Synthesis of Behavioral Controllers for DES: Increasing Efficiency

Kai Bollue * **Michaela Slaats** ** **Erika Ábrahám** ***
Wolfgang Thomas **** **Dirk Abel** †

* *Institute of Automatic Control, RWTH Aachen University, Aachen, Germany (e-mail: K.Bollue@irt.rwth-aachen.de).*
** *Chair of Computer Science 7, RWTH Aachen University, Aachen, Germany (e-mail: Slaats@automata.rwth-aachen.de)*
*** *Chair of Computer Science 2, RWTH Aachen University, Aachen, Germany, (e-mail: abraham@informatik.rwth-aachen.de)*
**** *Chair of Computer Science 7, RWTH Aachen University, Aachen, Germany (e-mail: Thomas@informatik.rwth-aachen.de)*
† *Institute of Automatic Control, RWTH Aachen University, Aachen, Germany (e-mail: D.Abel@irt.rwth-aachen.de).*

Abstract: In Bollue et al. (2009), a methodology was introduced for the synthesis of behavioral controllers for discrete-event systems. The approach is based on NCES-like Petri net models of the uncontrolled plant and additional goal and safety specifications given by linear marking constraints. This paper presents different approaches to improve the synthesis process with respect to efficiency and applicability. One focus is the use of satisfiability checking of systems of integer linear inequations in the preprocessing of the model for the elimination of unnecessary complexity during the process. Further, several improvements of the synthesis algorithm itself are discussed, which increase the efficiency by applying an advanced guided search and by reusing already found partial solutions.

Keywords: Petri nets, algorithms, synthesis, efficiency, controllers, discrete-event systems

1. INTRODUCTION

Although several approaches have been made to automatic synthesis of controllers for discrete event systems (see e.g. Pinzon et al. (1999)), the standard procedure in practice is still to do some intuitive controller design by hand and refine the controller in an iterative process of testing or simulating and manual adjustments.

Petri nets provide a powerful tool for the modeling of discrete-event systems (see Abel (1990), Orth et al. (2004)) and thus also a basis for methods for controller synthesis (Giua (1992), Moody and Antsaklis (2000)).

In order to achieve a better practical applicability, Petri nets can be augmented by elements from condition/event systems (Sreenivas and Krogh (1991), Hanisch et al. (1998), Thieme and Lueder (1999)) to allow for a clear and modular modeling technique.

The methodology presented in Bollue et al. (2009) is based on NCES-like Petri nets for the modeling of the uncontrolled plant, linear marking constraints for the expression of safety and goal constraints and an recursive synthesis algorithm, which derives possible controllers, that drive the plant from the current state to a state fulfilling the goal specifications while obeying the safety specifications in the process. One remaining problem is the potentially very high complexity of the algorithm, depending on the structure of the plant model. This paper presents several approaches to reduce the complexity even for not well suited plant models and the application to an example, which challenges the algorithm's efficiency very much and thus shows the effect of the improvements quite well.
The rest of the paper is organized as follows: Section 2 gives a recapitulation of the used modeling technique and the synthesis algorithm introduced in Bollue et al. (2009). Section 3 describes our optimizations of the preprocessing of the model using satisfiability checking. Section 4 presents structural improvements of the algorithm itself. After showing some results in section 5, we conclude with an outlook in Section 6.

⋆ Research supported by DFG Research Training Group AlgoSyn ("Algorithmic Synthesis of Reactive and Discrete-Continuous Systems").

2. THE SYNTHESIS ALGORITHM

2.1 Modeling Technique

To allow for a clear and modular modeling, the theory of classical Petri nets is augmented by additional arc types (see below). As also event arcs are used, the modeling technique resembles net condition event systems (NCES) as described in Hanisch et al. (1998), though several interpretations are still different.
Such a net consists of a set $P = \{p_1, \ldots, p_n\}$ of *places*, a set $T = \{t_1, \ldots, t_m\}$ of *transitions*, and a set of *arcs* $A \subseteq (P \times T) \cup (T \times P) \cup (T \times T)$. As can be seen

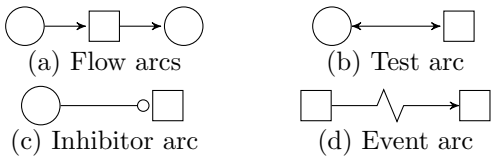

Fig. 1. Arc types used for plant modeling.

from the latter definition, arcs can lead from places to transitions and vice versa (flow arcs, inhibitor arcs, test arcs) or from transitions to transitions (event arcs). Each place can have some *tokens* placed on it, whereas the number of tokens on a place can be bounded from above by the place's *capacity*. To avoid decidability problems in connection with inhibitor arcs and due to the small relevance of unbounded Petri nets for the modeling of real plants, we only consider nets in which all places have a finite capacity, given by a function $cap : P \to \mathbb{N}$. A *marking* is a function $m : P \to \mathbb{N}$ which assigns to each place the number of tokens placed on it.

Changes in the marking of a net are caused by the *firing* of transitions. The condition and effect of firing a transition is specified by the arcs connected to it. The arcs have a weight given by $wgt : A \to \mathbb{N}$ and can be of four different types (see also Fig. 1):

- A *flow arc* a from a place p to a transition t takes $wgt(a)$ token(s) from p when t fires. If $m(p) < wgt(a)$, then the arc blocks the transition (i.e. prevents it from firing).
- A *flow arc* a from a transition t to a place p delivers $wgt(a)$ token to p if thereby $cap(p)$ is not exceeded, and blocks the firing of the transition otherwise.
- A *test arc* a between a place p and a transition t allows t to fire only if there are at least $wgt(a)$ tokens on p.
- An *inhibitor* arc a from place p to transition t blocks t, if there are at least $wgt(a)$ tokens on p.
- An *event arc* a from a transition t_1 to another transition t_2 synchronizes t_1 and t_2 as follows: t_2 can fire only together with t_1. Furthermore, if t_1 fires and t_2 is enabled then t_2 fires synchronously with t_1 (otherwise t_1 fires, but t_2 does not). To ease the handling of event arcs in the conversion and use of the model, we introduce some restrictions on the use of event arcs:
  - Event arcs must not form a loop, i.e. a transition must not be transitively reachable from itself by following event arcs only.
  - For each transition there may be at most one event arc ending at it.
  - There must not be any conflicts among transitions belonging to the same group of transitions transitively connected by event arcs. This means that the firing of one transition of such a group must not disable another transition of the same group, which was previously enabled.

To model a plant and especially the interaction of a controller with the plant, we specify for each transition, whether it is *controllable* or not (i.e., if the controller is able to block or force the transition), and for each place, whether it is observable or not (i.e., if the controller can determine the number of tokens on it).

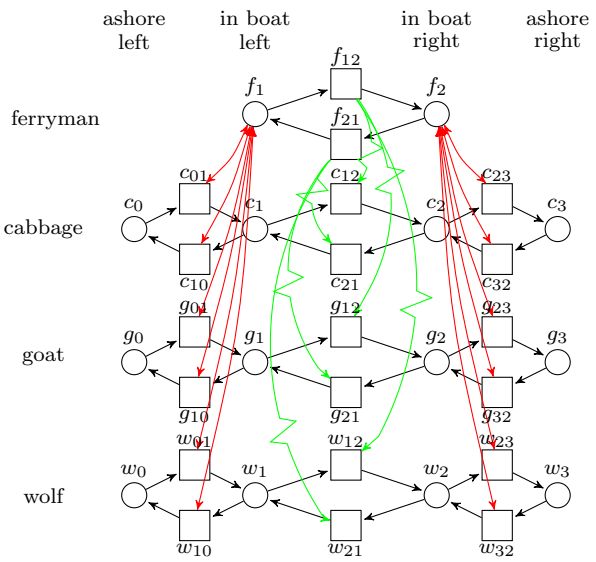

Fig. 2. Modeling of river crossing problem.

As said, both *goal* and *safety specifications* are given by linear constraints on the marking of the net.

2.2 Example: River Crossing

Let us demonstrate the notations and definitions so far on the well-known river crossing problem, which we use as a running example. The augmented Petri net model in Fig. 2 models a ferryman who is due to bring a cabbage, a goat and a wolf from the left shore to the right shore of a river using a boat. The model consists of four parts: The uppermost structure represents the boat which can go from one shore to the other and back. The other three structures represent the three passengers, each of which can reside on the left shore, on the left side in the boat, on the right side in the boat or on the right shore. The test arcs represent the fact, that a passenger can only leave or enter the boat on one side, if the boat currently resides on the corresponding side of the river. The event arcs are used to synchronize the movements of the passengers with those of the boat, i.e., if the boat moves from one shore to the other, a passenger residing inside the boat at this time will move with it. All places have a capacity of 1. The arcs have all a weight of 1 and all transitions are controllable. We introduce the following safety constraints

$$m(c_0) + m(g_0) + m(f_2) \leq 2 \quad (1)$$
$$m(g_0) + m(w_0) + m(f_2) \leq 2 \quad (2)$$
$$m(c_3) + m(g_3) + m(f_1) \leq 2 \quad (3)$$
$$m(g_3) + m(w_3) + m(f_1) \leq 2 \quad (4)$$
$$m(c_1) + m(g_1) + m(w_1) + m(c_2) + m(g_2) + m(w_2) \leq 1 \quad (5)$$

to express that the boat is so small that the ferryman can transport only one of the three objects at a time (Eq. 5), or the boat will sink. Furthermore - as the goat wants to eat the cabbage and the wolf likes to eat the goat - the goat and the cabbage (Eq. 1,3) as well as the goat and the wolf (Eq. 2,4) are not allowed to be left alone on one shore, i.e. they must not remain on one shore, while the boat (and with it the ferryman) moves to the other shore.

2.3 Conversion of the Net

We now give a short overview of the preprocessing of an NCES-like Petri net as introduced in Bollue et al. (2009). By this transformation, the net is converted into a set of so called *unified transitions*, each of which consists of sets of linear inequations describing the preconditions for firing as well as a mapping giving the change of the marking in case of firing. These unified transitions are later used to synthesize a controller.

A *marking change* describes the effect of the firing of a transition. It is defined by a function $c : P \to \mathbb{Z}$ assigning to each place the change of its marking caused by the transition. This means that for each place $p \in P$ with $m(p)$ tokens before firing the transition, there will be $m(p)+c(p)$ tokens on p after firing the transition. We use the notation $C = (c(p_1), \ldots, c(p_n))$ for the list of changes for each place. We write $\mathcal{C}(m, C) = (m(p_1)+c(p_1), \ldots, m(p_n)+c(p_n))$ for the application of a marking change to a marking.

A *marking bound* defines a condition on the marking of a net that must be fulfilled to fire a transition. Such a condition puts an upper bound on the weighted sum of the numbers of tokens on the places. Formally, a marking bound (w, b) consists of a function $w : P \to \mathbb{Z}$ assigning weights to places and an upper bound $b \in \mathbb{Z}$. We use the notation $B = ((p_1, \ldots, p_k), (w_1, \ldots, w_k), b)$ with $p_i \in P$, $w_i \in \mathbb{Z}$ for $1 \leq i \leq k$, $k \in \{1, \ldots, n\}$ and $b \in \mathbb{Z}$ for marking bounds to list the places with non-zero weights, their weights, and the upper bound.

The semantics of marking bounds is given by the evaluation function E_B which assigns to a marking bound $B = ((p_1, \ldots, p_k), (w_1, \ldots, w_k), b)$ and a marking m a truth value as follows:

$$E_B(B, m) = \begin{cases} \textit{true} \text{ if } \sum_{i=1}^{k} (w_i \cdot m(p_i)) \leq b \\ \textit{false} \text{ otherwise.} \end{cases}$$

A *linear marking constraint* $Co = \{B_1, \ldots, B_l\}$ is a set of marking bounds. It is fulfilled for some marking m if all marking bounds are fulfilled for m.

Linear marking constraints are used to define the enabling condition for firing a transition. We write Co^+ to denote that a constraint is positive and needs to be true in a marking to be able to fire the transition, and Co^- for negative constraints that must not be true.

For a transition $t \in T$, a *unified transition* $t^U = (Co_t^+, \{Co_1^-, \ldots, Co_\ell^-\}, C_t)$ specifies that t may fire in a marking if the positive linear marking constraint Co_t^+ is true and all negative linear marking constraints from the set $\{Co_1^-, \ldots, Co_\ell^-\}$ are false. The third component C_t specifies the effect of t by a marking change.

Now we show how to compute for a transition t the unified transition t^U starting by Co_t^+. We first assume that t is not connected to another transition by an event arc:

- For a flow arc coming from a place p, add the marking bound
$$B_{flow+} = -m(p) \leq -wgt(t).$$
- For a flow arc going to a place p, add the marking bound
$$B_{flow-} = m(p) \leq cap(p) - wgt(t).$$
- For a test arc coming from place p, add the marking bound
$$B_{test} = -m(p) \leq -wgt(t).$$
- For an inhibitor arc coming from place p, add the marking bound
$$B_{inhib} = m(p) \leq wgt(t) - 1.$$

The marking change induced by firing the transition t is defined by C_t which is computed as follows:

- For each flow arc from a place p to t add $c(p) = -wgt(t)$ to C_t.
- For each flow arc from t to a place p add $c(p) = wgt(t)$ to C_t.

So for each transition t without adjacent event arcs, the unified transition t^U is defined by $(Co_t^+, \emptyset, C_t)$. The unified transition t^U is marked controllable or uncontrollable according to the controllability of t.

In the case of transitions with adjacent event arcs, a more advanced procedure is necessary. Due to the restrictions on the use of event arcs (see above), we can order the event arcs of the net into a set of trees. For this, consider the transitions with incoming and outgoing event arcs as the nodes. Each transition which has only outgoing event arcs is a root of one tree. The transition, where the event arc ends, becomes a son of the transition where it starts. Now consider one of the trees, for example as in Fig. 3.

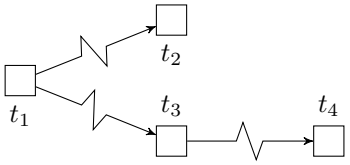

Fig. 3. Example for elimination of event arcs.

There are now several possibilities which transitions can fire together, depending on the specific pre and post conditions of the involved transitions. In this example, there are the following possibilities: $\{t_1\}$, $\{t_1, t_2\}$, $\{t_1, t_3\}$, $\{t_1, t_2, t_3\}$, $\{t_1, t_3, t_4\}$ and $\{t_1, t_2, t_3, t_4\}$. For each of these possibilities, a new unified transition is added, where Co^+ consists of all linear marking bounds of the transitions which should fire and C is the sum of their marking changes. Furthermore, it has to be ensured that the combination of transitions, which are fired synchronously, is always maximal, i.e. that a group of transitions can only be fired, if there are no more transitions anebled which are connected by event arcsd, because in this case - according to the semantics of the event arcs - these should also fire synchronously. To achieve this, we add the preconditions of each transition, that could have fired synchronously according to the event arcs, as additional negative constraints (cf. Thieme and Lueder (1999)). Considering a tree of event arcs as described above, this means that on each branch, the first transition not contained in the currently considered combination induces a negative constraint. Consider for example the set $\{t_1, t_3\}$ then we have to define the unified transition $t_{1,3}^U = (Co_{1,3}^+, \{Co_2^-, Co_4^-\}, C_{1,3})$ where $Co_{1,3}^+$ contains all marking constraints of t_1 and t_3 (computed like above), Co_2^- is the linear marking con-

straint of t_2, Co_4^- is the linear marking constraint of t_4 and $C_{1,3}$ contains the marking changes of t_1 and t_3.

To consider safety constraints on controllable transitions [1], for each unified transition, from the linear marking constraint implementing the safety constraints we subtract the marking change of the transition and conjunct the resulting constraint with the positive constraint of the transition (cf. Hanisch et al. (1998)). That means that for a safety marking bound $B = ((p_1, \ldots, p_n), (w_1, \ldots, w_n), b)$ and a unified transition $t^U = (Co^+, \{Co_1^-, \ldots, Co_n^-\}, C)$ a new marking bound

$$B' = ((p_1, \ldots, p_n)(w_1, \ldots, w_n), b - \sum_{i=1}^{n} w_i \cdot c(p_i))$$

is added to Co^+. This means that every unified transition is enhanced by a new precondition, which ensures that the safety constraints will still be met after firing the transition.

We will now come back to our example of the river crossing problem and give the unified transition $t^U_{f_{12}c_{12}} = (Co^+_{fc}, \{Co^-_g, Co^-_w\}, C_{fc})$ for the transitions f_{12} and c_{12}, which mean that the ferryman brings the cabbage from the left shore to the right shore.

$$\begin{array}{ll} Co^+_{fc} & = \{-m(f_1) \leq -1,\, m(f_2) \leq 0,\, -m(c_1) \leq -1,\, m(c_2) \leq 0\} \\ Co^-_g & = \{-m(g_1) \leq -1,\, m(g_2) \leq 0\} \\ Co^-_w & = \{-m(w_1) \leq -1,\, m(w_2) \leq 0\}\} \\ C_{fc} & = \{c(f_1) = -1,\, c(f_2) = 1,\, c(c_1) = -1,\, c(c_2) = 1\} \end{array}$$

When taking the safety constraints also into account the positve marking constraint has to be extended as follows:

$$\begin{array}{ll} Co^+ = & \{-m(f_1) \leq -1,\, m(f_2) \leq 0,\, -m(c_1) \leq -1,\, m(c_2) \leq 0, \\ & m(c_0) + m(g_0) + m(f_2) \leq 1,\, m(g_0) + m(w_0) + m(f_2) \leq 1, \\ & m(c_3) + m(g_3) + m(f_1) \leq 3,\, m(g_3) + m(w_3) + m(f_1) \leq 3, \\ & m(c_1) + m(g_1) + m(w_1) + m(c_2) + m(g_2) + m(w_2) \leq 1\} \end{array}$$

2.4 Synthesis Algorithm

The basic idea of the algorithm is as follows: To find a sequence of transitions leading from the current marking to a marking fulfilling the goal constraints, we choose a transition, which can be considered to be helpful to reach the goal. For each such transition, there are now two subproblems: On the one hand, a way must be found to enable the chosen transition, on the other hand, this transition was only somehow helpful, i.e. the goal is not necessarily achieved after firing it. So a way has to be found from the resulting marking to the goal. For both these tasks, the same procedure is used recursively.

After each firing of a transition, two additional measures are taken: Firstly the reachability graph is computed, firing uncontrollable transitions only. If there is more than one possible outcome, a branching is added to the control algorithm. In this case, the reachable markings have to be distinguishable by only regarding observable places. Secondly, to synchronize the controller with the plant, if an observable place has been influenced by the firing of the transition (or by the subsequent firing of an uncontrollable transition), a command is added to the control algorithm to wait until the assumed marking change is read back from the plant.

[1] At the moment, uncontrollable transitions, which might lead to forbidden markings, have to be treated separately.

First, we define, which transitions we consider as 'helpful'. For this, we introduce the terms *productive* and *adverse*.

A unified transition $t^U = (.,.,C)$ is called *productive in a marking m w.r.t. a marking bound $B = ((p_1, \ldots, p_n), (w_1, \ldots, w_n), b)$* iff:

$$\sum_{i=1}^{n}(w_i \cdot m(p_i)) > b \text{ and } \sum_{i=1}^{n}(w_i \cdot (m(p_i) + c(p_i))) < \sum_{i+1}^{n}(w_i \cdot m(p_i))$$

respectively t^U is called *adverse* in m w.r.t. B iff:

$$\sum_{i=1}^{n}(w_i \cdot m(p_i)) \leq b \text{ and } \sum_{i=1}^{n}(w_i \cdot (m(p_i) + c(p_i))) > \sum_{i+1}^{n}(w_i \cdot m(p_i)).$$

This means that a transition t^U is productive, if the marking bound is not yet fulfilled in the current marking m and if after firing t^U the resulting marking is at least closer to fulfill the marking bound.

A unified transition t^U is called *productive/adverse in a marking m w.r.t. a linear marking constraint $Co = \{B_1, \ldots, B_k\}$*, iff t^U is productive/adverse in m w.r.t. at least one B_i.

A unified transition t^U is called *productive/adverse in a marking m w.r.t. another unified transition $s^U = (Co^+_s, \{Co^-_{s,1}, \ldots, Co^-_{s,\ell}\}, C_s)$* iff t^U is

- productive/adverse in m w.r.t. Co^+_s or
- adverse/productive in m w.r.t. at least one $Co^-_{s,i}$.

Before giving the synthesis algorithm (see function achieveGoal()) we will first introduce some notations. We will note by $T^E(m)$ the set of unified transitions enabled in marking m and by $T^P(m, Co)$ the set of unified transitions productive in marking m for the goal constraint Co. A goal is a set $g = (\tilde{g}, \mathcal{F})$ where $\tilde{g}$ is either a transition which has to be enabled or a set $\{B_1, \ldots, B_\ell\}$ of linear marking constraints which have to be fulfilled and $\mathcal{F}$ is a set of transitions which are forbidden, which means that in the algorithm they should not be considered as productive to avoid cycling. We initialize $\mathcal{F}$ by the empty set. To describe the output of the algorithm, i.e. the synthesized controller we introduce the data structure PathSet which can be considered as a tree. All paths contain the information how to get from the initial marking to some marking fulfilling the conditions of the goal. The nodes of the tree are markings. The edges can be labeled by either a unified transition t^u which should be fired, a WAIT$_p$-instruction for an observable place p, an information that an uncontrollable transition firing leads to the resulting marking, or a branching into two or more subpaths, tagged either by a list of alternatives or as "if-then-else"-branching with an observable place as condition. We note by $leaves : PathSet \rightarrow Pow(m)$ a function that returns the set of all markings in the leaves of the PathSet. The set $\mathcal{M}$ contains already considered markings to avoid cycling and is initialized as the set which contains only the initial marking.

Function achieveGoal(Goal $g = (\tilde{g}, \mathcal{F})$, start marking m, set of markings $\mathcal{M}$)

```
if g == true then
    Return empty PathSet resultPathSet;
else
    pathSet1 := empty PathSet;
    foreach t^U ∈ T^P(m, g̃) ∧ t^U ∉ F do
        g' := (t^U, F ∪ {t^U});
        pathSet1 := achieveGoal(g', m, M);
        if pathSet1 ≠ Failure then
            foreach m_i ∈ leaves(pathSet1) do
                Compute reachability graph only firing
                uncontrollable transitions;
                Append the graph to the leaf m_i and introduce
                branching if necessary;
                foreach observable place p influenced by t^U or an
                uncontrollable transition do
                    Append a WAIT-instruction WAIT_p to
                    pathSet1;
                end
            end
            if m_i ∈ M then
                Return Failure;
            else
                pathSet2:=achieveGoal(g, m_i, M ∪ {m_i});
                if pathSet2 ≠ Failure then
                    resultPathSet := concatenate pathSet1 and
                    pathSet2;
                end
            end
        end
    end
end
if At least one solution was found then
    Return resulPathSet;
else
    Return Failure;
end
```

3. SIMPLIFICATION BY SATISFIABILITY CHECKING

In the previous Section 2 we described an approach to generate unified transitions and to make use of them to determine productive transitions for the controller synthesis. We applied the method to different case studies to test its applicability. Thereby we made the observation that there are some sources of inefficiency.

Firstly, the method often generates a large number of unified transitions for event arcs, since the procedure computes a unified transition for each possible combination for synchronization. However, some of them are contradictory and correspond to transitions that can never be enabled.

Secondly, the unified transitions may contain a large number of linear marking constraints, especially in the presence of safety constraints. However, the unified transitions often contain subsumed conditions, i.e., conditions that are implied by some other conditions in the unified transition.

Since the efficiency of the controller synthesis algorithm strongly depends on the number and size of the unified transitions, we propose optimizations, based on the above observations, to reduce the number and the size of the unified transitions.

3.1 Reducing the Number of Transitions

As mentioned above, some of the unified transitions computed for event arcs may be contradictory. In our running example, the transition for moving the ferry from the left to the right shore is connected with event arcs to the transitions for transporting the cabbage, the goat, and the wolf to the right shore. Given the safety condition that there is only one place besides the ferryman in the ferry, at most one of the three connected transitions may synchronize. All other cases correspond to transitions that can never be enabled.

The enabledness of a transition can be formalized as a Boolean combination of integer linear constraints, i.e., by a formula in the logic of integer linear arithmetic. By checking the *satisfiability* of such formulas we can decide if the transition can ever fire.

For example, the combination of synchronizing the movement of the ferryman with the movement of the cabbage and the goat from the left to the right would induce a integer linear arithmetic formula containing the conjunction

$$m(c_1) \geq 1 \quad \wedge \quad m(g_1) \geq 1 \quad \wedge$$
$$m(c_1) + m(g_1) + m(w_1) + m(c_2) + m(g_2) + m(w_2) \leq 1$$

for the cabbage and the goat being on the left shore and the safety requirement, which are contradicting.

There exist efficient decision procedures to check the satisfiability of such integer linear arithmetic formulas. We choose the Yices tool for this purpose (cf. Dutertre and de Moura (2006)), available as a free binary download including a library for the embedding of Yices in programs.

For the ferryman moving from the left to the right shore there are 8 possible combinations for synchronization. However, due to the safety requirement, only 4 combinations can be enabled: either the ferryman moves alone or with one of the three possible synchronization partners. The movement from the right to the left is analogous. Thus we can reduce the number of unified transitions for transitions with event arcs from 16 to 8. The running times of the satisfiability checks amounted to 0.028 seconds on a machine with a 2.5 GHz dual core processor.

3.2 Reducing the Size of Unified Transitions

The automatically generated unified transitions may contain constraints that are subsumed (implied) by the remaining constraints. We again use Yices to detect and remove such constraints from the unified transitions.

Let $t^U = (Co^+, \{Co_1^-, \ldots, Co_\ell^-\}, C)$ be a unified transition that is not removed by the above procedure. The corresponding integer linear arithmetic enabledness formula is a conjunction

$$(\bigwedge_{B \in Co^+} B) \wedge (\bigwedge_{i=1,\ldots,\ell} \neg(\wedge_{B \in Co_i^-} B)).$$

To determine if a term in the conjunction is subsumed, we can negate that term and check the resulting formula for satisfiability. If the formula is unsatisfiable, then the term is subsumed, otherwise it is not subsumed. We process this check for each term in the formula, whereas in case of satisfiability we keep the term, and in case of unsatisfiability we remove the term from the conjunction.

In our running example, the enabledness of transporting the cabbage from the left to the right shore is described by the conjunction of 11 terms:

$$\begin{array}{lll} m(f_1) \geq 1 & \wedge\ m(f_2) \leq 0 & \wedge \\ m(c_1) \geq 1 & \wedge\ m(c_2) \leq 0 & \wedge \\ \neg(m(g_1) \geq 1 \wedge m(g_2) \leq 0) & \wedge\ \neg(m(w_1) \geq 1 \wedge m(w_2) \leq 0) & \wedge \\ m(c_0) + m(g_0) + m(f_2) \leq 1 & \wedge\ m(g_0) + m(w_0) + m(f_2) \leq 1 & \wedge \\ m(c_3) + m(g_3) + m(f_1) \leq 3 & \wedge\ m(g_3) + m(w_3) + m(f_1) \leq 3 & \wedge \\ m(c_1) + m(g_1) + m(w_1) + m(c_2) + m(g_2) + m(w_2) \leq 1. & & \end{array}$$

We check for each of the 11 terms (in the order of their occurrence) if they are subsumed, under the assumption that all markings are non-negative and that the safety requirement holds before firing. We always remove subsumed terms before we continue with the next term. We detect 6 subsumed terms, yielding the reduced equivalent formula:

$$\begin{array}{llll} m(f_1) \geq 1 & \wedge & m(f_2) \leq 0 & \wedge \quad m(c_1) \geq 1 \wedge \\ m(c_0) + m(g_0) + m(f_2) \leq 1 & \wedge & m(g_0) + m(w_0) + m(f_2) \leq 1. & \end{array}$$

We could remove altogether 94 of 182 conjunctive terms in all transitions. The satisfiability checks needed 0.065 seconds on a machine with a 2.5 GHz dual core processor.

There is a possibility to further reduce the size of the unified transitions (which we did not implement yet): The negative conjunctive components $\neg(\wedge_{B \in Co_i^-} B)$, i.e., $\vee_{B \in Co_i^-} \neg B$, may refer to integer linear constraints which cannot be satisfied in the context of the transition, and thus can be removed.

4. STRUCTURAL SPEED UP OF THE ALGORITHM

4.1 Quantifying Productivity

The algorithm given above processes every productive unified transition in every step and thus computes all possible paths leading to a marking fulfilling the goal constraints. Although this is a good basis for further research and furthermore assures the optimality of the solution (as a shortest path can be chosen from all returned paths), it is not practical for models which are not very well suited for the presented approach, because the complexity of the procedure can even exceed the one of computing the reachability graph.

A way to significantly increase the efficiency of the algorithm is to drop the requirement to find every possible solution, but only try to find one (good or optimal) solution. If not all transitions should be processed in each step, we have to define some criterion for choosing which transition to try first. To obtain a heuristic rating of the productive transitions, we use two values:

- The first value is the quantification of the transition's productivity (or the *gain* G). We do not only consider *if* the transition's marking change shortens the distance to the goal constraints, but also by *how far*. This value can be negative even for a productive transition, as it might shorten the distance to one marking bound (and thus be productive by the definition given in Section 2), but also lengthen the distance to several other marking bounds.

  The gain $G^+(t^U, m, B)$ of a unified transition $t^U = (.,.,C)$ in a marking m w.r.t. a marking bound $B = ((p_1, \ldots, p_n), (w_1, \ldots, w_n), b)$ of a positive constraint can be computed by

$$\begin{aligned} G^+(t^U, m, B) := \ & \max(b, \sum_{i=1}^{n}(w_i \cdot m(p_i))) \\ & - \max(b, \sum_{i=1}^{n}(w_i \cdot (m(p_i) + c(p_i)))). \end{aligned}$$

  The gain $G^-(t^U, m, B)$ w.r.t. a marking bound $B = ((p_1, \ldots, p_n), (w_1, \ldots, w_n), b)$ of a negative constraint is computed accordingly:

$$\begin{aligned} G^-(t^U, m, B) := \ & \min(b+1, \sum_{i=1}^{n}(w_i \cdot m(p_i))) \\ & - \min(b+1, \sum_{i=1}^{n}(w_i \cdot (m(p_i) + c(p_i)))). \end{aligned}$$

  The gain w.r.t. a marking constraint is obviously the sum of the gains w.r.t. its marking bounds. The gain of a unified transition $t_1^U = (.,.,C)$ w.r.t. the enabling of a unified transition $t_2^U = (Co^+, \{Co_1^-, \ldots, Co_\ell^-\}, .)$ results in

$$G(t_1^U, m, t_2^U) := G^+(t_1^U, m, Co^+) + \sum_{i=1}^{\ell}(G^-(t_1^U, m, Co_i^-)).$$

- The second value is the distance between the current marking and the preconditions of the transition. The distance $d^+(m, B)$ of a marking m to a marking bound $B = ((p_1, \ldots, p_n), (w_1, \ldots, w_n), b)$ of a positive constraint can be computed by

$$d^+(m, B) := max(0, \sum_{i=1}^{n}(w_i \cdot m(p_i)) \ - \ b)$$

  Accordingly, the distance $d^-(m, B)$ of a marking m to a marking bound $B = ((p_1, \ldots, p_n), (w_1, \ldots, w_n), b)$ of a negative constraint can be computed by:

$$d^-(m, B) := max(0, b + 1 \ - \ \sum_{i=1}^{n}(w_i \cdot m(p_i)))$$

  In case of a positive constraint, the distance of a marking to the constraint is the sum of the distances to the marking bound, while the distance to a negative constraint is the minimum of the distances to the marking bounds, as only one bound has to be violated to make the whole constraint false. The distance of a marking to the enabling of a unified transition $t^U = (Co^+, \{Co_1^-, \ldots, Co_\ell^-\}, C)$ results in

$$d(m, t^U) := d^+(m, Co^+) + \sum_{i=1}^{\ell}(d^-(m, Co_i^-)).$$

  We use this value as an estimation for the effort necessary to enable the transition.

With these two values, we now have a heuristic for how efficient it will be to try a certain transition: A transition which has a great gain w.r.t. the goal to be achieved and which already is (or probably can easily be) enabled in the start marking, has a higher potential to help buidling a path from the start marking to the goal than another transition which is hard to enable and only provides a small gain w.r.t. the goal.

Using this heuristic, we give a revised synthesis algorithm [2] (see function achieveGoal2()). As we no longer

[2] For simplification, in the algorithm shown here several details such as an additional backtracking technique as well as the handling of

need the propagation of forbidden transitions, the goal now only contains either a marking constraint which has to be fulfilled or a unified transition which has to be enabled. In every step, the unified transition with the best estimated value is processed. Only if either this transition cannot be enabled or after firing it, there is no way to fulfill the goal, in a backtracking step the next transition is processed. Otherwise, the found solution is returned and no other transition tried.

Function achieveGoal2(Goal g, start marking m)

```
if g == true then
    Return empty PathSet resultPathSet;
else
    pathSet1 := empty PathSet;
    F := ∅;
    while T^P(m, g) \ F ≠ ∅ do
        Choose t^U ∈ T^P(m, g) \ F with maximal
        (G(t^U, m, g) − d(m, t^U));
        g' := (t^U);
        pathSet1 := achieveGoal2(g', m, M);
        if pathSet1 ≠ Failure then
            m' :=leaf(pathSet1);
            pathSet2:=achieveGoal2(g, m');
            if pathSet2 ≠ Failure then
                resultPathSet := concatenate pathSet1, t^U and
                pathSet2;
                Return resultPathSet;
            else
                F := F ∪ {t^U}
            end
        else
            F := F ∪ {t^U}
        end
    end
    Return Failure;
end
```

4.2 Bounding the Recursion Depth

The algorithm described in 4.1 leads to a depth-first search for a suitable path. By introducing a bound on the recursion depth, a balance can be made between a pure depth-first search and a breadth-first search. The best performance is achieved, if the bound is the smallest one, under which the algorithm still finds a solution.

A possible procedure is to start with a low bound and rerun the algorithm several times, each time increasing the bound, until a solution is found.

4.3 Reusing Already Found Partial Solutions

Often, the same or similar tasks (i.e. goals and start markings) have to be processed several times during the run of the algorithm. This can be due to backtracking or to the fact, that the same control actions have to be performed more than once to achieve a certain goal. Thus the efficiency of the algorithm can be further improved by saving already found solutions and reusing them wherever possible.

Base upon the improved algorithm introduced in 4.1, this can easily be achieved by introducing meta transitions. A meta transition includes a whole sequence of unified transitions. Its marking change is equal to the sum of marking changes of the contained transitions. The preconditions of the meta transition consist of the conjunctions the respective preconditions of the included unified transitions, from each of which the marking changes of the unified transitions firing earlier in the sequence are subtracted. So the preconditions of the meta transition ensure that each contained unified transition is enabled at the time it should fire according to the predetermined sequence. The meta transitions created during the execution of the algorithm can simply be added to the set of unified transitions and handled in the same way. By this approach, already found partial solutions can be reused even with different start markings, as long as these still fulfill the preconditions of the sequence.

At the moment, this approach has two drawbacks:

- In the presence of uncontrollable transitions, the preconditions of these have to be added as negative constraints in every step to assure that an uncontrollable transition that is not included in the sequence cannot be enabled during its application. Though this is possible, it makes the preconditions of the meta transitions very complicated und thus reduce the benefit of their use.
- Partial solutions which include branchings cannot easily be stored as meta transitions.

5. RESULTS

5.1 Application to the River Crossing Example

As can easily be seen, our running example is very badly suited for the presented approach: Instead of a linear structure of the 'plant' model which is able to guide the algorithm, a correct solution includes moving boat and passengers back and forth, always partially undoing earlier changes. On the other hand the reachability graph of the net is relatively small, thus other approaches might be better in this case.

Obviously there are two optimal control algorithms to get all three passengers to the right shore without violating the safety specifications: First the goat has to be moved to the right side. Afterwards, the ferryman has to go back alone and fetch one of the other two passengers. Now the goat has to be moved back to the left shore and the last passenger has to be taken over to the right side. At last, the ferryman has to go to the left shore again alone to fetch the goat.

Despite the above-mentioned problems, the improved algorithm - given the requirement of at least one token each on the places c_3, g_3 and w_3 as goal constraints - finds one of these two solutions in a very short time. The following list sums up our experimental findings:

- The reachability graph of the example net contains 54 nodes. During its computation 166 transitions fire.
- Without the structural improvements presented in Section 4, the algorithms runs too long to wait for the termination.

uncontrolled transitions are omitted. Thus we consider only a single end marking (leaf) for each PathSet.

- Without the elimination of transitions and constraints as described in Section 3, but with the structural improvements enabled, the algorithm finds an optimal solution within 11774 calls to the recursive function, firing 3022 transitions (time: ≈ 700 ms).
- With all improvements enabled, the algorithm finds an optimal solution within 539 calls to the recursive function, firing 186 transitions (time: ≈ 25 ms).

5.2 Application to a Real Plant

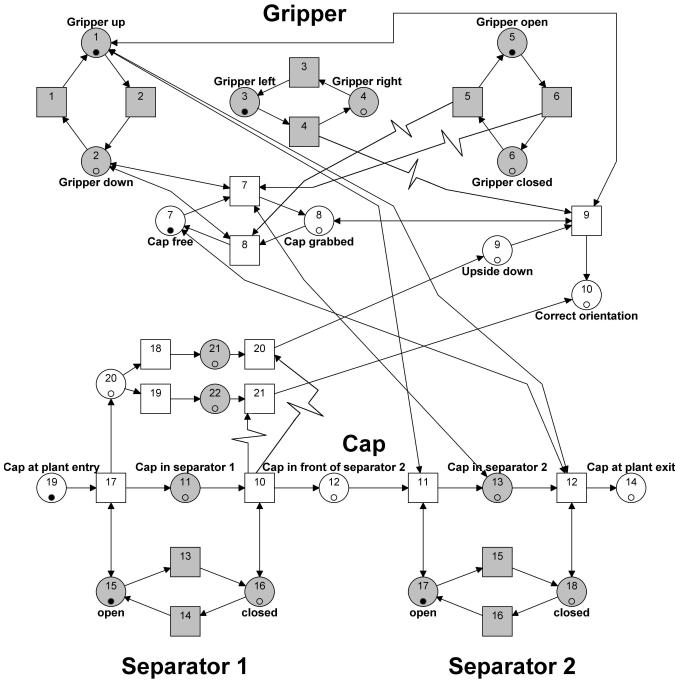

Fig. 4. Plant model of a cap turning plant

We shortly revisit the example of a real plant given in Bollue et al. (2009) (see Fig. 4): A cap has to be turned by a gripper. The controller has to operate two separators and the three degrees of freedom of the gripper. A correct solution includes placing the cap under the gripper with the help of the separators, moving the gripper down, closing it, moving it up, turning it, moving it down again, releasing the cap and moving the gripper up.

Due to the many combinations of positions of the different parts, the reachability graph of the model is relatively big: It contains 864 nodes. While computing the graph, 4798 transition firings occur. The presented algorithm finds an optimal solution [3] within 64 calls to the recursive function, firing 25 transitions (time: ≈ 2 ms).

6. SUMMARY AND OUTLOOK

We have introduced several improvements to the algorithm presented in Bollue et al. (2009). On the one hand, the elimination of transitions which cannot be enabled, and the deletion of unnecessary marking bounds during the preprocessing (both through satisfiability checking of integer linear inequations) lead to a great gain in performance. On the other hand, structural modifications of the algorithm itself severely reduce the average case complexity.

Applying the named improvements, we showed, that the algorithm is now even applicable to examples which are badly suited for the approach - though other methods might still be better here. For typical plant models providing a good-natured structure, which can be exploited by the algorithm, it shows a very good performance - especially compared to the effort necessary to compute the whole reachability graph of the net.

Future improvements may include:

- the application of satisfiability checking at several points in the algorithm to detect and ignore conditions which can never be fulfilled,
- more sophisticated heuristics for the choice of the unified transitions to be processed first or
- a more advanced way of reusing already found solutions, also in the presence of branchings and uncontrollable transitions and
- the completion of a ready-to-use implementation in our Petri net tool NETLAB.

[3] Due to technical reasons, the branching induced by the conflicting uncontrollable transitions 18 and 19 was omitted in this test.

REFERENCES

Abel, D. (1990). *Petri-Netze für Ingenieure. Modellbildung und Analyse diskret gesteuerter Systeme.* Springer-Verlag.

Bollue, K., Abel, D., and Thomas, W. (2009). Synthesis of behavioral controllers for discrete event systems with nces-like petri net models. In *Proc. of the European Control Conference 2009 - ECC '09, 23-26 August 2009, Budapest, Hungary*, 4786–4791.

Dutertre, B. and de Moura, L. (2006). A Fast Linear-Arithmetic Solver for DPLL(T). In *Proc. of the 18th Computer-Aided Verification conference*, volume 4144 of *LNCS*, 81–94. Springer-Verlag.

Giua, A. (1992). *Petri Nets as Discrete Event Models for Supervisory Control.* Ph.D. thesis, Rensselaer Polytechnic Inst., Troy, NY, USA.

Hanisch, H.M., Luder, A., and Thieme, J. (1998). A modular plant modeling technique and related controller synthesis problems. *Systems, Man, and Cybernetics, 1998.*, 1, 686–691 vol.1. doi:10.1109/ICSMC.1998.725493.

Moody, J. and Antsaklis, P. (2000). Petri net supervisors for des with uncontrollable and unobservable transitions. *IEEE Transactions on Automatic Control*, 45(3), 462–476.

Orth, Ph., Bollig, A., and Abel, D. (2004). Rapid Control Prototyping diskreter Steuerungen in der Automatisierungstechnik. In *Proc. of SPS/IPC/DRIVES'04*, 143–151.

Pinzon, L.E., Hanisch, H.M., Jafari, M.A., and Boucher, T. (1999). A comparative study of synthesis methods for discrete event controllers. *Formal Methods in System Design*, 15(2), 123–167. URL `citeseer.ist.psu.edu/pinzon97comparative.html`.

Sreenivas, R.S. and Krogh, B.H. (1991). Petri net based models for condition/event systems. In *Proc. of 1991 American Control Conference*, volume 3, 2899–2904.

Thieme, J. and Lueder, A. (1999). Transformation von netzmodellen zur analyse von technischen systemen. Technical report, Universität Magdeburg.