Comparing Versions of Banker’s Algorithm for Deadlock Avoidance in Resource Allocation Systems

Michal Žarnay\textsuperscript{1} and Jozef Marek\textsuperscript{2}

\textsuperscript{1} University of Žilina, Žilina, Slovak Republic, Michal.Zarnay@fri.uniza.sk
\textsuperscript{2} Smart Engine GmbH, Vienna, Austria, jozfmarek@gmail.com

\textbf{Abstract.} Deadlocks are undesirable states in resource allocation systems (RAS), a theoretical paradigm used for modelling of several application systems, e.g. flexible manufacturing systems or transportation systems. Avoiding deadlocks represents a major issue in control of RAS. The banker’s algorithm is one of available methods for this. It has several versions with different level of complexity, effectiveness and efficiency. The paper presents some versions of the algorithm and a methodology for their comparison using the tools EBZ and CoBA, specific tools designed for this purpose, and CPN Tools, a tool for modelling coloured Petri nets.

1 Introduction

\textit{Deadlock state} is a state of a system, where two or more system processes are blocked in their execution by waiting for resources that are occupied at the same time by the processes in the waiting list. The waiting processes thus block and are blocked. Unblocking this state is possible only by an exceptional operation.

The deadlock states are undesirable in any system since they attack system effectiveness. They occur as a result of allocation of resources to processes in certain states of the system. Formalism used for analysing and solving the problem is called Resource Allocation System (RAS). Based on several parameters, RAS can belong to various levels of complexity. The latter decides about applicable control methods for avoiding deadlock states. In this paper, we deal with the most complex RAS and a well-known control method: the banker’s algorithm (BA).

There are several versions of the BA in literature which differ in complexity and effectiveness. Every version accepts a limited number of states, i.e. the operation scope of the underlying RAS is restricted. The basic version restricts the most, while it is expected to be the fastest. More elaborated versions enlarge the number of accepted states for the price of longer calculation for some states. Based on our objective, we may need a faster or a more effective version. Out of the interest to find out what is the ratio between these two parameters, we have built a framework to compare the versions on concrete examples of the RAS.
In the following section, we describe necessary details for the two discussed topics: resource allocation systems and Banker’s algorithm, that are completed by a brief description of Petri nets, a formalism used to depict and discuss the problems. Section 3 explains the comparison framework followed by examples in section 4 and final conclusion.

2 Preliminaries

2.1 Petri net as modelling environment

The Petri net (PN) is a formalism used for modelling and analysis of systems with concurrent processes. It has graphical notation, precise mathematical language and analysis methods for specifying the system behaviour. Basic construction elements of PN are places, transitions, directed arcs and tokens. Places and transitions are two types of nodes in the net. Directed arcs link places with transitions, while no pair of nodes of the same type can be connected. Tokens are elements that move in the created network between places through arcs and transitions. Principal difference between places (drawn as circles) and transitions (rectangles) lies in their relation to tokens (black dots). Places can be marked with tokens. The number of tokens and their distribution in places represent the state of the net. Transitions take tokens from input places (i.e. places from which directed arcs lead to a transition) and send tokens to output places (i.e. places to which directed arcs lead from the transition) - number of transferred tokens is determined by arc weights. This process is called firing of transition - it performs an action in the net, changing its state. In this way, it is also possible to change the overall number of tokens in the net. More details as well as precise definition of Petri nets can be found in [4].

The basic PN formalism, called Place/Transition (P/T) net, is often enriched or restricted to obtain enhancements or subclasses of PN. In this paper, we also mention coloured Petri net (CPN), as it is introduced in [2]. Colour is a symbolic name for value added to a token in the CPN, what distinguishes individual tokens (they are not black dots anymore). This requires additional specification for places (colour sets - types of tokens allowed), transitions (binding elements - combinations of transition name and colours of tokens for firing), arcs (arc inscriptions defining token colours to pass on the arc) and markings by means of the CPN ML language. This enhancement allows construction of models with simpler net structure and added description, while keeping the same modelling power as it would be with the basic P/T net.

The fig. 1 contains a coloured Petri net. All the places $P_x$ ($x = 0..9$) are of the colour set $cProcessID$ that is an integer colour set with values 1 and 2. The place $Resources$ is of colour set $cResources$ - enumerated colour set with identifiers $R1$, $R2$ and $R3$. Arc inscriptions contain either a variable $p$ of the $cProcessID$ colour set or multiset expressions of the $Resources$ colour set, e.g. $1'R1++1'R3$ meaning two tokens, one of colour $R1$ and one of colour $R3$. Initial markings of places $P_x$ and $Resources$ have the form of multiset expressions.
2.2 Resource Allocation System

*Resource allocation system* (RAS) is a system consisting of concurrently running processes that in certain stages, in order to get successfully completed, require exclusive use of certain number of system resources [5]. Resources are limited and re-usable as their allocation and de-allocation changes neither their character nor quantity. Main purpose of RAS is to solve problems arising from allocation of resources to processes.

The coloured Petri net at the fig. 1 depicts a RAS that we will use for illustration. In such a CPN, subnets of two types are found: process and resource subnets. A *process subnet* consists of places, transitions and arcs in a structure starting by an initial transition (T1 in the fig.) and ending with a final transition (T9) and describing causal relations between stages of a process. A stage (a task) in the process corresponds to a place and beginning and ending events of

---

**Fig 1.** Illustration model in the state No. 5.
a task correspond to transitions, e.g. a task is modelled by a place $P_5$ between transitions $T_3$ and $T_6$. Together with a place for idle processes ($P_0$), which connects the final transition with the initial transition of the process description, it makes a strongly connected component. A resource subnet consists of one place ($Resources$) and its adjacent arcs. Content of the place represents actually free resources and arcs express their allocation and de-allocation to and from stages of processes. Typically, there are several process subnets, one for each modelled process type (the example net has just one), and one resource subnet in a RAS model.

Based on its character, a process in the RAS can be sequential or non-sequential, i.e. some parts of the process can run concurrently. The resulting system is then either sequential (if all involved processes are sequential) or not (if at least one process in the system is non-sequential). The RAS of the fig. 1 is in the latter category - it contains one non-sequential process. The concurrent processing is represented by branches starting from transition $T_1$ (a second, nested branching off is at $T_2$) and ending in $T_9$ ($T_8$).

Further a process can contain flexible routing. It means that in a certain moment, processing continues in one of available options, we call them variants. They are created, when at least one place has at least two output transitions (conflict in Petri net, like in state machines) and another place has at least two input transitions, e.g. places $P_1$ and $P_8$. All possible routes in the process subnet must contain the place $P_0$, so taking any of variants in flexible routing brings the process instance to the same final state.

Finally, number and character of resources allocated at the same time distinguishes between single-unit RAS (every process is allowed to have only one resource unit allocated at a time), single-type RAS (at least one process can have more units of the same resource type at a time) or multiple-type RAS (at least one process has units of more than one resource type at a time) which is the case of the example RAS: $R_1$, $R_2$ and $R_3$ represent 3 types of resources and e.g. in the stage of the place $P_5$, the process instance has allocated one unit of every resource type.

The outlined attributes of concurrent processing, flexible routing and resource allocation together create categories of processes and the whole RAS with varied complexity [6]. Category of the RAS decides on methods to be used for deadlock solving. The BA discussed in this paper works also for the most complex RAS: system with non-sequential processes, flexible routing and multiple-type resources allocation.

For a description of the system’s dynamic behaviour, we’ll distinguish between process types and process instances. The process type is an abstract description of a process. The process instance is a concrete occurrence of a process according to a process type. In the CPN, the process type is modelled by the process subnet and the process instance by one or more tokens of one colour. A position of tokens of one colour in places of the process subnet represents a stage of the process. Similarly, there are resource types and resource instances. The former modelled by colours of a colour set for all resources (one resource type
corresponds to one colour) and the latter by individual coloured tokens (number of tokens of a colour corresponds to number of resources of the respective resource type).

One can notice a difference: resources instances of the same type are not distinguished, while process instances of the same type are distinguished by different colours. The latter (making the state space larger) is necessary for the deadlock solving algorithm that in certain stages must recognize exact position and allocation of resources to the process instance to work with.

2.3 Banker’s Algorithm

The banker’s algorithm (BA), first introduced in [1], uses information about a current system state to decide, whether a process allocation request can be fulfilled. It is called every time, when an allocation request is made. It checks, if the allocation leads to a safe state that is a state, from which deadlock states can be avoided. If it does lead, the request can be fulfilled, otherwise, the requesting process must wait until another process returns resources.

In order to decide about the state’s safeness, the BA tries to order all active (i.e. non-idle) processes in such a sequence, so that each of the processes can be finished with resources that it currently occupies, those that are currently available in the system and those that are already returned from processes finished in the sequence prior to the tested process. If it succeeds in finding such a sequence, we say that the state is ordered (or the state has been accepted by the BA), and since every ordered state is safe [3], the state is also safe. If it fails to find the sequence, we say that the state is unordered (or the state has been rejected by the BA), which does not mean that the state is unsafe. However, the allocation request cannot be fulfilled. This is due to the suboptimality of the BA, because deciding the state safety is a NP-complete problem. The introduced in a basic version, we will call it banker’s algorithm A (BA-A).

Several modifications of banker’s algorithm exist with the primary goal to accept more safe states as ordered (i.e. to be more effective) and to do its calculations in the shortest time possible (i.e. to be more efficient). We took ideas introduced in [3] for sequential RAS and we work with following modifications of BA, implemented for RAS with non-sequential processes:

- Algorithm verifying partially ordered states, i.e. states where it is enough to order processes up to the process currently requiring resources. We call it algorithm B (BA - B)
- Algorithm advancing processes - an active process is advanced forward and the new state again checked with the BA - A - all that in iterations until an ordered state is found or the advanced process cannot be moved forward any more. When it cannot be moved any more and the state is still not ordered, the same algorithm $V_m$ is called and another active process in the current state, if there is any, is taken to advance. If an ordered state is found, the state is called $V_m$-ordered, where $V_m$ means a set of states that have been ordered after moving $m$ processes forward (i.e. $V_0$-ordered states are accepted by the BA - A). The algorithm will be called $V_m(BA - V_m)$. 
While the BA-B has the same theoretical complexity as the BA - A, practically, it is more efficient in those states, where a partially ordered state can be found. The BA - V_m algorithm has worse theoretical complexity, but, based on the underlying RAS model, it can admit more states.

Furthermore, BA versions differ in the following parameters (developed from ideas introduced in [7] and in chapter 4 of [6] for sequential RAS with flexible routing):

1. Information about remaining needed resources (shortly RemainNeed) for a process - BA uses this information to decide whether the process can be finished. There are several implementation options: from the raw way of having one overall vector of all resource instances needed in the whole process execution calculated before the process starts to the finest way of having individual vectors of resource instances needed till the end of process execution at every stage of route (variant, branch). The latter can be either static - calculated off-line for the whole process type description beforehand - or dynamic - calculated on-line as the process instance advances.

2. Default variant strategy is used in the case of flexible routing for deciding which variant’s RemainNeed is used for calculations as the primary, for instance deciding between variants starting with transitions T2 or T3 in the example at the fig. 1.

3. Process choice order strategy for situations, when a process for advancing is to be chosen from a set of active processes. It is also used by choosing a process for ordering in the BA-A, there however, it has no implication on effectiveness (deciding whether the state is ordered), it may influence efficiency (number of unsuccessful tries to order a process instance)

4. Transitions choice strategy by process advancing for situations, when a transition is to be chosen from a set of available transitions.

3 Comparing Versions of Banker’s Algorithm

3.1 EBZ

An editor made by Branislav Zríny (EBZ) is a specialized simple editor and simulator for RAS models that among other things allows creating and editing a RAS model in Petri net and saving it in own PN file format.

3.2 CoBA

Our CoBA (Comparison of versions of the Banker’s Algorithm) tool has been designed with the goal to supply the missing functionality for enabling the comparison of different versions of the BA for deadlock avoidance in RAS. It is based on representation of RAS in Petri net, including its simulation. The tool is able to:
Read the RAS model in the \textit{PN} and \textit{PN2} file formats, where \textit{PN} is an older format used by EBZ and \textit{PN2} is built upon the \textit{PN} format and used in CoBA.

Display the net according to coordinates of nodes and arcs.

Export the net to \textit{CPN} format of \textit{CPN Tools}.

Check the admissiveness of states with chosen BA version and RAS parameters. Tests can be performed for one state, batch of selected states and as part of simulation.

Simulate the net with chosen BA version and RAS parameters or without it, where the simulation is restricted by length of system time or number of performed steps in underlying Petri net.

Available parameters for calculation of BA are those named in the section 2.3. By the advancement of processes in the \textit{BA} \(-V_m\), the nesting of advanced processes continues at most \(m\) times, where the parameter \(m\) is determined before the test starts and forms a restriction to avoid too long calculations.

In addition to them, there is an option to repeat testing of process instances in the same state, which is designed for testing efficiency of the BA versions' implementation. If the same process instances are in the same state and one of them has just been ordered, the others can be also ordered without additional calculation. However, this requires additional testing as well as data structures to keep the information about such process instances in the current state. The goal here is to find out, when (by what number of process instances in the system) it pays off to repeat the test for other process instances in the same state as for a process already ordered and when not.

\section{3.3 \textit{CPN Tools}}

\textit{CPN Tools} is a tool for editing, simulating, and analysing Coloured Petri nets. Originally developed at the University of Aarhus, Denmark, and nowadays maintained at the University of Eindhoven, The Netherlands, it is a well-known and widely used tool.

Our work (with the 3.0 version) employs only the analysis functions: entering and calculating state space, and several ML commands to accommodate nodes details' format for easier further processing and to display all state space nodes.

\section{3.4 BA Versions Comparison Framework}

We compare the versions of the BA from two perspectives (see the comparison framework at the fig. 2):

A) Effectiveness - what is the number of states they admit compared to all safe states,

B) Efficiency - what is the time needed for calculation of admitted states

The effectiveness comparison procedure A requires 6 steps:
A1) A RAS model is created in Petri net. It can be done in the specialized editor EBZ or by editing a text file.
A2) The created net in the PN or PN2 format is opened in the CoBA application and exported to the CPN format for CPN Tools.
A3) The exported CPN net is opened in CPN Tools, where its state space (reachability graph) is calculated.
A4) From the calculated state space, a chosen set of states (usually all states) is drawn and the CPN file is saved.
A5) Back in the CoBA application, the calculated states are imported from CPN file and (optionally) saved to a file of the PN2 format.
A6) Finally, the analysis of states according to user’s needs (typically one specific state or all states, sometimes a subset of all sets) is performed in CoBA.

The efficiency comparison procedure B comprises only 2 steps:
B1) A RAS model is created in Petri net (specialized editor EBZ or text editor)
B2) The RAS model development is simulated with chosen parameters in CoBA.

4 Example

Doing the BA tests for all states of the RAS model depicted at the fig. 1, i.e. a RAS with one non-sequential process and 3 types of resources with 3, 2 and 3 units respectively, we get results displayed in the table 1. It is an extract of results for the first 2 parameters constant: the RemainNeed set as Static data about one variant and Default variant strategy set as First variant.
Table 1. Results of BA tests for state space composition of the illustration RAS model.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Number of states</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Proc. choice order</td>
<td>Trans. choice by adv.</td>
<td>V0</td>
<td>V1</td>
<td>V2</td>
<td>All</td>
</tr>
<tr>
<td>First process</td>
<td>Only first</td>
<td>26</td>
<td>22</td>
<td>68</td>
<td>12</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>Only last</td>
<td>20</td>
<td>14</td>
<td>56</td>
<td>12</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>All</td>
<td>30</td>
<td>18</td>
<td>68</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>Last process</td>
<td>Only first</td>
<td>26</td>
<td>22</td>
<td>68</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Only last</td>
<td>20</td>
<td>18</td>
<td>56</td>
<td>12</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>All</td>
<td>26</td>
<td>22</td>
<td>68</td>
<td>12</td>
<td></td>
</tr>
</tbody>
</table>

The state space contains 92 states (the last column of the table), out of which 12 are deadlock states (the DDL column). Number of ordered and non-ordered states in the remaining columns varies based on chosen parameters. While 20 states are always $V_2$-ordered (i.e. these and only these states are admitted by $BA - A$, the results column 1), the varying numbers in the remaining results columns are caused by deciding about some states, whether they are $V_1$- or $V_2$-ordered. That depends on the parameters combination: process choice order and transitions choice by advancing.

For instance, in the system state No. 5, $Process_1$ is in a state represented by tokens in places $P_2$ and $P_5$, owning 1 resource unit of $R_1$, 1 or $R_2$ and 2 units of $R_3$ and needing 2 units of $R_2$ for completion, while $Process_2$ is inactive. The $Process_1$ cannot be completed without advancing, so the $BA - A$ would fail here. It has, however, two transitions available: $T_4$ and $T_6$. By advancing the process with two steps $T_4$ and $T_7$, an ordered state is reached, therefore the analysed state is $V_1$-ordered. By choosing $T_6$ to advance, the process gets into an unordered state. So, choosing the transition to advance forward in the parameter Transition choice for advancing, the category of the state is determined.

5 Conclusion

In this paper, we introduced a comparison framework for versions of the banker’s algorithm (BA), differing in complexity and effectiveness of admitting states. It uses 3 applications: EBZ for editing models of resource allocation systems (RAS), CPN Tools for calculating state spaces of RAS models, essentially needed for algorithm tests, and CoBA for performing the actual analysis of BA versions on the RAS models and interface to the other two applications.

The outlined framework is used to carry out our current research that should answer several questions regarding the pay off between complexity appearing in RAS models and algorithm versions.
Acknowledgement

This work has been supported by the research grants VEGA 1/0296/12 “Public Service Systems with Fair Access to Service” and APVV-0760-11 “Designing of Fair Service Systems on Transportation Networks”. We would also like to thank to „Centre of excellence for systems and services of intelligent transport“ (ITMS 26220120028) for building up the infrastructure that has been used.

References