Problem solving using process algebra considered insightful

J.F. Groote (3-2196-6587) and E.P. de Vink (1-9514-2260)

Department of Mathematics and Computer Science,
Eindhoven University of Technology, Eindhoven, The Netherlands
J.F.Groote@tue.nl, E.P.d.Vink@tue.nl

Abstract. Process algebras with data, such as LOTOS, PSF, FDR, and mCRL2, are very suitable to model and analyse combinatorial problems. Contrary to more traditional mathematics, many of these problems can very directly be formulated in process algebra. Using a wide range of techniques, such as behavioural reductions, model checking, and visualisation, the problems can subsequently be easily solved. With the advent of probabilistic process algebras this also extends to problems where probabilities play a role. In this paper we model and analyse a number of very well-known – yet tricky – problems and show the elegance of behavioural analysis.

1 Introduction

There is great joy in solving combinatorial puzzles. Numerous books have appeared describing those [13,33]. And although some of the puzzles are easy to solve once properly understood, they are real brain teasers for most people.

Many of these puzzles are about behaviour. Classical mathematics and logic hardly provides an effective context to solve such problems systematically. This is apparent if one considers classical analysis. But also fields like graph theory, combinatorics, combinatorial optimisation, probability theory, and even logic all require a translation of the problem to the mathematical domain that is generally not completely straightforward.

This is where process algebras come in. Process algebras are very suited to describe the behaviour often present in the puzzles mentioned. In the last decades numerous tools have been developed to provide insight in the behaviour denoted in a process algebra expression as it quickly became clear that the behaviour described in such an expression can be rather intricate. This gave rise to hiding of actions, behavioural reductions, various visualisation techniques, as well as modal logics to express and validate properties about behaviour.

The early 1970s can be seen as the period when process algebra was born. Both Milner and Bekič wrote a treatise expressing that actions were important to study behaviour [2,25,27]. It was the seminal work of Milner in 1981 that put process algebras on the map [28]. This had quite some effect. For instance Hoare presented CSP in 1978 as an advanced programming language [21], whereas he

presented it in 1985 as a process algebra [22]. The work on CSP has been developed into the impressive family of tools, FDR, that are based on failure divergence refinement [31,14].

The work on CCS also inspired the design of the language LOTOS [24] as a language to model communication services and protocols. A major role in its development was played by the Technische Hogeschool Twente (now Twente University) first in the completely formal standardisation of the language, with Brinksma as main editor, and later in activities to build tools around it. Notable are the extensive formal specifications of standard protocols, but also those of manufacturing systems, that were developed at the time [5,7,32]. The CADP toolset stems from this period [12]. It is the only major toolset still capable of analysing LOTOS specifications. Furthermore, it has become quite powerful throughout the years.

The Algebra of Communicating Processes (ACP) was developed in Amsterdam [3,4] around the same time. In order to model practical systems first PSF (Protocol Specification Formalism) was designed [26], which was followed by the simpler formalism μCRL [18], later renamed to mCRL2, which was also directed towards analysis of practical specifications [17]. All these LOTOS-like formalisms use data based on abstract equational datatypes. mCRL2 also supports time and these days also probabilities.

An important feature of mCRL2 is the support for a modal logic with time and data, which is very useful to investigate properties of the described behaviour. Temporal logic, with the operators $[F]$ and $[P]$, stems from [30]. Pnueli pointed to the applicability of formal logics to analyse behaviour [29]. For mCRL2 we are using the modal mu-calculus which is essentially Hennessy-Milner logic [20] with fixed points [23]. An alternative is the use of linear time logic (LTL [29]) or computational tree logic (CTL [8]), but these are far less expressive than the modal mu-calculus [15].

In this paper we show process algebraic models of a number of well-known mathematical puzzles. Most people find them hard to solve when they are confronted with them for the first time. We show that the puzzles can straightforwardly be modelled into process algebra and using the standard analysis tools, such as behavioural reduction, model checking, and visualisation, the solutions to these puzzles are easy to obtain.

The major observation is that process algebra is an industrious mathematical discipline in itself due to its capacity to understand worldly phenomena. Traditionally, there is a tendency to think that process algebras, or more generally formal methods, are intended to analyse software, protocols, and complex distributed algorithms. But the application to examples as in this paper shows that process algebra has an independent stand.

In this paper we use the language mCRL2, as we are acquainted with it, and it offers all we need, namely the capacity to express behaviour, data structures, probabilities, time (although we do not exploit time here), and modal formulas. mCRL2 has a very rich toolset offering a whole range of analysis methods, far more than we use for the examples in this article. In the following we do

not explain the tool nor the formalism. For this we refer to [17] or the webpage `www.mcrl2.org`. The examples in this article are part of the mCRL2 distribution downloadable from this website.

2 The problem of the wolf, goat, and cabbage

A problem that is well-known, at least to the people in Western Europe, is the problem of the wolf, the goat, and the cabbage: A traveller walks through stretched Russian woods together with a friendly wolf, a goat, and a cabbage. Hungry and worn out, this companionship arrives at a river that they must cross. There is a small boat only sufficient to carry our traveller and either the wolf, the goat, or the cabbage. More than two do not fit. Crossing is complex as when left unsupervised by the traveller, the wolf will eat the goat, while the goat will eat the cabbage. The question to answer is whether it is possible to cross the river without the goat or the cabbage being eaten.

This problem is quite old. It already appeared in a manuscript from the eighth century A.D. [1]. Dijkstra wrote one of his well-known EWDs addressing this problem [10]. The description in mCRL2 can be found in Table 1. The description uses two shores, *left* and *right*, which are essentially sets of 'items', i.e. sets of wolf, goat, and/or cabbage, resting at that shore. The opposite shore is given by a function *opp*. An update function is used to remove items from one side and add it to the other.

The behaviour of crossing the river is given by the process WGC. It has two parameters, namely the shores s comprised of the sets of items at each side of the river, and the current position p of the traveller. Observe that mCRL2 accommodates the use of data types such as sets which allows to neatly describe the shores as a pair of sets containing items. The first two pairs of lines of the WGC process express that if the wolf and the goat, or the goat and the cabbage are at the side opposite of the traveller, something is eaten, expressed by the action `is_eaten`. The symbol δ indicates that the process stops after this action. Note that actions are typeset in a different font for easy recognition.

The third group of lines of the process expresses that the traveller can move to the other shore alone, by performing the `move` action. To reduce the number of transitions somewhat, we only allow this when no item can be eaten. The fourth group of lines expresses that the traveller can transport one item from one shore to the other. The last group of lines states that if the complete companionship arrives at the right shore, the action `done` can take place. Initially, the traveller, wolf, goat, and cabbage are at the left shore.

As the state space of this behaviour is small, it can nicely be visualised. See Figure 1. At the top we find the initial state, which is green. The goal state is coloured blue at the bottom. All states where an action `is_eaten` can be done are coloured red. They go to the white deadlocked state. All labels `is_eaten` are removed for readability. States where nothing is eaten are green, yellow, or blue. It is easy to see that there are paths from the green to the blue state through yellow states by moving counter clock wise through the graph. One of such paths

$$
\begin{array}{l}
\textbf{sort}\ \ \mathit{Item} = \textbf{struct}\ \mathit{wolf} \mid \mathit{goat} \mid \mathit{cabbage};\\
\qquad \mathit{Position} = \textbf{struct}\ \mathit{left} \mid \mathit{right};\\
\qquad \mathit{Shores} = \textbf{struct}\ \mathit{shores}(\mathrm{Set}(\mathit{Item}), \mathrm{Set}(\mathit{Item}));\\
\\
\textbf{map}\ \ \mathit{opp} : \mathit{Position} \to \mathit{Position};\\
\qquad \mathit{items} : \mathit{Shores} \times \mathit{Position} \to \mathrm{Set}(\mathit{Item});\\
\qquad \mathit{update} : \mathit{Shores} \times \mathit{Position} \times \mathit{Item} \to \mathit{Shores};\\
\\
\textbf{var}\ \ s,t : \mathrm{Set}(\mathit{Item});\\
\qquad i : \mathit{Item};\\
\textbf{eqn}\ \ \mathit{opp}(\mathit{left}) = \mathit{right},\ \mathit{opp}(\mathit{right}) = \mathit{left};\\
\qquad \mathit{items}(\mathit{shores}(s,t),\mathit{left}) = s,\ \mathit{items}(\mathit{shores}(s,t),\mathit{right}) = t;\\
\qquad \mathit{update}(\mathit{shores}(s,t),\mathit{right},i) = \mathit{shores}(s - \{i\}, t + \{i\});\\
\qquad \mathit{update}(\mathit{shores}(s,t),\mathit{left},i) = \mathit{shores}(s + \{i\}, t - \{i\});\\
\\
\textbf{proc}\ \ \mathit{WGC}(s : \mathit{Shores}, p : \mathit{Position}) =\\
\qquad\quad \{\mathit{wolf}, \mathit{goat}\} \subseteq \mathit{items}(s, \mathit{opp}(p)) \to\\
\qquad\qquad \mathtt{is\_eaten}(\mathit{goat}) \cdot \delta +\\
\qquad\quad \{\mathit{goat}, \mathit{cabbage}\} \subseteq \mathit{items}(s, \mathit{opp}(p)) \to\\
\qquad\qquad \mathtt{is\_eaten}(\mathit{cabbage}) \cdot \delta +\\
\qquad \neg(\{\mathit{wolf}, \mathit{goat}\} \subseteq \mathit{items}(s, \mathit{opp}(p))) \wedge \neg(\{\mathit{goat}, \mathit{cabbage}\} \subseteq \mathit{items}(s, \mathit{opp}(p))) \to\\
\qquad\qquad \mathtt{move}(\mathit{opp}(p)) \cdot \mathit{WGC}(s, \mathit{opp}(p)) +\\
\qquad\quad \textstyle\sum_{i:\mathit{Item}} .(i \in \mathit{items}(s,p)) \to\\
\qquad\qquad \mathtt{move}(\mathit{opp}(p), i) \cdot \mathit{WGC}(\mathit{update}(s, \mathit{opp}(p), i), \mathit{opp}(p)) +\\
\qquad\quad \mathit{items}(s, \mathit{right}) \approx \{\mathit{wolf}, \mathit{goat}, \mathit{cabbage}\} \to\\
\qquad\qquad \mathtt{done} \cdot \delta;\\
\\
\textbf{init}\ \ \mathit{WGC}(\mathit{shores}(\{\mathit{wolf}, \mathit{goat}, \mathit{cabbage}\}, \emptyset), \mathit{left});
\end{array}
$$

Table 1. An mCRL2 description of the problem of the wolf, the goat, and the cabbage

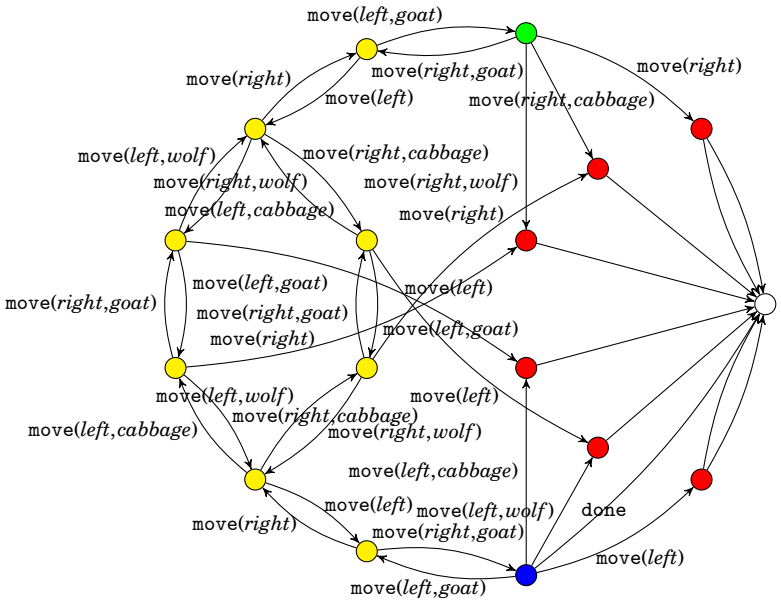

Fig. 1. The state space of the problem with the wolf, the goat and the cabbage

is

$$\texttt{move}(\mathit{right},\mathit{goat})\cdot\texttt{move}(\mathit{left})\cdot\texttt{move}(\mathit{right},\mathit{wolf})\cdot\texttt{move}(\mathit{left},\mathit{goat})\cdot \texttt{move}(\mathit{right},\mathit{cabbage})\cdot\texttt{move}(\mathit{left})\cdot\texttt{move}(\mathit{right},\mathit{goat})\cdot\texttt{done}.$$

Inspection of the state space also reveals that there is one other essential solution to this problem, namely one where the places of the wolf and the cabbage are exchanged. This is no surprise as the wolf and the cabbage have symmetrical roles. Note that it is also clear why this puzzle is considered tricky. Each solution requires the counter intuitive step of moving the goat three times across the river, an insight that requires humans to overcome their default mental set.

For this puzzle we are lucky that the number of states is sufficiently small to be depicted. In general this is not the case. Fortunately, modal formulas are a marvellous tool to investigate properties of behaviour. In this case we want to know whether there is a path from the initial state to a state where the action `done` is possible, while no action `is_eaten` is possible in any of the states on this path. In the modal mu-calculus as available in the mCRL2 toolset this is expressed by

$$\mu X.((\langle\text{true}\rangle X \vee \langle\texttt{done}\rangle\text{true}) \wedge \neg\langle\exists i{:}\mathit{Item}.\texttt{is\_eaten}(i)\rangle\text{true}).$$

The use of the minimal fixed point guarantees that the action `done` must be reached in a finite number of steps. The modality $\langle\text{true}\rangle$ says that an arbitrary action can be done. Checking this formula instantly yields true confirming that the traveller can safely reach the other shore with all the companions intact.

3 Crossing a rope bridge in the dark

The second problem is similar in nature to the first but not as well-known. Four people of different age arrive at a rope bridge across a canyon in the night. They need to cross the bridge as quickly as possible. Each person has its own time to cross the bridge, namely, 1, 2, 5, and 10 minutes. Unfortunately, the bridge can only carry the weight of two persons simultaneously. To make matters worse, they only carry one flashlight. Crossing without the flashlight is impossible. So, the flashlight needs to be returned for others to cross. The question is to find the minimal time in which the group of people can cross the bridge.

The problem is modelled in mCRL2 in Table 2. The location of each person is now given by a function $location : Person \to Position$. The function update construction is used to change a function. The expression $location[p \to s]$ represents a new function which is equal to *location* except that person p is now mapped to position s. The parameter *time* records the total time to cross the bridge and *light_position* keeps track of the place of the flashlight.

The behaviour consists of three summands, and is a direct translation of the problem. The first summand expresses that if all people are at the far side, a `ready` action is done, reporting the time to cross. The second summand expresses that one person crosses the bridge, and the third summand indicates that two people move to the other side together.

Natural numbers in mCRL2 are specified using abstract data types and have no upper bound. This means that the state space of this problem is infinite as there are inefficient crossing strategies that can take arbitrarily large amounts of time. Although not strictly necessary, as mCRL2 is very suitable to investigate infinite state spaces, it is generally a wise strategy to keep state spaces finite and even as small as reasonably possible. Solving the problem naively, quickly leads to a crossing time of 19 minutes. We therefore limit the maximal crossing time to 20 minutes and focus on in the question whether crossing under 19 minutes is possible.

The generated state space is somewhat larger, namely 470 states and 1607 transitions, which disallows inspection as an explicit graph. Fortunately, we can use the tool `ltsview`, which can visualise the structure of large transition systems [16], in some case up to millions of states. Pictures made by `ltsview` appear to be rather pointless pieces of art at first glance, but when investigated, provide remarkable insight in the depicted behaviour.

The behaviour of crossing the rope bridge is depicted in Figure 2 at the left. The initial state is at the top. The layering corresponds to the number of crossings of the bridge. The individually visible states and structures that grow to the side of the picture indicate deadlocks, i.e., states where the crossing time exceeds 20. For instance, the states at the end of the outward moving structure at the top right indicate that the bound of 20 minutes can be exceeded in three crossings. The red disk (the one but lowest) is the disk containing the action `ready(17)`. There are no `ready` actions with a lower argument. This indicates that the bridge cannot be crossed in less than 17 minutes.

$$
\begin{array}{ll}
\textbf{sort} & \mathit{Position} = \textbf{struct}\ \mathit{this\_side} \mid \mathit{far\_side}; \\
& \mathit{Person} = \textbf{struct}\ p_1 \mid p_2 \mid p_3 \mid p_4; \\
 & \\
\textbf{map} & \mathit{travel\_time} : \mathit{Person} \to \mathbb{N}; \\
& \mathit{initial\_location} : \mathit{Person} \to \mathit{Position}; \\
& \mathit{other\_side} : \mathit{Position} \to \mathit{Position}; \\
& \mathit{max\_time} : \mathbb{N}; \\
 & \\
\textbf{var} & p : \mathit{Person}; \\
\textbf{eqn} & \mathit{initial\_location}(p) = \mathit{this\_side}; \\
& \mathit{travel\_time}(p_1) = 1;\ \mathit{travel\_time}(p_2) = 2; \\
& \mathit{travel\_time}(p_3) = 5;\ \mathit{travel\_time}(p_4) = 10; \\
& \mathit{other\_side}(\mathit{this\_side}) = \mathit{far\_side}; \\
& \mathit{other\_side}(\mathit{far\_side}) = \mathit{this\_side}; \\
& \mathit{max\_time} = 20; \\
 & \\
\textbf{proc} & X(\mathit{light\_position} : \mathit{Position}, \mathit{location} : \mathit{Person} \to \mathit{Position}, \mathit{time} : \mathbb{N}) = \\
& \quad \mathit{time} \leq \mathit{max\_time} \wedge \forall p{:}\mathit{Person}.\mathit{location}(p) \approx \mathit{far\_side} \to \\
& \quad\quad \mathtt{ready}(\mathit{time}) \cdot \delta + \\
& \quad \sum_{p:\mathit{Person}} \cdot \\
& \quad\quad \mathit{time} \leq \mathit{max\_time} \wedge \mathit{location}(p) \approx \mathit{light\_position} \to \\
& \quad\quad\quad \mathtt{move}(p, \mathit{other\_side}(\mathit{location}(p))) \cdot \\
& \quad\quad\quad\quad X(\mathit{other\_side}(\mathit{light\_position}), \\
& \quad\quad\quad\quad\quad \mathit{location}[p \to \mathit{other\_side}(\mathit{location}(p))], \\
& \quad\quad\quad\quad\quad \mathit{time} + \mathit{travel\_time}(p)) + \\
& \quad \sum_{p,p':\mathit{Person}} \cdot \\
& \quad\quad p \not\approx p' \wedge \mathit{time} \leq \mathit{max\_time} \wedge \mathit{location}(p) \approx \mathit{light\_position} \wedge \\
& \quad\quad\quad\quad \mathit{location}(p') \approx \mathit{light\_position} \to \\
& \quad\quad\quad \mathtt{move}(p, p', \mathit{other\_side}(\mathit{location}(p))) \cdot \\
& \quad\quad\quad\quad X(\mathit{other\_side}(\mathit{light\_position}), \\
& \quad\quad\quad\quad\quad \mathit{location}[p \to \mathit{other\_side}(\mathit{location}(p))][p' \to \mathit{other\_side}(\mathit{location}(p'))], \\
& \quad\quad\quad\quad\quad \mathit{time} + \max(\mathit{travel\_time}(p), \mathit{travel\_time}(p'))); \\
 & \\
\textbf{init} & X(\mathit{this\_side}, \mathit{initial\_location}, 0);
\end{array}
$$

Table 2. The problem of crossing a rope bridge specified in mCRL2

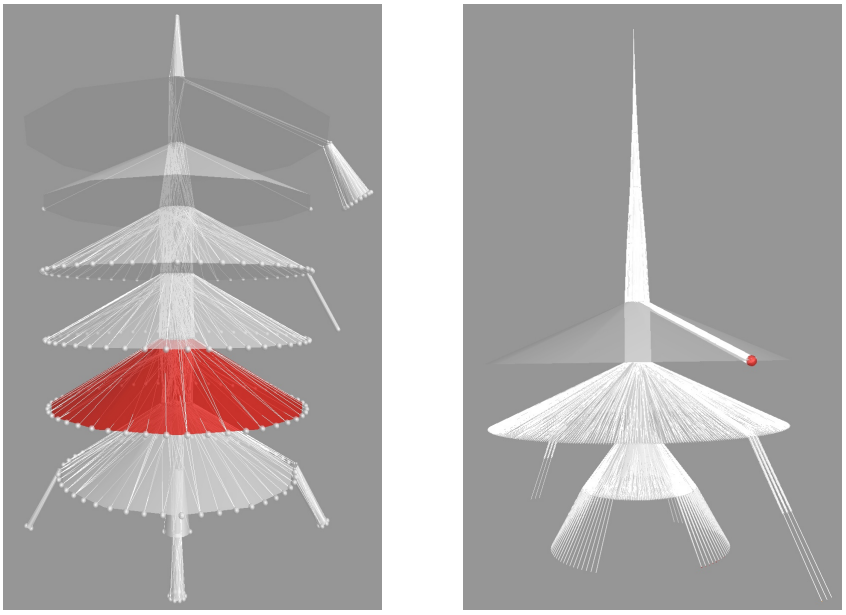

Fig. 2. An `ltsview` visualisation of crossing a rope bridge and the game tic-tac-toe

`ltsview` is not the most efficient way to inspect which `ready` actions are possible. By searching for actions while generating the state space it becomes immediately clear that the actions `ready`(17), `ready`(19) and `ready`(20) are possible. A trace to `ready`(17) is

$$\texttt{move}(p_2,p_1,\mathit{far\_side})\cdot\texttt{move}(p_1,\mathit{this\_side})\cdot\texttt{move}(p_4,p_3,\mathit{far\_side})\cdot \texttt{move}(p_2,\mathit{this\_side})\cdot\texttt{move}(p_2,p_1,\mathit{far\_side})\cdot\texttt{ready}(17).$$

This trace shows why this puzzle is hard to solve. The idea to save time to let the two slowest persons cross simultaneously does not easily come to mind for most people.

Using modal logics we can also easily check that 17 is the most optimal crossing time. The next formula, which says that there is a path to the action `ready`(17) and not to any action `ready`(n) for any $n < 17$, is readily proven to hold:

$$\langle \text{true}^\star\cdot\texttt{ready}(17)\rangle\,\text{true} \wedge \forall n{:}\mathbb{N}.(n < 17 \rightarrow [\text{true}^\star\cdot\texttt{ready}(n)]\text{false}).$$

4 A winning strategy in tic-tac-toe

Finding winning strategies in games can also be neatly expressed and studied in process theory. One of the simplest well-known games that can be analysed in this way is tic-tac-toe. Essentially, tic-tac-toe consists of a 3 by 3 board where two

$$
\begin{array}{ll}
\textbf{sort} & \mathit{Piece} = \textbf{struct}\ \mathit{empty} \mid \mathit{naught} \mid \mathit{cross}; \\
& \mathit{Board} = \mathbb{N}^+ \to \mathbb{N}^+ \to \mathit{Piece}; \\
 & \\
\textbf{map} & \mathit{empty\_board} : \mathit{Board}; \\
& \mathit{did\_win} : \mathit{Piece} \times \mathit{Board} \to \mathbb{B}; \\
& \mathit{other} : \mathit{Piece} \to \mathit{Piece}; \\
 & \\
\textbf{var} & b : \mathit{Board}; \\
& p : \mathit{Piece}; \\
& i, j : \mathbb{N}^+; \\
\textbf{eqn} & \mathit{empty\_board}(i)(j) = \mathit{empty}; \\
& \mathit{other}(\mathit{naught}) = \mathit{cross};\ \mathit{other}(\mathit{cross}) = \mathit{naught}; \\
& \mathit{did\_win}(p, b) = \\
& \quad (\exists i : \mathbb{N}^+ . (i \leqslant 3 \wedge b(i)(1) \approx p \wedge b(i)(2) \approx p \wedge b(i)(3) \approx p)) \vee \\
& \quad (\exists j : \mathbb{N}^+ . (j \leqslant 3 \wedge b(1)(j) \approx p \wedge b(2)(j) \approx p \wedge b(3)(j) \approx p)) \vee \\
& \quad (b(1)(1) \approx p \wedge b(2)(2) \approx p \wedge b(3)(3) \approx p) \vee \\
& \quad (b(1)(3) \approx p \wedge b(2)(2) \wedge p \approx b(3)(1) \approx p); \\
 & \\
\textbf{proc} & \mathit{TicTacToe}(\mathit{board} : \mathit{Board}, \mathit{player} : \mathit{Piece}) = \\
& \quad \mathit{did\_win}(\mathit{other}(\mathit{player}), \mathit{board}) \to \\
& \qquad \mathtt{win}(\mathit{other}(\mathit{player})) \cdot \delta \\
& \quad \diamond \big(\textstyle\sum_{i,j:Pos} . (i \leqslant 3 \wedge j \leqslant 3 \wedge \mathit{board}(i)(j) \approx \mathit{empty}) \to \\
& \qquad \mathtt{put}(\mathit{player}, i, j) \cdot \\
& \qquad\quad \mathit{TicTacToe}(\mathit{board}[i \to \mathit{board}(i)[j \to \mathit{player}]], \mathit{other}(\mathit{player})) \big); \\
 & \\
\textbf{init} & \mathit{TicTacToe}(\mathit{empty\_board}, \mathit{cross});
\end{array}
$$

Table 3. An mCRL2 formalisation of tic-tac-toe

players alternatingly put a naught or cross at empty positions on the board. The first player that has three of naughts or crosses in a row, horizontally, vertically or diagonally, wins the game.

Table 3 contains a rather natural formalisation of this game. The playing board is given by a function from pairs of naturals to pieces. A less elegant formulation uses lists of lists of pieces, but for state space generation this is much faster. A player moves by putting its own piece at an empty position on the board using the action `put`. The action `win` is used to indicate that one of the players did win. The most complex function is $did\_win(p,b)$, checking whether player p, represented by a piece, did win the game. In the formalisation we use $c \to p \diamond q$ denoting 'if c then p else q'.

The total behaviour of this game has 5479 states and 17109 transitions, which is not very large. This behaviour is depicted in Figure 2 at the right. The red dot at the right middle indicates where player 'naught' can win. There are more such states two disks lower, but they are hardly visible in the figure.

Although the transition system for this game is relatively small, it makes no sense to investigate it directly to determine whether the player that starts the game has a winning strategy. Fortunately, modal formulas come to the rescue. The following formula expresses that player 'cross' has a winning strategy. It says that there is a way to put a cross on the board after which player 'cross' wins, or for every counter move by player 'naught', X must hold again, saying that also in that case player 'cross' has a winning strategy. This formula is invalid. There is no winning strategy for the player 'cross', and due to symmetry neither for player 'naught'.

$$\mu X.\langle \exists i,j:\mathbb{N}^+.\mathtt{put}(cross,i,j)\rangle \big(\langle \mathtt{win}(cross)\rangle \mathrm{true} \vee [\exists i,j:\mathbb{N}^+.\mathtt{put}(naught,i,j)]X \wedge \langle \mathrm{true}\rangle \mathrm{true}\big)$$

Note that we use of a minimal fixed point operator expressing that winning must happen within a finite number of steps. As there are only a limited number of moves in tic-tac-toe this is always satisfied, hence a maximal fixed point operator could also have been used.

5 The Monty Hall problem

Processes algebras have seen various extensions. One of these extensions is the addition of probabilities, which gives rise to the interesting combination of non-deterministic and probabilistic behaviour. This opens up the field of probabilistic puzzles to be modelled. The Monty Hall problem is a very nice example, because when understood is it very simple, yet most people fail to solve it properly.

The Monty Hall problem is a tv-quiz from the 1960s. A player can win a prize when he opens one of three doors with the prize behind it. Initially, the player selects a door with probability $\frac{1}{3}$. Subsequently, the quizmaster opens one of the remaining doors showing that it does not hide the prize. The question is whether the player should switch doors to optimise his winning probability.

The problem is expressed in the specification in Table 4. The process only consists of a single action player_collects_prize(b) where the boolean argument b is true if a prize is collected. The **dist** keyword is used to indicate a probability distribution. The process **dist** $x : S[D(x)].p$ indicates that variable x of sort S is selected with probability distribution $D(x)$. One of the doors hides the prize. This door is represented by the variable *door_with_prize* which can have values d_1, d_2, or d_3, each with a probability of $\frac{1}{3}$. Initially, the player selects a door. If the player decides to switch doors after the quizmaster opened a door, the player has a prize if and only if the initially chosen door did *not* carry the prize. This is expressed by the use of not equal sign ($\not\approx$) in the argument of the action. If the player decides to stick to the door that was initially selected, the not equal sign should be replaced by equality.

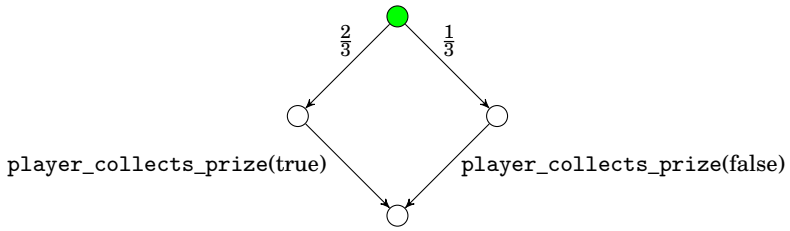

Fig. 3. The reduced probabilistic transition system for the Monty Hall problem

The resulting state space has 9 transitions each with a probability $\frac{1}{9}$. It is convenient to apply a probabilistic bisimulation reduction on the transition system. This leads to the reduced transition system in Figure 3. It is clearly visible that the action player_collects_prize(true) can be done with probability $\frac{2}{3}$. Thus, when switching doors the probability of obtaining a prize is $\frac{2}{3}$, opposed to $\frac{1}{3}$ when not switching doors.

sort *Doors* = **struct** d_1 | d_2 | d_3;
init **dist** *door_with_prize* : *Doors*[1/3].
dist *initially_selected_door_by_player* : *Doors*[1/3].
player_collects_prize(*initially_selected_door_by_player* $\not\approx$ *door_with_prize*)·δ;

Table 4. An mCRL2 specification of the Monty Hall quiz

```
map  N : ℕ⁺;
eqn  N = 100;

proc Plane(everybody_has_his_own_seat : 𝔹, number_of_empty_seats : ℕ) =
       (number_of_empty_seats ≈ 0) →
         last_passenger_has_his_own_seat(everybody_has_his_own_seat)·δ
       ◇ (enter·
         dist b0 : 𝔹[if(everybody_has_his_own_seat, if(b0, 1, 0),
                if(b0, 1 − 1/number_of_empty_seats, 1/number_of_empty_seats))].
           b0 → select_seat·
             Plane(everybody_has_his_own_seat, number_of_empty_seats − 1)
             ◇ dist b1 : 𝔹[if(b1, 1/number_of_empty_seats, 1−1/number_of_empty_seats)].
               select_seat·
                 Plane(if(number_of_empty_seats≈1, everybody_has_his_own_seat, b1),
                   number_of_empty_seats − 1));

init  dist b : 𝔹[if(b, 1/N, (N − 1)/N)].Plane(b, N − 1);
```

Table 5. An mCRL2 specification of the lost boarding pass

6 The problem of the lost boarding pass

More complex probabilistic problems can become rather hard even with the full strength of probability theory at ones disposal. Yet modelling the problem in mCRL2 is again pretty straightforward. The tools can subsequently help to obtain the required answer.

A particularly intriguing puzzle is that of the lost boarding pass as it has a remarkable answer, defying the intuition of most people trying to solve the problem: There is a plane with 100 seats. The first passenger boarding the plane lost his boarding ticket and selects a random seat. Each subsequent passenger will use his own seat unless it is already occupied. In that case he also selects a random seat. The question is what the probability is that the *last* passenger entering the plane will sit in his own seat.

The behaviour is modelled in Table 5. The number N is the number of seats, which is set to 100. The behaviour of entering the plane is characterised by two parameters. The parameter *number_of_empty_seats* indicates how many seats are still empty in the plane. The parameter *everybody_has_his_own_seat* indicates that all remaining seats correspond exactly with the places for all passengers

that still have to board the plane. Except if the number of empty seats is 0. In that case it indicates whether the last passenger got its own seat.

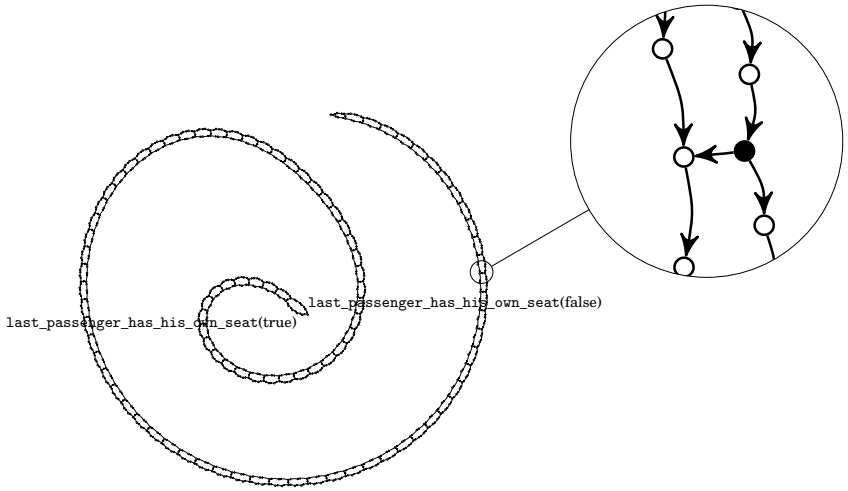

Fig. 4. The state space of the problem of the lost boarding pass with 100 passengers

Initially the first passenger selects his seat at random. With probability $\frac{1}{N}$ he will end up at his own seat. This corresponds with the situation where b is true. In the main process *Plane*, when all passengers have boarded the plane, the action `last_passenger_has_his_own_seat` indicates by its argument whether the last passenger got his own seat. If not all passengers boarded the plane yet, a next passenger enters (indicated by the action `enter`) and then it can either be that he finds his own seat free (b_0 is true) or occupied (b_0 is false). If everybody is sitting at is own seat this next passenger will for sure find his own seat free. Otherwise, he finds his own seat free with probability $1-1/number\_of\_empty\_seats$ as exactly one person is sitting on a wrong seat.

When this next passenger finds his own seat free he can sit down. This is done by the action `select_seat` with two parameters. But if his own seat is occupied, he must randomly select a seat for himself. If he selects the seat such that all passengers are sitting on their assigned seats (modulo a permutation) this is indicated in the variable b_1, where this passenger has probability

$$1/number\_of\_empty\_seats$$

of doing this.

The generated state space turns out to be linear in the size of the number of seats. It has 791 states and 790 transitions. Modulo strong probabilistic bisimulation there are 399 states and 398 transitions. It has the shape of a long sequence, as depicted in Figure 4. Detailed exploration of this figure indicates that whence

all the remaining passengers correspond to the remaining seats the last passenger will certainly get his own seat. Yet it is not obvious what the probability for the last passenger to get his own seat is. For this we use two – at present experimental – tools[1]. The first one applies a probabilistic weak trace reduction. The obtained state space, see Figure 5, is rather non-exciting but indicates clearly that the probability of the last passenger to end up at its own seat is $\frac{1}{2}$. The remarkable property of this exercise is that this probability is independent of the number of seats.

There is another way to obtain this probability by employing modal formulas over reals. These formulas are derived from the modal mu-calculus but deliver a real number, instead of a boolean. In this case the formula is just

$$\langle \mathrm{true}^{\star} \cdot \texttt{last\_passenger\_has\_his\_own\_seat}(\mathrm{true}) \rangle \, \mathrm{true}$$

which is possible as the state space is deterministic. Needless to say that the verification of this formula yields $\frac{1}{2}$ as well.

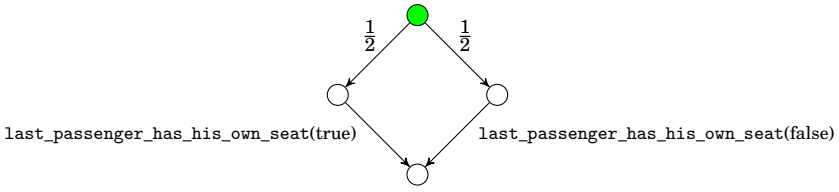

Fig. 5. The state space of the lost boarding pass problem modulo weak trace equivalence

7 Concluding remarks

Process algebra is generally well-suited to solve many behaviour-oriented mathematical puzzles. In this paper we have used the process algebraic framework of mCRL2 to show how to model a number of such puzzles. Subsequently, the standard analysis tools available in mCRL2 (and occasionally an experimental one) were used for behavioural reduction, model checking and visualisation. From this it is clear that process algebra has a wider scope than the usual fields of software analysis and distributed computing in which it finds many applications.

Process algebra focuses on behavioural aspects of the subject of study. The underpinning algebraic and equational theory allows to relate to logics, in particular modal logics [6], as descriptions of properties or requirements over space, time, and probabilities. Logical characterisations and their assessment via model

[1] The tools are by Olav Bunte (evaluation of modal formulas on probabilistic transition systems) and Ferry Timmers (probabilistic trace reduction).

checking are a valuable replacement in situations where visual techniques, highlighted for the puzzles discussed here, become impractical.

Also other authors indicated that a notion of behaviour or state space is required for proper conditional reasoning, especially in the probabilistic setting. In [19] the distinction is made between 'naive' and 'sophisticated' space. For the Monty Hall puzzle this amounts to the three doors for the naive space, and to sequences of events for the sophisticated space. In the process algebraic modelling of the problem, it is exactly the latter that is determined by the specified behaviour, thus making the underlying protocol explicit.

Although we defend the use of process algebra as a qualitatively better approach to solving behavioural problems, this is a subjective opinion, influenced by our experience with process algebras. To substantiate this in a more objective manner one should measure how much time people need to solve particular problems with particular techniques, for instance by psychological tests.

If process algebraic techniques become commonplace, it might be that the nature of 'tricky' puzzles will shift where the proper behaviour is not directly obvious. Nice examples are for instance Freudenthal problems, containing knowledge, like the Muddy Children puzzle [11]. Translating knowledge into behaviour often requires a twist. In such cases dynamic epistemic logic might be more suitable [9].

Acknowledgement. The authors are grateful to the reviewers for their constructive and inspiring comments.

References

1. Alcuinus Flaccus. Propositiones ad Acuendos Juvenes. Manuscript. 780.
2. H. Bekič. Towards a mathematical theory of processes. Technical Report TR25.125, IBM Laboratory, Vienna, 1971. Also appeared in Programming Languages and Their Definition, C.B. Jones (ed.), Lecture Notes in Computer Science 177, Springer, 1984.
3. J.A. Bergstra and J.W. Klop. Fixed point semantics in process algebras. Report IW 206, Mathematisch Centrum, Amsterdam, 1982.
4. J.A. Bergstra and J.W. Klop. Process algebra for synchronous communication. Information and Computation 60(1/3):109–137, 1984.
5. F. Biemans and P. Blonk. On the formal specification and verification of CIM architectures using LOTOS. Computers in Industry 7(6), 491–504, 1986.
6. P. Blackburn, J. van Benthem, and F. Wolter (eds.). Handbook of Modal Logic. Studies in Logic and Practical Reasoning volume 3. Elsevier, 2007.
7. E. Brinksma and G. Karjoth. A specification of the OSI transport service in LOTOS. In Protocol Specification, Testing and Verification IV. Y. Yemini, R.E. Strom and S. Yemini (eds), pp. 227–251. North-Holland, 1984.
8. E.M. Clarke and E.A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In Logic of Programs, D. Kozen (ed.), pp. 52–71. Lecture Notes in Computer Science 131, Springer, 1981.
9. H. van Ditmarsch, W. van der Hoek, and B. Kooij. Dynamic Epistemic Logic. Studies in Epistemology, Logic, Methodology, and Philosophy of Science volume 337. Springer, 2008.
10. E.W. Dijkstra. Pruning the search tree. EWD1255. Available at `www.cs.utexas.edu/users/EWD/transcriptions/EWD12xx/EWD1255.html`. Accessed June 2017.

11. P. van Emde Boas, J. Groenendijk, and M. Stokhof. The Conway paradox: Its solution in an epistemic framework. In Truth, Interpretation, and Information: Selected Papers from the Third Amsterdam Colloquium. J. Groenendijk, T.M.V. Janssen, and M. Stokhof (eds.), pp. 159–182. Foris Publications, 1984.
12. H. Garavel, F. Lang, R. Mateescu, and W. Serwe. CADP 2011: A toolbox for the construction and analysis of distributed processes. International Journal on Software Tools for Technology Transfer 15(2):89–107, 2013
13. M. Gardner. My best mathematical and logic puzzles. Dover, 1994.
14. T. Gibson-Robinson, P. Armstrong, A. Boulgakov, and A.W. Roscoe. FDR3 — A Modern Refinement Checker for CSP. In Tools and Algorithms for the Construction and Analysis of Systems, E. Abraham and K. Havelund (eds.), pp. 187–201. Lecture Notes in Computer Science 8413, 2014.
15. S. Cranen, J.F. Groote, and M.A. Reniers. A linear translation from CTL* to the first-order modal mu-calculus. Theoretical Computer Science 412(28):3129–3139, 2011.
16. J.F. Groote and F. van Ham. Interactive visualization of large state spaces. International Journal on Software Tools for Technology Transfer 8(1):77–91, 2006.
17. J.F. Groote and M.R. Mousavi. Modeling and Analysis of Communication Systems. The MIT Press 2014. (See for the toolset `www.mcrl2.org`).
18. J.F. Groote and A. Ponse. The syntax and semantics of μCRL. Report CS-R9076, CWI, Amsterdam, 1990.
19. P. Grünwald and J.Y. Halpern. Updating Probabilities. In Uncertainty in Artificial Intelligence, A. Darwiche and N. Friedman (eds.), pp. 187–196. Morgan Kaufman, 2002.
20. M.C.B. Hennessy and R. Milner. On observing nondeterminism and concurrency. In Automata, Languages and Programming (ICALP'80), J.W. de Bakker and J. van Leeuwen (eds.), pp. 299–309. Lecture Notes in Computer Science 85, Springer, 1980.
21. C.A.R. Hoare. Communicating sequential processes. Communications of the ACM 21(8):666–677, 1978.
22. C.A.R. Hoare. Communicating sequential processes. Prentice Hall International, 1985.
23. D. Kozen. Results on the propositional μ-calculus. Theoretical Computer Science, 27:333–354, 1983.
24. ISO 8807:1989. Information processing systems – Open Systems Interconnection – LOTOS – A formal description technique based on the temporal ordering of observational behaviour. ISO/IECJTC1/SC7 1989.
25. R. Milner. An approach to the semantics of parallel programs. In Proceedings Convegno di Informatica Teorica, Pisa, pp. 283–302, 1973.
26. S. Mauw and G.J. Veltink. A process specification formalism. Fundamenta Informaticae, XIII:85–139, 1990.
27. R. Milner. Processes: A mathematical model of computing agents. In Proceedings Logic Colloquium 1972, H.E. Rose and J.C. Shepherdson (eds.), pp. 158–173. North-Holland, 1973.
28. R. Milner. A calculus of communicating systems. Lecture Notes in Computer Science 92, Springer, 1979.
29. A. Pnueli. The temporal logic of programs. In Foundations of Computer Science, pp. 46-57. IEEE, Piscataway, 1977.
30. A.N. Prior. Time and modality. Oxford University Press, 1957.
31. A.W. Roscoe. Understanding concurrent systems. Springer, 2010.
32. M. van Sinderen, I. Ajubi, and F. Caneschi. The application of LOTOS for the formal description of the ISO session layer. In Formal Description Techniques, K.J. Turner (ed.), pp. 263–277. North-Holland, 1989.
33. P. Winkler. Mathematical Puzzles. A connaisseur's collection. A.K. Peters, 2004.