

In this section, our emphasis lies on singles tennis matches, wherein two players are denoted as $P1$ and $P2$. To predict the match outcome, we examine the winning probability within a *simplified* tiebreak game, modeled as a representation of the entire match. In a simplified tiebreak game, victory is attained by the player who first accumulates 7 points. In this example, 7-6 still counts as a win. Our analysis operates under the premise that the player with the highest likelihood of triumphing in the tiebreak game also possesses a superior probability of clinching the entire match. This abstraction improves computational performance when using PAT.

We model a tennis match using Markov Decision Processes (MDP), incorporating insights derived from expert knowledge in tennis. States and actions within the model are described using tennis-specific terminology to facilitate understanding by players and coaches. The model integrates diverse factors including court positioning (deuce court, middle court, or ad court), player characteristics (right-handed or left-handed), and various shot types. The ingredients of the MDP are outlined below.

State space. A state within the model corresponds to the instance when a player performs a shot. These states are grouped into four distinct categories: serve, return, stroke, and termination states. Serve states have four types: first serve in the deuce court, second serve in the deuce court (if the initial serve is unsuccessful), first serve in the ad court, and second serve in the ad court (if the first serve is unsuccessful). Return states denote the action of returning a serve and are classified into four types: forehand return of a serve from the

deuce court, backhand return of a serve from the deuce court, forehand return of a serve from the ad court, and backhand return of a serve from the ad court. Stroke states include actions executed within the deuce court, middle court, and ad court. Termination states signal the completion of the tiebreak game. Possible termination state outcomes include scores such as 7-1 (a win for $P1$), 5-7 (a win for $P2$), and 7-6 (a win for $P1$).

Action space. For each state s, a player has a range of potential actions depending on their positioning on the court and handedness. In total, there are 16 actions available, comprising:

- First/Second serve to T, B, W;
- Fore/Backhand (FH/BH): cross-court (CC), down the line (DL), down the middle (DM), inside-out (IO), inside-in (II).

Transition function. The transition function, denoted as $P : S \times A \times S \to [0, 1]$, quantifies the probability of transitioning to a new state given a state-action pair. Each state-action pairing can result in one of three possible outcomes: **in** (indicating that the action succeeds without immediately resulting in a point), **winner** (denoting that the action directly leads to winning a point), or **error** (signifying that the action fails). In the event of a **winner** or **error**, the process advances to a new state for the subsequent serve, with a point awarded to the rally winner. Upon either player accumulating 7 points, the process transitions to the termination state. Conversely, if the action outcome is **in**, the process progresses to a non-termination state depending on the current state and action undertaken.

Policy. The policy π denotes the probability distribution across all possible actions for each state, expressed as $\pi(a \mid s) = Pr(A = a \mid S = s)$. It characterizes how players select various actions within each state. For instance, a serving policy might allocate 60% probability to serving to T, 30% to W, and 10% to B.

The policy and transition probabilities for the MDP are derived from historical data. The likelihood of a player selecting action a from a given state s (i.e., the policy) is determined by

$$Pr(a \mid s) = \frac{N(s, a)}{N(s)}.$$

The transition probability, indicating the likelihood of transitioning to the subsequent state s' after executing action a from state s, is computed by

$$Pr(s' \mid s, a) = \frac{N(s, a, s')}{N(s, a)},$$

where $N(s)$ represents the frequency of a player's visits to state s, $N(s, a)$ is the number of times a player has executed action a in state s, and $N(s, a, s')$ is the number of times when a player has taken action a from state s, resulting in state s'.

Alternative models. Several alternative methods exist for modeling a tennis match. For example, we have developed models that feature more detailed

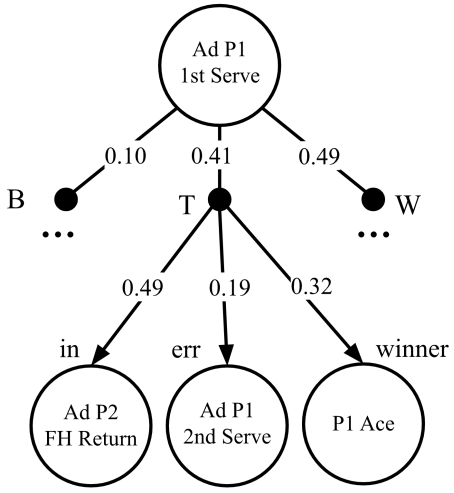

Fig. 2. A partial MDP model demonstrating a serve.

Table 1. Nadal's serve distributions against Federer.

	0-0	30-0	15-40	30-30	30-40	40-AD
T	0.849	0.700	0.571	0.643	0.353	0.238
B	0.038	0.033	0.143	0.119	0.412	0.477
W	0.113	0.267	0.286	0.238	0.235	0.285

court divisions by incorporating front and backcourt distinctions. Furthermore, we have devised models that consider the varying pressure levels associated with different score scenarios. By integrating scores into the state space, the probability distributions of actions are adjusted accordingly. An example of Rafael Nadal's serve distributions against Roger Federer at different scores is illustrated in Table 1.

However, in this study, we have chosen to present a simplified 6-region model without point-level analytics to facilitate player comprehension and streamline our data processing methods. It is crucial in actual tennis matches to avoid inundating players with excessive information that may lead to hesitation. Figure 2 presents a partial illustration of our MDP model, wherein $P1$ executes the first serve from the ad court with three possible actions (i.e., 41% serve T, 10% serve B, and 49% serve W). Each action entails three possible outcomes (i.e., in, error, winner) with corresponding transition probabilities leading to various subsequent states.

5.4 Implementation in PCSP#

We demonstrate the application of the developed MDP model in the PCSP# language through an example of a tiebreaker game played between two right-handed players. The tennis court is partitioned into 6 regions, as depicted in

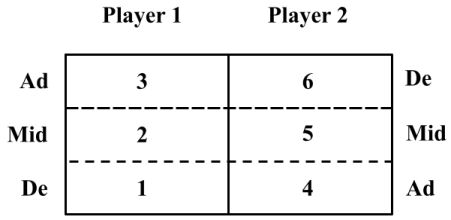

Fig. 3. A 6 region tennis court with ball position variables 1-6. Variable 0 indicates a winner, and 9 indicates an error.

Figure 3, where the labels *de*, *mid*, and *ad* denote the deuce court, middle court, and ad court, respectively.

Model for Serve Each player has the option to serve from either their deuce court or ad court, with a second serve opportunity available if the first attempt is unsuccessful. The ensuing example depicts a right-handed player executing their initial serve from the ad court, mirroring the process depicted in Figure 2.

$$
\begin{aligned}
&Ad\_P1Serve = pcase\ \{\\
&\quad 20 : ServeT\_in\{ball = 5\} \rightarrow Ad\_P2\_FHR\\
&\quad 8 : ServeT\_winner\{ball = 0\} \rightarrow Ace\{score1{+}{+}\,;\\
&\qquad\quad if\ (score1 == 7)\ \{won = P1\}\} \rightarrow NextPt\\
&\quad 13 : ServeT\_err\{ball = 9\} \rightarrow Ad\_P1Serve\_2nd\\
&\quad 25 : ServeW\_in\{ball = 4\} \rightarrow Ad\_P2\_BHR\\
&\quad 5 : ServeW\_winner\{ball = 0\} \rightarrow Ace\{score1{+}{+}\,;\\
&\qquad\quad if\ (score1 == 7)\ \{won = P1\}\} \rightarrow NextPt\\
&\quad 19 : ServeW\_err\{ball = 9\} \rightarrow Ad\_P1Serve\_2nd\\
&\quad 7 : ServeB\_in\{ball = 4\} \rightarrow (Ad\_P2\_FHR\ [\!]\ Ad\_P2\_BHR)\\
&\quad 0 : ServeB\_winner\{ball = 0\} \rightarrow Ace\{score1{+}{+}\,;\\
&\qquad\quad if\ (score1 == 7)\ \{won = P1\}\} \rightarrow NextPt\\
&\quad 3 : ServeB\_err\{ball = 9\} \rightarrow Ad\_P1Serve\_2nd\ \}\,;
\end{aligned}
$$

To associate actions with their corresponding probabilities during serving, the aforementioned process employs the PCSP# probabilistic choice operator "pcase". There exist three available actions: (1) 41% serve to the T ($ServeT$), (2) 49% serve wide ($ServeW$), and (3) 10% serve to the body ($ServeB$). Each action entails three potential outcomes: *in*, *winner*, or *err*. Taking the action $ServeT$ as an example, player $P1$ possesses a 20% probability of successfully hitting the ball in without directly scoring a point (modeled by event $ServeT\_in$, with the ball variable set to 5), with their opponent responding by returning a forehand shot from the ad court (modeled by process Ad_P2_FHR). There is a 13% chance

of failing the initial $ServeT$ (modeled by $ServeT\_err$, with the ball variable set to 9), necessitating a second serve, depicted by the process $Ad\_P1Serve\_2nd$. Additionally, there is an 8% probability of directly scoring a point (modeled by $ServeT\_winner$, with the ball variable set to 0), resulting in an increment to player $P1$'s score (score1++). If $P1$ attains 7 points, the won variable is updated to $P1$; otherwise, the process transitions to $NextPt$ for the subsequent serve or the conclusion of the game.

Model for Return A player has the option to return a serve from the deuce or ad court using either a forehand or backhand shot. For instance, the process Ad_P2_BHR describes how player $P2$ executes a backhand return against player $P1$'s wide serve from the ad court.

$$
\begin{aligned}
&Ad\_P2\_BHR = pcase\ \{\\
&\quad 24 : BH\_CrossCourt\_in\{ball = 3\} \rightarrow P1\_ad\_stroke\\
&\quad 21 : BH\_DownLine\_in\{ball = 1\} \rightarrow P1\_de\_stroke\\
&\quad 36 : BH\_DownMid\_in\{ball = 2\} \rightarrow P1\_mid\_stroke\\
&\quad 4 : Ad\_BHR\_winner\{ball = 0\} \rightarrow Winner\{score2++;\\
&\qquad\quad if\ (score2 == 7)\ \{won = P2\}\} \rightarrow NextPt\\
&\quad 15 : Ad\_BHR\_err\{ball = 9\} \rightarrow \{score1++;\\
&\qquad\quad if\ (score1 == 7)\ \{won = P1\}\} \rightarrow NextPt\,\};
\end{aligned}
$$

When player $P1$ serves from their ad court, player $P2$ may opt to return using a backhand cross-court ($BH\_CrossCourt$), backhand down the line ($BH\_DownLine$), or backhand down the middle ($BH\_DownMid$), each with different probabilities. Processes $P1\_de\_stroke$, $P1\_ad\_stroke$, and $P1\_mid\_stroke$ represent player $P1$'s stroke from the deuce, ad, and middle courts, respectively. To simplify the model, all potential winning shots (or errors) are amalgamated into a single event denoted as Ad_BHR_winner (or Ad_BHR_err). If player $P2$ returns with a winning shot, they score directly (i.e., score2++); conversely, if they commit an error, player $P1$ gains one point (i.e., score1++).

Model for Stroke Each player has a stroke process for each court position (deuce, ad, or middle court). The process for each court incorporates strokes executed with both forehand and backhand techniques, as well as scenarios involving winning shots and errors. The following exemplifies player $P1$'s stroke from the deuce court.

$$
\begin{aligned}
&P1\_de\_stroke = pcase\ \{\\
&\quad 40 : FH\_CrossCourt\_in\{ball = 6\} \rightarrow P2\_de\_stroke\\
&\quad 11 : FH\_DownLine\_in\{ball = 4\} \rightarrow P2\_ad\_stroke\\
&\quad 23 : FH\_DownMid\_in\{ball = 5\} \rightarrow P2\_mid\_stroke
\end{aligned}
$$

$$
\begin{aligned}
&0 : BH\_InsideOut\_in\{ball = 6\} \rightarrow P2\_de\_stroke \\
&2 : BH\_InsideIn\_in\{ball = 4\} \rightarrow P2\_ad\_stroke \\
&1 : BH\_DownMid\_in\{ball = 5\} \rightarrow P2\_mid\_stroke \\
&9 : de\_stroke\_winner\{ball = 0\} \rightarrow Winner\{score1++; \\
&\qquad if\ (score1 == 7)\ \{won = P1\}\} \rightarrow NextPt \\
&14 : de\_stroke\_err\{ball = 9\} \rightarrow \{score2++; \\
&\qquad if\ (score2 == 7)\ \{won = P2\}\} \rightarrow NextPt\,\};
\end{aligned}
$$

Predict winning probability To predict the probability of a player winning, we establish a predicate designating the target player as the victor of the tiebreaker game (i.e., the first to accumulate 7 points), and subsequently find a state that meets the criteria using probabilistic reachability analysis. For instance, the below assertion describes the property of player $P1$ emerging victorious in the game.

$$
\begin{aligned}
&\#\ define\ \ player1Win\ \ won\ ==\ player1; \\
&\#\ assert\ \ TieBreakGame\ \ reaches\ \ player1Win\ \ with\ \ prob;
\end{aligned}
$$

5.5 Strategy recommendations

Through PAT's capability to check the ramifications of alterations to the probability distribution of actions, we can propose strategies aimed at enhancing a player's likelihood of winning, drawing on sensitivity analysis derived from probabilistic model checking. Two primary categories of strategies can be discerned, as outlined in Algorithm 1.

Pre-match strategy. The first category of strategy pertains to intricate tactics concerning play patterns. This category is inherently linked to the probability distributions of actions. For instance, one might opt to shift 10% of T serves to W serves against a specific opponent if historical data indicates that the opponent struggles with returning W serves. Since coaching is prohibited during a tennis match, we denote this first type of strategy as "pre-match strategy", as it can be directly implemented by a player prior to the commencement of the match without altering the ability (i.e., reliability/success rate) of their sub-skills. The details of this strategy are outlined in Algorithm 1, wherein optimal actions to increase are identified for each state (e.g., player $P1$ should utilize more forehand cross-court shots in the deuce court against player $P2$).

Training strategy. The second category of strategy aims to enhance the success rates of specific shot types through targeted training. This strategy is termed "training strategy". It may not be immediately applicable just before a match, as a player cannot instantaneously improve a skill. For instance, prior to facing Nadal, renowned for his potent forehand, Federer might concentrate on refining his backhand down-the-line shots in the ad court to reduce errors by 2%. Such a strategy not only elevates Federer's success rate with his backhand but also diminishes the threat posed by Nadal's forehand, as the down-the-line shot directs play to Nadal's backhand side.

Algorithm 1: Computing the optimal strategy.

Data: current state s, policy π, percentage change $\delta\%$, maximum increase in winning chance Δ_{max}
Result: a_{best} (the best action to increase in state s)

$\Delta_{max} \leftarrow -\infty$; $a_{best} \leftarrow None$;
for $a \in \pi(s)$ **do**
  if *pre-match strategy* **then**
    `// pre-match strategy`
    `// add action a's percentage by` $\delta\%$
    $Pr(a \mid s) \leftarrow Pr(a \mid s) + \delta\%$;
    for $a' \in \pi(s) \setminus [a]$ **do**
      $Pr(a' \mid s) \leftarrow Pr(a' \mid s) - \frac{\delta\%}{|\pi(s)|}$;
  else
    `// training strategy`
    `// reduce action a's error by` $\delta\%$
    $Pr(in \mid s, a) \leftarrow Pr(in \mid s, a) + \delta\%$;
    $Pr(err \mid s, a) \leftarrow Pr(err \mid s, a) - \delta\%$;
  `// compute the increase in winning chance`
  $\Delta p_{win} \leftarrow$ increase in winning chance;
  `// update the best action to increase`
  if $\Delta p_{win} > \Delta_{max}$ **then**
    $\Delta_{max} = \Delta p_{win}$; $a_{best} = a$;

5.6 Evaluation

We employed the PAT model checker to realize our model and conducted experiments on real-world professional tennis matches to evaluate the effectiveness of our strategy analytics methodology. Specifically, our experiments were devised to address the following research questions:

RQ1: How accurate is the model in predicting players' winning chances when playing against different opponents?
RQ2: Does our model provide effective pre-match and training strategies to increase players' winning chances?
RQ3: What new insights a player/coach can get from our strategy recommendations?

RQ1: Winning chance prediction accuracy To predict one's probability of winning against a specific opponent, our framework offers the flexibility to extract data from pertinent matches occurring prior to the date of the target match. For example, let us consider predicting a match between players $P1$ and $P2$, each with Elo rankings $e1$ and $e2$, respectively. To acquire data for $P1$, we gather information from matches involving $P1$ and opponents akin to $P2$ during the preceding two years. Here, "akin" denotes opponents sharing (1) the same handedness as $P2$, and (2) Elo rankings falling within the range of $[e2 - \delta_{elo}, e2 + \delta_{elo}]$, where $\delta_{elo} \in \mathbb{N}$. A similar approach is adopted to gather

data for player $P2$. Once the relevant matches have been selected from historical records, we can construct the MDP model and predict the outcome of the match. Typically, the processing time for each match is approximately 1 second.

We utilize the bookmakers' odds[4] as our reference point, as it currently represents the state-of-the-art approach in the domain. Previous investigations have not consistently outperformed the predictions made by bookmakers over an extended period. This is primarily attributed to the valuable information typically possessed by bookmakers, including weather conditions, recent player performances, injury reports, and insider insights, which are often challenging to access. Nevertheless, we demonstrate that our predictions yield positive returns when compared to the bookmakers' odds through betting simulations spanning the past decade. Furthermore, we conduct a comparative analysis of our methodology against existing tennis prediction models documented in the literature.

Betting simulation. In our experiment, we apply a well-established betting strategy — Kelly criterion [45], which is given as

$$f = k \times (p - \frac{1-p}{b}) \tag{1}$$

where p represents the predicted probability of winning the bet, b signifies the payout resulting from a successful bet, k denotes the Kelly multiplier for risk management, and f denotes the fraction of the current bankroll to be wagered (with betting being avoided when $f < 0$). To assess profitability, we compute the return on investment (ROI) and the annualized return on investment (annualized ROI) [29]. The initial bankroll is initialised at \$10,000.

We investigate different constraints regarding the range of Elo ranking differences, denoted as $\delta_{elo} \in [50, 200]$, during the selection of related historical matches. When δ_{elo} assumes a smaller value, the chosen matches exhibit higher quality, albeit potentially limited in quantity. Conversely, with an increase in δ_{elo}, a larger pool of matches is included, albeit they may possess less direct relevance to the target matches.

We designate the Kelly multiplier as $k = 0.1$ for the purpose of long-term betting strategies. Additionally, our betting approach concentrates exclusively on matches that entail a minimum of 4 relevant historical matchups for each player, ensuring prediction accuracy. The outcomes of the betting simulations are depicted in Table 2. Examination of the table reveals that models with $\delta_{elo} = 50$ and 100 yield long-term profits, with annualized ROIs of 7.49% and 14.98%, respectively. However, a further elevation in δ_{elo} leads to negative profitability. This occurs because, although a larger number of matches become available for betting, the selected historical matchups exhibit decreased relevance to the target players. Consequently, a value of $\delta_{elo} = 100$ emerges as the optimal point for performance. This experiment underscores the exceptional predictive performance of our model in determining the winning probability of tennis matches.

[4] `http://www.tennis-data.co.uk/`

Table 2. Betting results over the past 10 years.

δ_{elo} (±)	# of bets	Profits	ROI	Annualized ROI
50	461	$10,592	105.92%	7.49%
100	1,388	$30,385	**303.85%**	**14.98%**
150	2,177	-$8,194	-81.94%	-15.73%
200	2,871	-$8,471	-84.71%	-17.12%

Table 3. ECE scores for different methods.

Method	ECE
Point-based [16]	0.0973
Paired comparison [25]	0.0317
Bookmakers	0.0207
Our method	**0.0099**

Comparison with existing methods. A reliable winning probability prediction model should support well-calibrated estimations that closely align with real-world outcomes. Notably, predictions indicating a 70%-30% chance or 95%-5% chance of winning significantly impact performance analytics, despite both scenarios predicting the same winner. Traditional metrics such as accuracy and log-loss do not adequately capture the true winning probabilities. To address this limitation, we assess the models' predictions using the expected calibration error (ECE) [10]. The ECE quantifies the disparity between predicted probabilities and observed outcomes. For instance, if player $P1$ is predicted to have a winning probability of 40% against player $P2$, we gather all matches where the predicted winning chance hovers around 40% and ascertain whether approximately 40% of those matches indeed resulted in victory for player $P1$.

Formally, we partition the predicted winning probabilities into M bins and calculate the average discrepancy between the predicted and observed outcomes. This calculation is weighted by the number of examples in each bin. The ECE is computed as follows:

$$ECE = \sum_{m=1}^{M} \frac{\mid B_m \mid}{N} \mid acc(B_m) - conf(B_m) \mid \tag{2}$$

where B_m is the m^{th} bin, $\mid B_m \mid$ is the size of the bin, N is the number of samples, $acc(B_m)$ is the proportion of positives in bin B_m, and $conf(B_m)$ is the average predicted probability in bin B_m. In our experiment, we use $M = 5$.

We evaluate the performance of our model with $\delta_{elo} = 100$ against other established match outcome prediction models, which include a point-based method [16], a paired comparison method [25], and bookmakers. As illustrated in Table 3, our model exhibits the best performance with the lowest ECE of 0.0099.

Our model demonstrates accurate prediction of players' winning probabilities, validated through betting simulations spanning the last decade, achieving an annualized ROI of 14.98%. Furthermore, our model surpasses existing prediction models, exhibiting the lowest ECE.

Table 4. The system recommends actions for enhancement compared to the actual strategy adjustments made by players. "Win%" indicates the likelihood of winning. "Align_P" represents the proportion of our pre-match suggestions that align with players' actual strategy adjustments. "Align_T" indicates the fraction of our training recommendations that align with players' actual sub-skill enhancements.

Player	Opponent	Turning-point year	Win% before year	Win% after year	Align_P	Align_T
Federer R.	Nadal R.	2017	20.0%	83.3%	9/11	8/11
Nadal R.	Djokovic N.	2017	26.1%	60.0%	7/11	8/11
Wawrinka S.	Djokovic N.	2016	28.6%	50.0%	8/11	7/11
Murray A.	Nadal R.	2015	14.3%	50.0%	10/11	8/11
Thiem D.	Nadal R.	2018	16.7%	57.1%	6/11	6/11
Medvedev D.	Zverev A.	2020	25.0%	83.3%	6/11	7/11
Zverev A.	Tsitsipas S.	2021	20.0%	50.0%	8/11	7/11
Thiem D.	Djokovic N.	2018	25.0%	60.0%	9/11	6/11
Djokovic N.	Tsitsipas S.	2020	50.0%	100.0%	9/11	8/11
Zverev A.	Nadal R.	2020	20.0%	66.7%	6/11	8/11
Federer R.	Murray A.	2014	40.0%	100.0%	6/11	9/11
Djokovic N.	Medvedev D.	2021	50.0%	100.0%	9/11	7/11
Zverev A.	Federer R.	2018	33.3%	66.7%	8/11	9/11
Thiem D.	Federer R.	2019	50.0%	100.0%	7/11	10/11
Djokovic N.	Murray A.	2014	58.3%	82.4%	7/11	7/11
Cilic M.	Djokovic N.	2016	0.0%	33.3%	8/11	8/11
Rublev A.	Medvedev D.	2021	0.0%	50.0%	7/11	10/11

RQ2: Tennis strategy effectiveness Our system empowers users to assess the impact of different pre-match or training strategies on their performance, facilitating the identification of optimal approaches for improvement. However, assessing the real-world efficacy of these strategies poses challenges, as asking professional players to modify their playing styles or skill levels for observation is impractical. Therefore, we evaluate the effectiveness of these strategies based on insights gleaned from historical data.

For example, consider the dataset comprising 16 documented matches between Roger Federer and Rafael Nadal from 2011 to 2022. Prior to 2017, Federer won only 2 out of 10 matches, yielding a 20% winning rate. However, post-2017, his winning rate surged to 83% after securing victories in 5 out of 6 matches. Notably, upon analyzing Federer's actions and the reliability of his sub-skills, noticeable differences are evident before and after 2017. To ascertain whether this improvement stems from strategic alterations, we constructed two Markov Decision Process (MDP) models based on historical data—one before 2017 and the other after. Employing Probabilistic Model Checking, we computed Federer's winning rate, yielding results of 35.7% and 53.2%, respectively, aligning with the actual outcomes.

Moreover, we aim to investigate whether our system can aid players in pinpointing pre-match and training strategies to attain optimized enhancements in the future. Our objective is to evaluate the following:

– For pre-match strategy, identify the best action to increase in each state by modifying **probability distributions**.

- For training strategy, identify the best action to improve/train on in each state by modifying **success rates**.
- Check whether the player has indeed increased/improved on the identified actions later on.

For instance, utilizing data predating 2017, our system suggests that Federer's optimal pre-match maneuver against Nadal involves employing the "forehand cross-court" strategy when returning the ball at the deuce court. Post-2017 data corroborates that Federer did indeed execute more forehand cross-court returns. Concerning training strategies, leveraging data from before 2017, our approach suggests that Federer's most advantageous improvement opportunity in the "ad court stroke" scenario entails employing the "backhand down the line" tactic toward Nadal's backhand side. Subsequent to 2017, data indicates that Federer did, in fact, enhance this sub-skill, achieving an increased success rate.

We implemented the aforementioned approach across all 11 states for both pre-match and training strategies. The outcomes revealed that: (1) concerning pre-match strategies, 9 out of 11 optimal actions identified by our method corresponded with Federer's actual adjustments; and (2) for training strategies, 8 out of 11 recommended enhancement actions were in concordance with Federer's actual improvements, specifically in terms of elevating action success rates.

To further corroborate our methodology, we curated additional instances akin to Federer's matchups against Nadal and subjected them to the same validation procedure. These instances adhere to a pattern wherein a player exhibits a notable surge in win rate against a specific opponent subsequent to a certain temporal threshold (e.g., the year 2017). We leveraged data predating this threshold to formulate recommendations and utilized post-threshold data to verify our suggestions. Table 4 synthesizes the empirical findings, revealing that the majority of recommendations delineated by our system align with players' actual strategy adjustments and enhancements. Hence, we conlude that our strategy analytics approach is both rational and efficacious.

The strategies posited by our model demonstrate effectiveness as they closely correspond with real-world strategy adaptations made by elite players, resulting in substantial enhancements when facing specific opponents, as evidenced by historical data.

RQ3: New insights for players and coaches It may be contended that our recommendations significantly coincide with the practices already employed by professional athletes. Nonetheless, these invaluable insights may not be readily available to players lacking equivalent levels of coaching staff and resources. Our objective is to democratize high-caliber strategy analysis for the wider tennis community. In this section, we will delve further into the insights engendered by our system and illustrate how players and coaches can derive benefits from them.

Our system is capable of generating opponent-specific strategy recommendations tailored to various player matchups. Table 5 showcases examples of our pre-match and training strategy suggestions. It's important to emphasize that

Table 5. Different pre-match and training strategies at each state.

Player	Opponent	De_Serve	Ad_Serve	De_FHR	Ad_FHR	De_BHR	Ad_BHR	De_Stroke	Mid_Stroke	Ad_Stroke
Pre-match Strategy										
Nadal R.	Djokovic N.	W	W	CC	DL	CC	IO	FH_IO	FH_IO	BH_IO
Wawrinka S.	Djokovic N.	W	T	CC	II	IO	DL	BH_II	BH_IO	FH_IO
Thiem D.	Djokovic N.	T	W	CC	CC	IO	CC	FH_CC	BH_IO	FH_II
Cilic M.	Djokovic N.	T	B	DL	CC	DM	DL	FH_CC	FH_IO	BH_DL
Zverev A.	Tsitsipas S.	B	T	CC	CC	CC	CC	FH_DL	FH_CC	FH_II
Zverev A.	Nadal R.	T	W	DL	IO	CC	CC	FH_CC	FH_IO	FH_DM
Zverev A.	Federer R.	T	T	CC	DM	II	DM	FH_DL	FH_CC	BH_DL
Training Strategy										
Thiem D.	Nadal R.	T	W	CC	II	IO	DL	FH_CC	FH_CC	FH_II
Thiem D.	Djokovic N.	T	W	CC	CC	IO	DL	FH_CC	FH_CC	BH_DL
Thiem D.	Federer R.	W	T	DL	II	DM	CC	FH_DL	FH_IO	BH_CC

we present only the most effective strategy in some states, which corresponds to the action yielding the highest gains when increasing the probability distributions or success rates by 2%. Our strategy recommendations demonstrate significant diversity across different matchups. For example, we propose distinct pre-match strategies for Zverev against different opponents, such as employing more forehand cross-court (FH_CC) against Nadal and forehand down the line (FH_DL) against Federer in the De_Stroke state. Moreover, when different players face the same opponent, our system suggests Wawrinka to increase his forehand inside-out (FH_IO) and Cilic to enhance his backhand down the line (BH_DL) at Ad_Stroke when playing against Djokovic.

At times, our method can propose "unusual" or novel strategy suggestions that players or coaches may not have considered. For example, through analyzing matches between Roger Federer and Andy Murray before 2014, our system indicates that Federer's most effective pre-match strategy was to employ more backhand down the line shots towards Murray's forehand, contrary to the common belief that attacking the opponent's backhand side is more effective. Interestingly, after 2014, Federer did not lose any match against Murray and indeed increased his usage of backhand down the line shots by 4.8%. Given our model's ability to accurately predict match outcomes, we can provide formal verification that our suggested strategies have the potential to enhance players' winning chances in theory.

Moreover, even if players or coaches are aware of the specific sub-skills requiring refinement, they may lack precise guidance on the adjustments or improvements needed to maximize their winning probabilities effectively. Our system is able to compute exact figures for these enhancements. One potential application is to aid players in tailoring their training regimes more efficiently. For instance, if two players are both instructed to enhance their forehand down the line, but the anticipated impacts on their winning probabilities vary (for instance, by reducing the error rate of forehand down the line by 2%, one player's winning probability may increase by 4%, while the other's may increase by 8%), the second player should allocate a larger proportion of their training regimen to this skill, assuming a fixed training duration.

Overall, our model offers insightful strategy recommendations that surpass conventional knowledge. These strategies demonstrate significant diversity across various matchups and can be rigorously validated. Furthermore, we can provide players with tailored training regimens to enhance their performance efficiently and effectively.

6 Related Work

TCOZ [21–23], proposed by Mahony and Dong, is a formal language that integrates two formal specification languages, Object-Z [37] and Timed CSP [5], and is inspired by Woodcock's research on formal mathematics for systems specification and design [48]. TCOZ combines the strengths of Object-Z in complex data and algorithm modeling with the advantages of Timed CSP in real-time concurrent modeling. Inspired by TCOZ, CSP# is proposed by replacing Object-Z with C# to leverage the benefits of C# as a widely-used programming language, which offers better support for model checking and practical implementation. Later, Dong and his team developed the model checker Process Analysis Toolkit (PAT) [40,43], which adopts CSP# as the modeling langauge. PAT has then been extend with numerous modules that encompass different variants of CSP#, including Stateful Timed CSP [26,42], probabilistic CSP [38], probabilistic real-time module [39], game theory module [7], and nested and hierarchical model checking [2]. Such variety of modeling and verification techniques support applications in many different domains, including PDDL modeling [19], program verification [53], cybersecurity [18], smart homes [17], and so on.

Unifying Theories of Programming (UTP) [15] proposed by Hoare and He in the late 1990s aims to unify different "theories of programming" by providing a common mathematical language for expressing semantics, which formalises the theories of different programming paradigms, such as sequential programming, object-oriented programming [12,20], concurrent programming through reactive processes and communications, higher-order logic programming, and so on. The UTP framework is built on the principles of algebraic theories and provides a mathematical basis for reasoning about the semantics of programming languages. Its algebraic foundations and support for various programming paradigms make it a valuable framework for formal specification and verification. The semantics of programming languages based on the UTP theories include CSP [3, 15], *Circus* [27,28], *Circus Time* [34,46], TCOZ [30], CSP# [36], Modelica [8], rCOS [12], Web service [11], and MDESL [33].

Miyazawa et al. [24] proposes a formalisation of RoboChart which is a state machine-based notation designed for modelling real-time current robot systems. The formalisation uses CSP [3] and its dialect tock-CSP [32] based on the UTP framework. Conserva Filho [4] defines the semantics of the probabilistic operator in RoboChart using the CSP probabilistic operator $\boxplus$ [9]. Woodcook et al. [49] proposes a probabilistic denotational semantics to the RoboChart language using the powerful approach of weakest completion semantics, addressing challenges related to demonic and probabilistic choice. The methodology ensures

consistency and facilitates the transition of programming concepts between the standard and probabilistic languages. In [51], a new construct for probabilistic choice in introduced in RoboChart's state machines for probabilistic modelling. The RoboChart model is transformed into a PRISM model for verification through model checking.

He et al. [13] discusses two models for an extension of Dijkstra's language of guarded commands, focusing on the introduction of a probabilistic choice operator to express randomized algorithms. Bresciani et al. [1] introduces a probability modeling framework in the style of UTP, offering a solution that allows for better consideration and handling of probabilistic properties in the modeling and verification of systems. While the paper extends UTP to handle probabilities, there might still be limitations in terms of the expressiveness of the formalism. Certain probabilistic concepts or advanced modeling scenarios may not be fully captured within the proposed framework.

Zhu et al. [52] provides a foundational denotational semantics model for a probabilistic timed shared-variable language. By presenting a rigorous denotational semantics model, the paper lays a solid theoretical foundation for understanding the intricacies of system behavior encoded in such languages. However, it may not explicitly address event-driven characteristics commonly found in probabilistic CSP# models.

7 Conclusion

PCSP# enables the specification of probabilistic behaviors and uncertainties in system designs, expanding the expressiveness of CSP to model a broader range of real-world systems. Its features, including probabilistic transitions, uncertain actions, and random events, have found applications in various domains such as probabilistic systems modeling, reliability analysis, fault tolerance analysis, security analysis, and performance evaluation. This paper has demonstrated a novel application of PCSP# in sports analytics through the PAT model checker, specifically in modeling tennis matches. By leveraging past match data, PCSP# facilitates detailed analysis of match outcomes, strategy assessment, and training recommendations. Moving forward, the continued development and adoption of PCSP# and similar formal modeling techniques hold great potential for advancing sports analytics and enhancing the understanding and optimization of complex systems in various domains.

References

1. Bresciani, R., Butterfield, A.: Towards a UTP-style framework to deal with probabilities. Technical Report TCD-CS-2011-09, FMG, Trinity College Dublin, Ireland (August 2011)
2. Bride, H., Cai, C., Dong, J.S., Goré, R., Hóu, Z., Mahony, B.P., McCarthy, J.: N-PAT: A nested model-checker - (system description). In: 10th International Joint Conference on Automated Reasoning, IJCAR 2020. Lecture Notes in Computer Science, vol. 12167, pp. 369–377. Springer (2020)

3. Cavalcanti, A., Woodcock, J.: A Tutorial Introduction to CSP in *Unifying Theories of Programming*. In: Refinement Techniques in Software Engineering. Lecture Notes in Computer Science, vol. 3167, pp. 220–268. Springer (2006)
4. Conserva Filho, M.S., Marinho, R., Mota, A., Woodcock, J.: Analysing RoboChart with Probabilities. In: Formal Methods: Foundations and Applications. pp. 198–214. Springer International Publishing (2018)
5. Davies, J.: Specification and proof in real-time CSP. Cambridge University Press (1993)
6. Dong, J.S., Shi, L., Jiang, K., Sun, J., et al.: Sports strategy analytics using probabilistic reasoning. In: 20th International Conference on Engineering of Complex Computer Systems, ICECCS 2015. pp. 182–185. IEEE (2015)
7. Fernando, D., Dong, N., Jégourel, C., Dong, J.S.: Verification of nash-equilibrium for probabilistic BAR systems. In: 21st International Conference on Engineering of Complex Computer Systems, ICECCS 2016. pp. 53–62. IEEE Computer Society (2016)
8. Foster, S., Thiele, B., Cavalcanti, A., Woodcock, J.: Towards a UTP semantics for modelica. In: Unifying Theories of Programming, UTP 2016. Lecture Notes in Computer Science, vol. 10134, pp. 44–64. Springer (2016)
9. Goldsmith, M.: CSP: The Best Concurrent-System Description Language in the World - Probably! In: Communicating Process Architectures 2004 (2004)
10. Guo, C., Pleiss, G., Sun, Y., Weinberger, K.Q.: On calibration of modern neural networks. In: International conference on machine learning. pp. 1321–1330. PMLR (2017)
11. He, J.: UTP semantics for web services. In: Integrated Formal Methods, IFM 2007. Lecture Notes in Computer Science, vol. 4591, pp. 353–372. Springer (2007)
12. He, J., Li, X., Liu, Z.: rCOS: A refinement calculus of object systems. Theor. Comput. Sci. **365**(1-2), 109–142 (2006)
13. He, J., Seidel, K., McIver, A.: Probabilistic models for the guarded command language. Science of Computer Programming **28**, 171–192 (1999)
14. Hoare, C.: Communicating Sequential Processes. Prentice-Hall (1985)
15. Hoare, C., He, J.: Unifying Theories of Programming. Prentice-Hall (1998)
16. Ingram, M.: A point-based bayesian hierarchical model to predict the outcome of tennis matches. Journal of Quantitative Analysis in Sports **15**(4), 313–325 (2019)
17. Lee, V.Y., Liu, Y., Zhang, X., Phua, C., Sim, K., Zhu, J., Biswas, J., Dong, J.S., Mokhtari, M.: ACARP: auto correct activity recognition rules using process analysis toolkit (PAT). In: 10th International Conference on Smart Homes and Health Telematics, ICOST 2012. Lecture Notes in Computer Science, vol. 7251, pp. 182–189. Springer (2012)
18. Li, L., Sun, J., Liu, Y., Dong, J.S.: Tauth: Verifying timed security protocols. In: 16th International Conference on Formal Engineering Methods, ICFEM 2014. Lecture Notes in Computer Science, vol. 8829, pp. 300–315. Springer (2014)
19. Li, Y., Sun, J., Dong, J.S., Liu, Y., Sun, J.: Translating PDDL into CSP# - the PAT approach. In: 17th IEEE International Conference on Engineering of Complex Computer Systems, ICECCS 2012. pp. 240–249. IEEE Computer Society (2012)
20. Liu, Z.: Linking formal methods in software development - A reflection on the development of rcos. In: Theories of Programming and Formal Methods - Essays Dedicated to Jifeng He on the Occasion of His 80th Birthday. Lecture Notes in Computer Science, vol. 14080, pp. 52–84. Springer (2023)
21. Mahony, B., Dong, J.S.: Blending Object-Z and Timed CSP: an introduction to TCOZ. In: Proceedings of the 20th International Conference on Software Engineering. pp. 95–104 (1998)

22. Mahony, B., Dong, J.S.: Timed Communicating Object Z. IEEE Transactions on Software Engineering **26**(2), 150–177 (2000)
23. Mahony, B.P., Dong, J.S.: Overview of the Semantics of TCOZ. In: Proceedings of the 1st International Conference on Integrated Formal Methods. p. 66–85. IFM '99, Springer-Verlag (1999)
24. Miyazawa, A., Ribeiro, P., Li, W., Cavalcanti, A., Timmis, J., Woodcock, J.: Robochart: modelling and verification of the functional behaviour of robotic applications. Software & Systems Modeling **18**, 3097–3149 (2019)
25. Morris, B., Bialik, C., Boice, J.: How we're forecasting the 2016 us open. Also available at https://fivethirtyeight. com/features/how-were-forecasting-the-2016-us-open/. Search in (2016)
26. Nguyen, T.K., Sun, J., Liu, Y., Dong, J.S.: Symbolic model-checking of stateful timed CSP using BDD and digitization. In: 14th International Conference on Formal Engineering Methods, ICFEM 2012. Lecture Notes in Computer Science, vol. 7635, pp. 398–413. Springer (2012)
27. Oliveira, M., Cavalcanti, A., Woodcock, J.: A Denotational Semantics for Circus. Electronic Notes in Theoretical Computer Science **187**, 107–123 (2007)
28. Oliveira, M., Cavalcanti, A., Woodcock, J.: A UTP Semantics for *Circus*. Formal Aspects of Computing **21**(1-2), 3–32 (2009)
29. Phillips, J.J.: Measuring return on investment, vol. 2. American Society for Training and Development (1994)
30. Qin, S., Dong, J.S., Chin, W.N.: A Semantic Foundation for TCOZ in Unifying Theories of Programming. In: FME 2003: Formal Methods. Lecture Notes in Computer Science, vol. 2805, pp. 321–340. Springer (2003)
31. Roscoe, A.W.: The Theory and Practice of Concurrency. Prentice Hall (1997)
32. Roscoe, A.: Understanding Concurrent Systems. Springer-Verlag, Berlin, Heidelberg, 1st edn. (2010)
33. Sheng, F., Zhu, H., He, J., Yang, Z., Bowen, J.P.: Theoretical and practical approaches to the denotational semantics for MDESL based on UTP. Formal Aspects Comput. **32**(2-3), 275–314 (2020)
34. Sherif, A., Cavalcanti, A., Jifeng, H., Sampaio, A.: A process algebraic framework for specification and validation of real-time systems. Form. Asp. Comput. **22**(2), 153–191 (mar 2010)
35. Shi, L., Zhao, Y., Liu, Y., Sun, J., Dong, J.S., Qin, S.: A UTP Semantics for Communicating Processes with Shared Variables. In: 15th International Conference on Formal Engineering Methods (ICFEM'13). Lecture Notes in Computer Science, vol. 8144, pp. 215–230. Springer (2013)
36. Shi, L., Zhao, Y., Liu, Y., Sun, J., Dong, J.S., Qin, S.: A UTP semantics for communicating processes with shared variables and its formal encoding in PVS. Form. Asp. Comput. **30**(3–4), 351–380 (2018)
37. Smith, G.: The Object-Z specification language. Kluwer Academic Publishers (2000)
38. Song, S., Hao, J., Liu, Y., Sun, J., Leung, H., Dong, J.S.: Analyzing multi-agent systems with probabilistic model checking approach. In: 34th International Conference on Software Engineering, ICSE 2012. pp. 1337–1340. IEEE Computer Society (2012)
39. Song, S., Sun, J., Liu, Y., Dong, J.S.: A model checker for hierarchical probabilistic real-time systems. In: 24th International Conference on Computer Aided Verification, CAV 2012. Lecture Notes in Computer Science, vol. 7358, pp. 705–711. Springer (2012)

40. Sun, J., Liu, Y., Dong, J.S.: Model Checking CSP Revisited: Introducing a Process Analysis Toolkit. In: Leveraging Applications of Formal Methods, Verification and Validation. pp. 307–322. Springer Berlin Heidelberg (2008)
41. Sun, J., Liu, Y., Dong, J.S., Chen, C.: Integrating Specification and Programs for System Modeling and Verification. In: The 3rd IEEE International Symposium on Theoretical Aspects of Software Engineering (TASE'09). pp. 127–135. IEEE Computer Society (2009)
42. Sun, J., Liu, Y., Dong, J.S., Liu, Y., Shi, L., André, E.: Modeling and verifying hierarchical real-time systems using stateful timed CSP. ACM Trans. Softw. Eng. Methodol. **22**(1), 3:1–3:29 (2013)
43. Sun, J., Liu, Y., Dong, J.S., Pang, J.: PAT: Towards flexible verification under fairness. In: International Conference on Computer Aided Verification. pp. 709–714. Springer (2009)
44. Sun, J., Song, S., Liu, Y.: Model checking hierarchical probabilistic systems. In: Proceedings of the 12th International Conference on Formal Engineering Methods and Software Engineering. p. 388–403. ICFEM'10, Springer-Verlag, Berlin, Heidelberg (2010)
45. Thorp, E.O.: Portfolio choice and the kelly criterion. In: Stochastic optimization models in finance, pp. 599–619. Elsevier (1975)
46. Wei, K., Woodcock, J., Burns, A.: Timed Circus: Timed CSP with the Miracle. In: 16th IEEE International Conference on Engineering of Complex Computer Systems, ICECCS 2011. pp. 55–64. IEEE Computer Society (2011)
47. Woodcock, J.: Software engineering mathematics. CRC Press (1988)
48. Woodcock, J., King, S., Sorensen, I.: Mathematics for specification and design: The problem with lifts. In: Fourth International Workshop on Software Specification and Design. pp. 265–268 (1987)
49. Woodcock, J., Cavalcanti, A., Foster, S., Mota, A., Ye, K.: Probabilistic Semantics for RoboChart. In: Unifying Theories of Programming. pp. 80–105. Springer International Publishing (2019)
50. Woodcock, J., Davies, J.: Using Z: specification, refinement, and proof. Prentice-Hall, Inc., USA (1996)
51. Ye, K., Cavalcanti, A., Foster, S., Miyazawa, A., Woodcock, J.: Probabilistic modelling and verification using RoboChart and PRISM. Software and Systems Modeling **21**(2), 667–716 (2022)
52. Zhu, H., Sanders, J., He, J., Qin, S.: Denotational semantics for a probabilistic timed shared-variable language. In: Unifying Theories of Programming, UTP 2012. Lecture Notes in Computer Science, vol. 7681, pp. 224–247. Springer (2012)
53. Zhu, H., Sun, J., Dong, J.S., Lin, S.: From verified model to executable program: the PAT approach. Innov. Syst. Softw. Eng. **12**(1), 1–26 (2016)