A Data-Centric Approach to Deadlock Elimination in Business Processes

Christoph Wagner

Institut für Informatik, Humboldt Universität zu Berlin,
Unter den Linden 6, 10099 Berlin, Germany
cwagner@informatik.hu-berlin.de

Abstract. In this paper, we sketch a data-centric approach to avoid deadlocks of a business process. If dependencies between data values are neglected or modelled incorrectly, this can lead to errors in the control flow of the business process. We address the problem of detecting deadlocks which are caused by the improper handling of data. We show by example, how these deadlocks can be detected by means of a symbolic reachability graph. Under certain conditions, we can derive the correct dependency between the involved data values. This allows to modify the business process in a way so that the detected deadlocks will not be reachable.

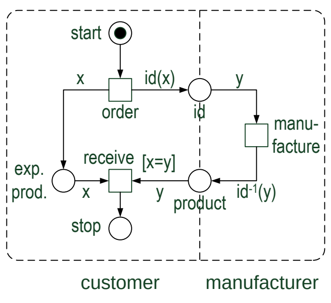
1 Background

The design of business processes is an error prone task. This motivates the use of formal verification to help a business process designer to avoid certain kinds of errors. Models of business processes can be transformed into more formal models like process algebra [4] or Petri nets [9,7]. A business process can also be designed as a Petri net directly (e. g. with CPN Tools [10]). These models can be checked for soundness and other properties by means of formal verification.

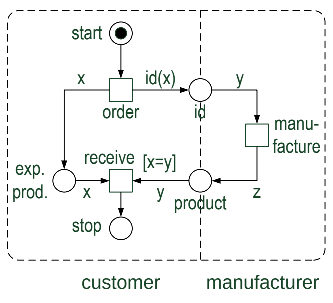
However, little attention has been paid to the influence of *data* on the correctness of a business process. Most formal models represent data only in a highly abstracted and imprecise form. Models that explicitly include data usually have a clear separation between the control flow part and the data part [3]. Often, the dependencies between data flow and control flow are not very complex, i. e. there is only a small set of values a data item can have. For many the properties in focus of recent research, the actual value of a data item is not important [12], [13] and primarily concerns the order in which read and write activities on variables are carried out. E. g., reading an uninitialized value is considered an error.

In this paper, we consider processes that are heavily influenced by data and might be unable to finish a task for some combinations of data values. Technically, this means that a *deadlock* is reachable under some conditions. Our goal is to find out those harmful combinations of data values and describe the relation the value must adhere to in order to avoid a deadlock (e. g. in the form of a function) and use this information to fix the process. Concerning this aspect, our approach is more general than the approach of [1] which does not deal with relations between data values.

We represent a business process by a *High-Level Petri net* [5]. A High-Level Petri net is an extension of a Petri net where places are typed, tokens have values of the respective type and arcs are inscribed with terms. When a transition fires, values are assigned to the variables appearing in the inscriptions of adjacent arcs. The evaluation of the inscriptions determines which values are produced and consumed. The terms are evaluated with a fixed interpretation (note that the Petri net is not a *schema* in the sense of [11]). We do not exploit restrictions on the Petri net's structure that a certain business process modelling language might impose. This allows us to handle more general models (e. g. as obtained from CPN Tools). We assume that the Petri net is bounded, acyclic and fulfils some technical requirements of minor importance which do not restrict the expressivity and will not be mentioned here. The restriction to acyclic nets can be relaxed as long as computational issues are neglected. We assume that the set of values used by the Petri net can be so large that an explicit enumeration would be computationally inefficient.



(a) Correct business process



(b) Incorrect business process

Fig. 1: Business process consisting of a customer an a manufacturer, represented as a Petri net

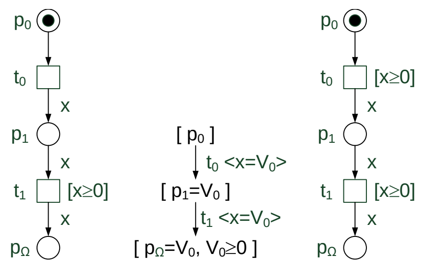
Consider a business process formed by a customer and a manufacturer (Fig. 1a). The customer orders a product x from the manufacturer by telling the manufacturer the product's id (**order**). The manufacturer assembles the product associated with the id and returns the product to the customer (**manufacture**). Let us now assume that due to a design error in the manufacturer's internal workflow, the id y obtained from the customer is lost and replaced by the id of an arbitrary product z (Fig. 1b). Then, the customer will get a product different from the one he expects. In that case, no further action (**receive**) can be performed because the condition $x = y$ is not satisfied and we reach a *deadlock*. We call this a *conditional deadlock*, because it occurs only if the compared data values are not equal. Note that in a more complex scenario, a choice dependent on data may not always lead to a deadlock instantly but later on in the process. In that case, the deadlock condition has to be propagated backwards. This aspect will not be covered in this paper.

We introduce an approach to *detect* conditional deadlocks and to *derive* the dependency between values that must be used in order to *avoid* the deadlock. In Sect. 2, we illustrate by simple examples, how to identify deadlocks by means of a *symbolic reachability graph.* Section 3 shows by a more sophisticated example, how to derive the precise conditions under which an conditional deadlock can be avoided. It is not always possible to derive these conditions precisely. Section 4 shows an example that can not be corrected with our approach due to imprecise results. In Sect.5, we conclude our work.

2 Basic Idea

In this section, we show how to detect a deadlock by means of the *symbolic reachability graph* (SRG) of the Petri net. In the following examples, we assume that every place has the type integer except for some places that carry tokens (denoted as black dots) that do not have a value. A marking m of a Petri net N is considered a *deadlock*, if no transition is enabled and m is not contained in a set Ω of *final markings* of N.

In each of the following examples, we assume that the net is in a final marking exactly when the place p_Ω is marked. Consider the net N_1 in Fig. 2a. We can easily see that N_1 is not deadlock free and $M = \{[p_1 = n] | n \in \mathbb{Z}, n < 0\}$ is the set of deadlocks reachable in N_1. Obviously, by adding the guard $x \geq 0$ to t_0, we can ensure that t_1 will always be enabled and N_1 will eventually reach the final marking $[p_\Omega]$ (Fig. 2c).

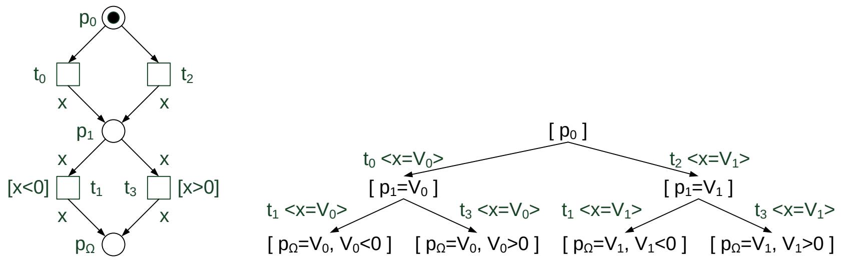


(a) N_1 (b) $SRG(N_1)$ (c) N_1'

Fig. 2: Net with a conditional deadlock and its correction

In a marking of the symbolic reachability graph, every value is represented by a term. Without going into technical details, we show how to construct the symbolic reachability graph of N_1. Starting from initial marking $[p_0]$, t_1 produces an arbitrary integer on p_1 (Fig. 2b). We represent this integer by a unique identifier V_0. Thus we get the marking $[p_1 = V_0]$. While formally V_0 is a constant, we treat V_0 as a variable: V_0 may later be *instantiated* by any value from $\mathbb{Z}$. Since t_1 is enabled only if V_0 is non-negative, we keep the condition in $V_0 \geq 0$ in the successor marking $[p_2 = V_0, V_0 \geq 0]$ of $[p_1 = V_0]$ (we obtain the condition by combining the firing mode of t_1 and the guard of t_1). We consider an instance of a marking of the symbolic reachability graph *valid*, if every condition denoted in the marking evaluates to *true*. Obviously, a marking of N_1 is reachable exactly if it is a valid instance of a marking of the symbolic reachability graph.

With the symbolic reachability graph, we can identify under which condition a marking instantiates to a deadlock. Here, $[p_1 = V_0]$ is a *conditional deadlock* for $\neg V_0 \geq 0$ since each instance of $[p_1 = V_0]$ has no successor for this condition. We now *enforce* that the condition $V_0 \geq 0$ holds in $[p_1 = V_0]$ by adding the guard $x \geq 0$ to the predecessor-transition t_0 of $[p_1 = V_0]$. We obtain a corrected version N_1' of N_1, which is deadlock free.



(a) N_2 (b) $SRG(N_2)$

Fig. 3: Net with a branching symbolic reachability graph

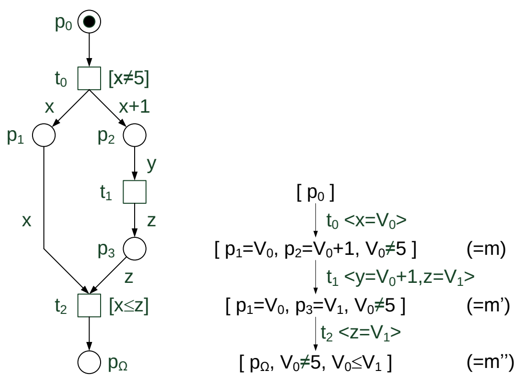
Fig. 3a shows a net N_2 which is not deadlock free and which has a branching symbolic reachability graph. Note that the values produced by t_0 and t_2 obtain different identifiers (although both branches behave symmetrically). $[p_1 = V_0]$ and $[p_1 = V_1]$ are conditional deadlocks of N_2. $[p_1 = V_0]$ is a deadlock for $\neg((V_0 < 0) \vee (V_0 > 0))$. Any successor of an instance of $[p_1 = V_0]$ belongs either to the branch with condition $V_0 < 0$ or the branch with condition $V_0 > 0$. We enforce the condition $(V_0 < 0) \vee (V_0 > 0)$ by adding the guard $(x < 0) \vee (x > 0)$ to t_0 (which is effectively equivalent to $x \neq 0$). By repeating this procedure for $[p_1 = V_1]$, we get the same guard for t_2 and obtain a deadlock free net.

3 Derivation of a deadlock-preventing guard

In the previous section, we have shown that by adding an appropriate guard, we can prevent the reachability of deadlocks. Adding a guard or replacing a guard by a more restrictive one makes the net less *permissive*, that is the set of reachable markings gets smaller. Naturally, we want to prevent all deadlocks from being reachable. On the other hand, we do not want to prevent the reachability of more markings than necessary. This section addresses the issue of deriving a least restrictive guard. How to derive a guard (like $x > 0$) from a condition (like $V_0 > 0$) is not obvious if more than one variable is involved and functions are used in the arc inscriptions.

Without loss of generality, we assume that the symbolic reachability graph is a tree (if not, we can unfold it). Note that the symbolic reachability graph usually has an acyclic structure since names of value identifiers never repeat. In the tree, we always modify the guard of the transition that directly precedes the deadlock. It should be mentioned here, that due to restrictions inherent to the modelled the business process (e. g. the dependency on of external events which can not be influenced), it might not be possible to modify that transition. In that case, we choose the first modifiable predecessor transition in the tree. As guard derivation is more involved in that case, it will not be shown here. Consider the net N_3 in Fig. 4a, which reaches a deadlock if the integer produced by t_1 on p_3 is greater than the integer produced by t_0 on p_1. An ad-hoc way to fix N_3 is to add the guard $z \geq y$ to t_1. Then, N_3 eventually reaches the final marking $[p_\Omega]$. However, the guard $z \geq y - 1$ would also ensure that N_3 reaches $[p_\Omega]$ but is less *restrictive* than $x \geq y$, since it evaluates to true for more assignments of x and y. The guard $y = 6 \vee z \geq y - 1$ is even less restrictive than $z \geq y - 1$.

We derive this guard from the symbolic reachability graph in Fig. 4b. m' is a deadlock for condition $\neg(V_0 \leq V_1)$ since every valid instance of m'' satisfies $V_0 \leq V_1$. We prevent the reachability of m' under condition $\neg(V_0 \leq V_1)$ by adding a guard to t_1. It is sufficient that the guard forbids the violation of $V_0 \leq V_1$ only for valid instances of m and m'. So we can assume that for a step $m \xrightarrow{a} m'$ (where $a = t_1\langle y = V_0 + 1, z = V_1\rangle$) with integers V_0 and V_1 given
(1) the condition $V_0 \neq 5$ already holds (precondition in m)
(2) y and z are bound to the valuations of $V_0 + 1$ and V_1 (firing mode of t_1).
This motivates the definition of the expression



(a) N_3 (b) $SRG(N_3)$

Fig. 4: A net that is less obvious to correct

$$\forall V_0, V_1 \in \mathbb{Z} : V_0 \neq 5 \wedge y = V_0 + 1 \wedge z = V_1 \implies V_0 \leq V_1$$

We call this expression *least restrictive* $V_0 \leq V_1$*-enforcing (for step* $m \xrightarrow{a} m'$*)*. Note that the more preconditions an expression has, the less restrictive it is. V_0 and V_1 are all-quantified because the condition $V_0 \leq V_1$ shall be enforced for *any* valid instance of m'. It is easy to see that this expression is indeed equivalent to $y = 6 \vee z \geq y - 1$. Since V_0, V_1 are uniquely determined by y and z, we can replace V_0 by $y - 1$ and V_1 by z, thus eliminating V_0, V_1 from the expression.

We go back to the business process introduced in Fig. 1a. The reader may believe that there is a deadlock $m' = [\text{exp. prod.} = V_0, \text{product} = V_1]$ for condition $\neg(V_0 = V_1)$ which is reachable from $m = [\text{exp. prod.} = V_0, id(V_0)]$ via transition **manufacture**. This leads to the expression $\forall V_0, V_1 \in \mathbb{Z} : y =$

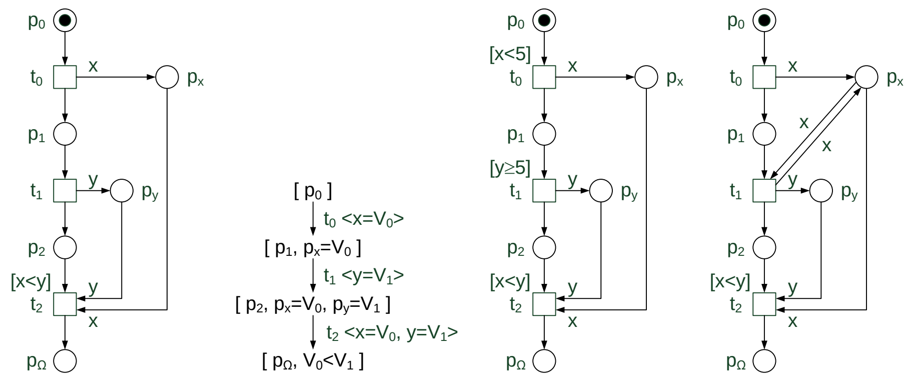
$id(V_0) \wedge z = V_1 \implies V_0 = V_1$, which is equivalent to $z = id^{-1}(y)$. Thus we have reconstructed the dependency between products and their id's and may correct the business process by replacing z by $id^{-1}(y)$.

In general, several successive deadlock elimination steps are necessary in order to obtain a deadlock free net, as every step may introduce new deadlocks. Our approach is similar to Dijkstra's method to derive the weakest precondition for which a given program terminates in a specified state [2]. However, our approach allows to derive a modification even for an intermediate step because a precondition and a postcondition are already given. Therefore, we may perform modifications in a local manner and do not have to start at the final markings. Conceptually, the approach is applicable even if no final marking is specified at all.

Note that modifying a transition may have non-local side-effects if the transition appears more than once in the tree. In that case, more markings are rendered unreachable than intended, leading to a suboptimal solution. As the next section shows that even without non-local side-effects, it is not always possible to get an optimal solution.

4 Uncorrectable net

Some nets can not be corrected using the expression derived in the last section. Consider the net N_4 in Fig. 5a. There is no *unique* way to avoid the deadlocks of



(a) N_4 (b) $SRG(N_4)$ (c) N_4' (d) N_4''

Fig. 5: A net and two possible corrections

N_4. For example, we get a deadlock free net N_4' (Fig. 5c) by adding the guard $x < 5$ to t_0 and $y \geq 5$ to t_1. However, instead of 5, we could have chosen any other integer. The symbolic reachability graph does not give us a hint on how to derive the general structure of the two guards. Here, the expression enforcing

$V_0 < V_1$ for $m \overset{t_1\langle y=V_1\rangle}{\rightarrow} m'$ (with $m = [p_2 = V_0, p_1], m' = [p_2 = V_0, p_3 = V_1]$) gives

$$\forall V_0, V_1 \in \mathbb{Z} : y = V_1 \implies V_0 < V_1$$

which evaluates to false for every $y \in \mathbb{Z}$. The expression is too restrictive due to a *lack of information.* The condition $V_0 \leq V_1$ that shall be enforced depends on both the values of V_0 and V_1, but t_1 has no access to the place p_x on which V_1 is stored. For the slightly different net N_4'' (Fig. 5d) in which t_1 has access to both values, an appropriate guard for t_1 can be derived: The expression enforcing $V_0 < V_1$ gives $\forall V_0, V_1 \in \mathbb{Z} : x = V_0 \wedge y = V_1 \implies V_0 < V_1$, which is equivalent to $x < y$.

A related phenomenon is known from controller synthesis. A controller forbids the supervised system to perform some actions in certain situations. The guards we add to a transition have an impact on the net comparable to a controller. If certain sets of states are indistinguishable for the controller, then there is no *unique* maximal permissive controller [6]. From the point of view of t_1, all markings that differ only on place p_x are indistinguishable.

5 Conclusion and Future work

The detection and correction of errors in a business process is a tedious task. Petri nets provide of formal foundation to apply formal verification on business processes. We have shown how to identify and avoid deadlocks of a High-Level Petri net by means of a symbolic reachability graph. As a byproduct, our approach allows to formulate the dependencies between data values that must hold in order to avoid a deadlock.

Our goal is to apply our approach in a distributed setting. Therefore, several problems have to be considered. Since one part of a business process usually does not have complete information about the state of the other parts, problems caused by a lack of information as described in Sect. 4 are more likely to occur. It is also less likely that a deadlock can be fixed locally. Especially in the presence of cycles, non-local side-effects occur inevitably.

In contrast to [8], which proposes a correction algorithm for services (but ignores the data issue), we can not add elements to the structure of the net. Having net N_4 from Fig. 5a in mind, this imposes a strong restriction on the applicability of our approach. We believe that our approach can still provide valuable hints for a business process designer. The designer may first design the business process model with the help of algorithms which do not take data into account but are more precise on the structure. After the general design of the business can be considered correct, our approach can be used to find small errors that occur only for very special combinations of values. In that case, the modification proposed by our approach might be precise enough to provide a reasonable correction. We also believe that our approach will produce useful results if the service that is modified has a very canonical structure (e. g., acyclic or even tree-like structure). We hope to gain valuable insights regarding the synthesis of a service that can communicate deadlock-freely with a given service.

References

1. Awad, A., Decker, G., Lohmann, N.: Diagnosing and repairing data anomalies in process models. In: Rinderle-Ma, S., Sadiq, S., Leymann, F. (eds.) Business Process Management Workshops, BPM 2009. Lecture Notes in Business Information Processing, vol. 43, pp. 5–16. Springer-Verlag (Mar 2010)
2. Dijkstra, E.W.: A Discipline of Programming. Prentice Hall, Inc. (October 1976)
3. Fan, S., Dou, W., Chen, J.: Dual Workflow Nets: Mixed Control/Data-Flow Representation for Workflow Modeling and Verification. pp. 433–444 (2007)
4. Ferrara, A.: Web services: a process algebra approach. In: Proceedings of the 2nd international conference on Service oriented computing. pp. 242–251. ICSOC '04, ACM, New York, NY, USA (2004)
5. Jensen, K.: Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use (Volume 1), EATCS Series, vol. 1. Springer Verlag (April 1992)
6. Kalyon, G., Le Gall, T., Marchand, H., Massart, T.: Control of Infinite Symbolic Transition Systems under Partial Observation. In: European Control Conference. Budapest Hungary (Aug 2009)
7. Lohmann, N.: A feature-complete Petri net semantics for WS-BPEL 2.0 and its compiler BPEL2oWFN. Informatik-Berichte 212, Humboldt-Universität zu Berlin (Aug 2007)
8. Lohmann, N.: Correcting deadlocking service choreographies using a simulation-based graph edit distance. In: BPM 2008. LNCS, Springer-Verlag (Sep 2008)
9. Lohmann, N., Verbeek, E., Dijkman, R.: Petri net transformations for business processes – A survey. In: Jensen, K., van der Aalst, W. (eds.) Transactions on Petri Nets and Other Models of Concurrency II, Lecture Notes in Computer Science, vol. 5460, pp. 46–63. Springer Berlin / Heidelberg (2009)
10. Ratzer, A.V., Wells, L., Lassen, H.M., Laursen, M., Qvortrup, J.F., Stissing, M.S., Westergaard, M., Christensen, S., Jensen, K.: CPN Tools for editing, simulating, and analysing coloured petri nets. In: van der Aalst, W.M.P., Best, E. (eds.) ICATPN. Lecture Notes in Computer Science, vol. 2679, pp. 450–462. Springer (2003)
11. Reisig, W.: On the Expressive Power of Petri Net Schemata. In: ICATPN 2005, Miami, USA. Proceedings. Lecture Notes in Computer Science, vol. 3536, pp. 349–364. Springer Verlag (May 2005)
12. Sun, S.X., Zhao, J.L., Nunamaker, J.F., Sheng, O.R.L.: Formulating the data-flow perspective for business process management. Information Systems Research 17(4), 374–391 (2006)
13. Trčka, N., Sidorova, N.: Data-flow anti-patterns: Discovering data-flow errors in workflows. In: CAiSE 2009. LNCS 5565. p. 425. Springer (2009)