CoRA: An Online Intelligent Tutoring System to Practice Coverability Graph Construction

Jan Martijn E. M. van der Werf(✉) and Lucas Steehouwer

Department of Information and Computing Science,
Utrecht University, Utrecht, The Netherlands
j.m.e.m.vanderwerf@uu.nl, lucas@architecturemining.org

Abstract. While teaching Petri nets, many students face difficulties in constructing coverability graphs from Petri nets. Providing students with individual feedback becomes infeasible in large classes.

In this paper, we present CoRA: the Coverability and Reachability graph Assistant. It is an online intelligent tutoring system designed to support users in constructing a coverability graph for a Petri net. Its main goal is to provide additional tutorial support to students, so they can practice on their own and ask questions to staff when required. CoRA is capable of giving personalized feedback; whenever a user submits a solution CoRA provides targeted feedback stating what is correct in what is not. CoRA's feedback is designed to be both guiding and informational; a user should be able to understand what went wrong and how they can improve their graph.

Keywords: Petri nets · Coverability graph · Education · Intelligent tutoring

1 Introduction

Since several years, the course Information Systems is taught as a mandatory course to first year Bachelor students Information Sciences. The course serves as an introduction in process modeling and analysis. For the theoretical aspects, the book of Van der Aalst & Stahl (2011) is used [1]. Many properties of Petri nets are explored during the course, and students are required to make statements about the nets they produce for assignments. One of the skills the students have to learn is to construct a coverability graph from a given Petri net [11]. Over the years, we observed that many students struggle with this topic and require additional guidance. As each year the course has over 180 participants, providing sufficient individual guidance and feedback becomes unfeasible. Therefore, we searched for a more creative solution, allowing students to practice converting Petri nets to coverability graphs in their own time, at their own pace, while still giving adequate feedback to support the learning process of the student.

S. Donatelli and S. Haar (Eds.): PETRI NETS 2019, LNCS 11522, pp. 91–100, 2019.
https://doi.org/10.1007/978-3-030-21571-2_6

Software-based solutions for these kinds of problems come in the form of Intelligent Tutoring Systems (ITS) [12]. These are systems which provide feedback, guidance, and other supporting factors one would find in a typical educational environment focusing on a specific topic [12]. Over the years many ITSs have been built. For example, [5] reports on the tool LISPITS, an ITS which was used to teach students the contents of a LISP course at Carnegie Melon University. In their experiment, students using LISPITS and the students receiving human tutoring worked through the material much faster than the students learning on their own. LISPITS students were slower than the students tutored by lecturers, but not by much: 11.4 hours versus 15 hours, whereas the third group took 26.5 h on average to go through the material [2]. Hence, an implemented ITS can provide sufficient tutorial support to students.

In this paper, we present CoRA: the Coverability and Reachability graph Assistant. CoRA is an ITS that focuses on providing tutorial support on how to convert a Petri net to a coverability graph. Software to automatically infer coverability graphs for a given Petri net already exist for many years. For example, the Low Level Analyzer (LoLA) [15] supports coverability analysis, including various reduction techniques. Similarly, the open-source framework ProM [16] has plugins that create a coverability graph inference. However, these tools focus on automatic model checking, rather than on providing feedback for learning coverability graph construction.

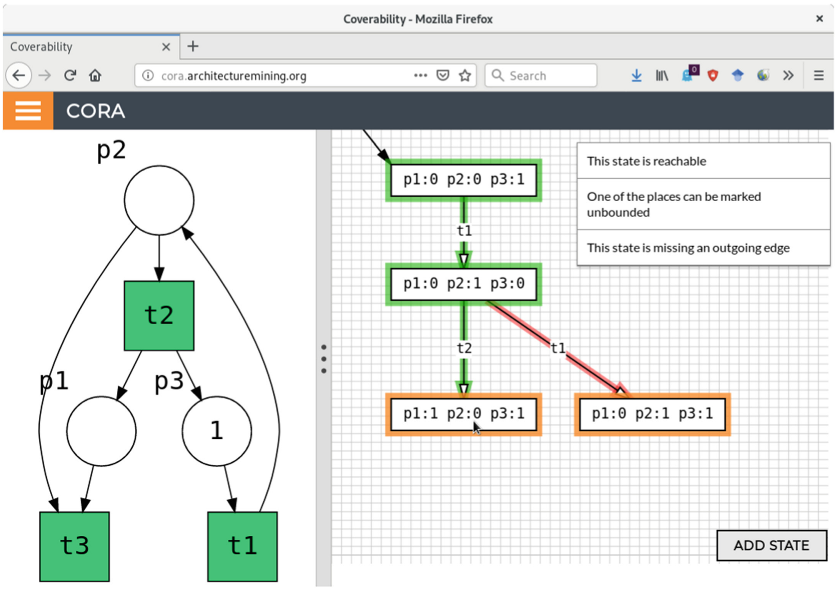
The remainder of this paper is structured as follows. In the next section, we introduce the tool by presenting a typical use case showing most of the steps a student would go through when practicing coverability graphs. In Sect. 3, we present the principles behind CoRA: coverability-graph validation and feedback generation. The tool has been tested and evaluated by a selected group of students, on which we report in Sect. 4. Last, Sect. 5 concludes the paper discussing future work.

2 CoRA to Assist Students

Over the years that the course Information Systems has been taught, we observed that many students had problems with constructing coverability graphs. For example, students often find it difficult to traverse over the state space of the Petri net in a structured manner. As a consequence, students overlook many states. Another example is that many students simply do not know when to introduce the symbol ω. CoRA supports students by providing them feedback during the construction process. It is a web-based tool and runs in most browsers[1].

Upon entering the website, the user registers with a new user name. Next, the user needs to upload a Petri net in LoLA format [15]. We deliberately chose to not include a Petri net editor, but instead rely on existing tools, such as Yasper [9], and WoPeD [8]. After a short introduction to the main interactions with the tool, the coverability construction phase starts.

[1] We tested the tool in Google Chrome and Mozilla Firefox.



This state is reachable

One of the places can be marked unbounded

This state is missing an outgoing edge

Fig. 1. Screenshot of CoRA, showing a partial coverability graph with feedback.

After uploading the file, a workspace as presented in Fig. 1 is shown. The uploaded Petri net is shown on the left. The grid canvas on the right is where the student has to construct the coverability graph. Initially, the canvas is empty, and the student can start adding states. By double clicking on a state, the student can update the state, by setting the token values for the different places. Similarly, arcs can be labeled with an action by double clicking on it, and selecting a transition from a drop-down list.

Constructing the coverability graph can be done in any order, contrary to feedback generation, which requires an initial state. Therefore, as long as no state has been marked as the initial, the only feedback the tool gives is that it does not know the initial state. Once the student sets the initial state, the tool starts providing feedback.

Colors are used to indicate whether a state or arc is correct (green), partially correct (orange), or incorrect (red). Upon hovering on an element, the tool displays the generated feedback as a list. For example, in Fig. 1, the first two states are green, indicating that these are correct, i.e., that they are reachable, have the correct token values, and the correct outgoing transitions. The last two states are orange, showing that the state is only partially correct. In this example, The student has highlighted state $[p1 : 1, p2 : 0, p3 : 1]$. As this state covers the initial marking, the tool hints that one of the places can be marked as unbounded. Additionally, outgoing transitions are missing. Although missing edges gives an incorrect result, we color the state as partially correct, to provide better feedback. Transition $t1$ from this state is colored red, hinting that the transition is not enabled in this state.

After each action by the student, the tool validates the model and generates new feedback. This process continues until the student has a correct coverability graph, i.e., that all states and transitions are colored green. By default, the tool provides immediate feedback. This is typically preferred for the target audience: novice learners [12]. Students can also choose to set feedback to manual. They then can decide themselves when to submit the graph, and when to receive feedback.

3 Design of CoRA

CoRA is a web-based tool, consisting of a client for modeling, and a server for validation and feedback generation. CoRA-client provides a coverability-graph modeling environment which uses the API exposed by CoRA-server. CoRA-client runs in the browser as a combination of HTML, JavaScript and CSS. Its JavaScript library is written in Typescript, a programming language with static-typing, which can be compiled to JavaScript. If immediate feedback is switched on, CoRA-client sends the complete coverability graph to CoRA-server after each user action. The server then validates it, and returns the generated feedback.

CoRA-Server is built on the Slim-framework[2], a bare-bones PHP framework for creating web applications, and implements a REST-service [7]. CoRA-Server runs on any server with support for PHP7. It uses a MySQL database for storage.

The tool is publicly available at http://CoRA.architecturemining.org/, its source code can be found at GitHub[3].

3.1 Analyzing Coverability Graphs

In theory, composing a coverability graph for a given Petri net is nothing more than following Algorithm 1. To find all paths from the initial marking to the newly added marking we use a graph traversal algorithm like Breadth First Search [6]. From a tutorial perspective, we have an initial state (an empty graph) and a goal state (a correct coverability graph). The student "only" needs to learn the correct strategy to arrive at a goal state from the initial state [12]. However, a coverability graph is not unique for a Petri net. Consequently, the traversal and discovery of nodes in the graph may happen in any order. This needs to be taken into account when generating automatic feedback.

To show that the order of traversal determines the resulting coverability graph, consider the example depicted in Fig. 2, taken from [1]. Starting in initial marking $[p_1]$, two transitions are enabled, t_1 and t_2. Firing t_1 results in $[p_2]$, transition t_1 in $[p_3]$. From $[p_3]$, firing transition t_4 brings us back to $[p_2]$. Now, we fire transition t_3, resulting in marking $[p_2, p_3]$. This marking clearly covers the earlier visited marking $[p_2]$, showing that place p_3 is unbounded. Similarly, this marking covers the already visited marking $[p_3]$ as well. This results in the left

[2] https://www.slimframework.com.
[3] https://github.com/ArchitectureMining/CoRA.

Algorithm 1. Generating a coverability graph

```
1: procedure GenerateCoverabilityGraph($(P, T, F), m_0$)
2:     $s_0 \leftarrow m_0$ ; $V \leftarrow \emptyset$ ; $E \leftarrow \emptyset$
3:     $O \leftarrow \emptyset$        ▷ Frontier: Set of markings which still have to be expanded
4:     $Q \leftarrow$ Queue of markings        ▷ Frontier provides $O(1)$ lookup of markings
5:     $Q$.Enqueue($s_0$)
6:     while $Q$ not empty do
7:         $s \leftarrow Q$.Dequeue() ; $O \leftarrow O \setminus \{ s \}$ ; $V \leftarrow V \cup \{ s \}$
8:         $R \leftarrow$ All paths from $s_0$ to $s$
9:         for all $r \in R$ do
10:            for all $m \in r$ do
11:                if $s \geq m$ then
12:                    for all $p \in P$ do
13:                        if $s(p) > m(p)$ then $s(p) = \omega$
14:        for all $t \in T$ do
15:            if $t$ is enabled for $s$ then
16:                $s' \leftarrow \forall p \in P : s(p) - F(p, t) + F(t, p)$
17:                if $s' \notin V \wedge s' \notin O$ then    ▷ Add the state to the Queue and Frontier
18:                    $O \leftarrow O \cup \{ s' \}$ ; $Q$.Enqueue($s'$)
19:                $E \leftarrow E \cup \{ (s, s', t) \}$        ▷ Add the discovered edge
20:    return $G = (V, E, s_0)$
```

coverability graph depicted in Fig. 3 would we have followed a different strategy in generating a coverability graph, e.g. by first continuing with transition t_1, we would not yet have discovered the unboundedness of place p_2, resulting in the middle coverability graph of Fig. 3.

Another challenge for feedback generation is that marking a place as unbounded does not have to happen immediately. Students may discover only later in the process that a place could already be marked unbounded, and unfold the graph unnecessarily deep, as shown in the third example of Fig. 3. Note that the third example in Fig. 3 cannot be produced by Algorithm 1. Still, the delivered coverability graph is correct. When a student omits to mark a place as unbounded while it is possible to do so, then the feedback generation algorithm needs to adapt to this. It needs some way of "remembering" that this place can be marked as unbounded and that all markings in the postset of the current marking can also mark this place as such.

3.2 Providing Feedback

The main goal of CoRA is providing useful feedback on coverability graphs. Designing good feedback however, is not an easy task. Feedback can inform the user about his or her progress and can also guide the user to the correct solution, for example by giving pointers [10]. These two types of feedback, informational and guiding feedback, can overlap. The goal of CoRA is providing both these forms of feedback; it should provide information to the users about which elements of the coverability graph are correct and incorrect, but CoRA should also

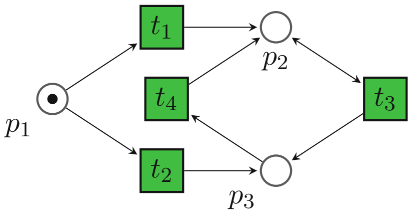


Fig. 2. Petri net example taken from [1]

$$\begin{tikzcd}
& {} \arrow[d] & \\
& {[\,p_1\,]} \arrow[dl, "t_1"'] \arrow[dr, "t_2"] & \\
{[\,p_2\,]} \arrow[d, "t_3"] & & {[\,p_3\,]} \arrow[ll, "t_4"'] \\
{[\,\omega \cdot p_2, \omega \cdot p_3\,]} \arrow[loop below, "{t_3, t_4}"] & &
\end{tikzcd}$$

$$\begin{tikzcd}
& {} \arrow[d] & \\
& {[\,p_1\,]} \arrow[dl, "t_1"'] \arrow[dr, "t_2"] & \\
{[\,p_2\,]} \arrow[d, "t_3"] & & {[\,p_3\,]} \arrow[ll, "t_4"'] \\
{[\,p_2, \omega \cdot p_3\,]} \arrow[loop right, "t_3"] \arrow[d, "t_4"] & & \\
{[\,\omega \cdot p_2, \omega \cdot p_3\,]} \arrow[loop below, "{t_3, t_4}"] & &
\end{tikzcd}$$

$$\begin{tikzcd}
& {} \arrow[d] & \\
& {[\,p_1\,]} \arrow[dl, "t_1"'] \arrow[dr, "t_2"] & \\
{[\,p_2\,]} \arrow[d, "t_3"'] & & {[\,p_3\,]} \arrow[ll, "t_4"'] \\
{[\,p_2, p_3\,]} \arrow[rr, "t_4"] \arrow[d, "t_3"'] & & {[\,\omega \cdot p_2\,]} \arrow[d, "t_3"'] \\
{[\,p_2, \omega \cdot p_3\,]} \arrow[loop below, "t_3"] \arrow[rr, "t_4"] & & {[\,\omega \cdot p_2, \omega \cdot p_3\,]} \arrow[loop below, "{t_3, t_4}"]
\end{tikzcd}$$

Fig. 3. Three correct coverability graphs of the Petri net of Fig. 2, the first and second followed a different strategy in traversing the states. In the third, unbounded places were discovered at a later stage.

provide messages which suggest a corrective measure if required. For example, if a state misses an outgoing edge, a message should be given stating that an edge is missing. This message shows that something is not right (an edge is missing), but also provides a hint on how this problem should be solved (adding an edge).

There are three kinds of feedback messages that we can give with CoRA. For each element of the graph we can decide whether it is a correct, partially correct, or incorrect element. For example, the user can model two reachable states, but the label of the edge between those states is not correct. CoRA gives feedback that this edge represents the wrong transition.

CoRA can also provide warnings. Warnings are intended for elements that are not incorrect, but do provide a risk to make mistakes later on. For example, students can make coverability graphs with duplicate states. Having duplicate states in a coverability graph does not make the graph incorrect, but it does make it unnecessarily difficult for the student to not make mistakes.

Another example is the introduction on unboundedness of places. Instead of following Algorithm 1, CoRA checks for each state created by the user if this state covers some state on the path from the initial state to the state under analysis. If it is indeed a cover, but no ω has been introduced, the user receives feedback that it is possible to introduce an ω in this state.

Next to the kinds of feedback, there is a difference on when feedback should be provided: for beginners it is preferred that feedback is given instantly, whereas for people with some experience it is better to provide delayed feedback [12]. CoRA supports both forms. At first, most of the students will not be familiar with composing coverability graphs and will therefore need immediate guidance, whereas at later stages it can be more beneficial for students to only get feedback when they feel like they need it or when they are finished.

With this knowledge in mind, we designed a set of messages to be shown to the user. Messages can be assigned to particular elements of the graph, as well as to the graph as a whole. Table 1 shows the feedback messages that we designed for CoRA. All feedback messages have been constructed to be both informational as well as guiding. However, further research is required to validate these messages.

4 Initial Evaluation

CoRA was presented and pitched during the Information Systems course. This provided an opportunity to test CoRA immediately with its target audience: students. CoRA was introduced during one of the lectures and a small demo was given on how to use it. After this demo a set of Petri nets was given to the students to practice with. Students were also asked to fill in a questionnaire after practicing a few times. The questionnaire was presented in an on-line form, and was available at all times. This questionnaire contained the System Usability Scale (SUS) [4]. The SUS provides a simple way to measure the usability of a system. The Net Promoter Score (NPS) [13] was included as it is correlated to the SUS ($r = 0.61$, [14]), and thus provides an extra validation. The recommendations section of the questionnaire had questions on how to improve CoRA, as well as a question whether they would recommend CoRA to fellow students.

Only ten responses to the survey were recorded, probably since the tool was presented only in one of the last lectures before the exam, Eight participants were following the Information Systems course while taking the survey. These eight participants did state that they found CoRA to be at least somewhat supportive in their efforts to understand the subject of coverability graphs. With a score of 3.25 out of 5 for this question the response was quite neutral, with a slight favor towards being helpful. For the System Usability Scale scores ranged between 62.5 and 95, with an average score of 70.25, and a median of 76.25. According to [3] this is often an acceptable score, but there certainly is room for improvement. The scores for the NPS seem to indicate this as well. NPS scores range between -100% and 100%. With a score of 20% we have more promoters than detractors, but clearly the score could be much better.

Eight of the participants provided suggestions on improving CoRA. The main complaint was that it was not possible to upload a second Petri net, to keep practicing. Instead, they needed to restart CoRA-client each time they wanted to start a new session, which also means going through the hoops of registering

Table 1. Types of feedback and hints the tool provides

Type of feedback	Hint provided to user
Initial state	
No initial state	No state has been declared as the initial state
Incorrect initial state	The initial state of the graph is not equal to the initial marking of the Petri net
Correct initial state	The initial state of the graph and the initial marking of the Petri net are identical
Unbounded place	At least one of the places in the initial state is marked as unbounded. This is impossible!
States	
Reachable from preset	This state is reachable from its preset
Duplicate state	This state occurs multiple times in the graph
Omitted ω	It is possible to introduce an ω in this state, but this has not been done
Not reachable from preset	This state is not reachable from at least one of the states in its preset
Edge missing	This state is missing an outgoing edge
Not reachable from initial state	This state is not reachable from the initial state
Omitted ω from preset	At least one of the states in the preset has at least one place marked as unbounded, while this state does not mark this place as such
Edges	
Enabled and correct post	The edge's label corresponds to an enabled transition and points to a reachable state
Duplicate edge	There are multiple edges with the same label originating from the same state
Enabled, correct post state, wrong label	The transition corresponding to the label is enabled and the target of the label is reachable, but firing this transition does not lead to this state
Enabled, incorrect post state	The transition corresponding to the label is enabled, but the target state is not reachable from the source state
Disabled	The transition corresponding to the label is disabled
Disabled, correct post state	The transition corresponding to the label is disabled, but the target state is reachable by firing a different transition

a new account again. Participants also indicated that there were a few bugs with the modeler, especially regarding opening menu's for editing elements of the graph.

5 Conclusions

In this paper, we presented CoRA: the Coverability and Reachability graph Assistant. It is an online intelligent tutoring system to practice constructing coverability graphs, and provides feedback based on the users' input.

Initial evaluation shows that the tool has the desired potential, but requires some improvements. In the near future, we plan to extend the tool with a framework to support multiple exercises per session, and to add PNML support.

Future work lies in assessing the quality of the feedback messages the tool provides. The tool stores all intermediate stages of the coverability graph, allowing to study which feedback is given, and how this feedback is perceived by students. Another direction of research lies in studying the possibilities in the realm of gamification and applied games. Games provide a continuous loop of feedback messages [10]. CoRA only provides feedback when the client requests it. With a game, feedback could be provided constantly, possibly providing better support to the user. Currently, CoRA just assumes the user is a novice. A possible avenue of further research, perhaps in combination with the idea of gamification, would be to automatically infer the experience level of a user and adapt the provided feedback based on this level.

CoRA gives feedback on the conversion of Petri nets to coverability graphs. There are many other analysis techniques students need to learn. Future work therefore also includes the development of tutoring systems for other techniques, such as invariants. These solutions could then be integrated into a electronic learning environment, providing the students an environment where they can practice all sorts of techniques regarding Petri nets, including coverability analysis with CoRA.

References

1. van der Aalst, W.M.P., Stahl, C.: Modeling Business Processes–A Petri Net-Oriented Approach. Cooperative Information Systems Series. MIT Press (2011)
2. Anderson, J.R., Boyle, C.F., Reiser, B.J.: Intelligent tutoring systems. Science **228**(4698), 456–462 (1985)
3. Bangor, A., Kortum, P.T., Miller, J.T.: An empirical evaluation of the system usability scale. Int. J. Hum.-Comput. Interact. **24**(6), 574–594 (2008)
4. Brooke, J., et al.: SUS-a quick and dirty usability scale. In: Usability Evaluation in Industry, vol. 189, no. 194, pp. 4–7 (1996)
5. Corbett, A.T., Anderson, J.R.: Lisp intelligent tutoring system: research in skill acquisition. In: Computer-Assisted Instruction and Intelligent Tutoring Systems: Shared Goals and Complementary Approaches, pp. 73–109 (1992)
6. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms. MIT Press, Cambridge (2009)

7. Fielding, R.T., Taylor, R.N.: Architectural styles and the design of network-based software architectures, vol. 7. University of California, Irvine Doctoral dissertation (2000)
8. Freytag, T., Sänger, M.: WoPeD - an educational tool for workflow nets. In: Proceedings of the BPM Demo Sessions, CEUR Workshop Proceedings, vol. 1295, pp. 31–35 (2014). CEUR-WS.org
9. van Hee, K.M., Oanea, O., Post, R.D.J., Somers, L.J., van der Werf, J.M.E.M.: Yasper: a tool for workflow modeling and analysis. In: ACSD, pp. 279–282 (2006)
10. Kapp, K.M.: The Gamification of Learning and Instruction: Game-based Methods and Strategies for Training and Education. Wiley (2012)
11. Karp, R.M., Miller, R.E.: Parallel program schemata. J. Comput. Syst. Sci. **3**(2), 147–195 (1969)
12. Passier, H.J.M.: Aspects of Feedback in Intelligent Tutoring Systems for Modeling Education. PhD thesis, Open University, The Netherlands (2013)
13. Reichheld, F.F.: The one number you need to grow. Harvard Bus. Rev. **81**(12), 46–55 (2003)
14. Sauro, J.: Does better usability increase customer loyalty? (2010). https://measuringu.com/usability-loyalty/. Accessed 05 July 2018
15. Schmidt, K.: LoLA a low level analyser. In: Nielsen, M., Simpson, D. (eds.) ICATPN 2000. LNCS, vol. 1825, pp. 465–474. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-44988-4_27
16. Verbeek, H.M.W., Buijs, J.C.A.M., van Dongen, B.F., van der Aalst, W.M.P.: XES, XESame, and ProM 6. In: Soffer, P., Proper, E. (eds.) CAiSE Forum 2010. LNBIP, vol. 72, pp. 60–75. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-17722-4_5